

Fujitsu Software

Technical Computing Suite V4.0L20

Development Studio

C User's Guide

J2UL-2560-01ENZ0(14)
March 2025

Preface

Purpose of This Manual

This manual describes how to use the C language processing system (called "this system" in this manual) intended for the system that has the Fujitsu CPU A64FX installed (called "FX system" in this manual).

It also details the language specifications that are based on the following C language standard, and OpenMP API, and which are expanded under this system.

- American National Standard for Information Systems-Programming Language C, X3.159-1989 (ISO/IEC 9899:1990)

In this manual, this standard is called C89.

Specifications based on this standard are called C89 specifications.

- Programming languages -- C(ISO/IEC 9899:1999)

In this manual, this standard is called C99.

Specifications based on this standard are called C99 specifications.

- Programming languages -- C(ISO/IEC 9899:2011)

In this manual, this standard is called C11.

Specifications based on this standard are called C11 specifications.

- OpenMP Application Program Interface Version 3.1 July 2011

In this manual, specification based on this standard is called OpenMP 3.1.

- OpenMP Application Program Interface Version 4.0 July 2013

In this manual, specification based on this standard is called OpenMP 4.0.

- OpenMP Application Programming Interface Version 4.5 November 2015

In this manual, specification based on this standard is called OpenMP 4.5.

- OpenMP Application Programming Interface Version 5.0 November 2018

In this manual, specification based on this standard is called OpenMP 5.0.

Intended Readers

This manual assumes an understanding of the C language standard. For details, refer to the C language standard or commercial books based on the standard.

This manual was written for people who write C programs and process them using this system. Readers should also know how to use Linux(R) commands, file manipulation and shell programming.

Structure of This Manual

The structure of this manual is as follows:

[Chapter 1 Overview](#)

Describes an overview of how to use this system to process C programs.

[Chapter 2 From Compilation to Execution](#)

Describes the procedures from compiling to executing C programs.

[Chapter 3 Optimization](#)

Describes how to use optimization functions that ensure programs execute as quickly as possible.

[Chapter 4 Multiprocessing](#)

Describes how to use parallelization functions or OpenMP specifications that ensure programs execute as quickly as possible.

[Chapter 5 Emitting Information](#)

Describes how to emit information in this system.

[Chapter 6 Language Specifications](#)

Describes the language specifications that are extended in this system based on the C language standard, and the support status of the language specifications.

[Chapter 7 Notes on Linking with Different Languages and Trad/Clang Modes](#)

Describes notes about linking with object programs in different programming languages and Trad/Clang Modes.

[Chapter 8 Debugging Functions](#)

Describes the program debugging functions in this system.

[Chapter 9 Clang Mode](#)

Describes the mode using a compiler that has been enhanced based on Clang/LLVM.

[Appendix A Implementation-dependent Specifications](#)

Describes the operation of the items defined as processing implementation-dependent in the C language standard.

[Appendix B Compilation Limitations](#)

Lists the compilation limitations in this system.

[Appendix C Restrictions and Notes](#)

Describes the restrictions and notes of this system.

[Appendix D Compatibility with GNU C Specifications](#)

Describes compatibility with GNU C.

[Appendix E Data and Memory Regions](#)

Describes the attributes of data and memory regions in this system.

[Appendix F Code Coverage](#)

Describes the code coverage.

[Appendix G Fujitsu Extended Functions](#)

Describes Fujitsu Extended Functions supported in this system.

[Appendix H Runtime Information Output Function](#)

Describes the Runtime Information Output Function.

[Appendix I Using High-Speed Facility on Job Operation Software](#)

Describes information on the compilation and execution processes for using the high-speed facility on FX system.

[Appendix J Fujitsu OpenMP Library](#)

Describes how to perform parallel processing of a C program using Fujitsu OpenMP libraries.

Notes of This Manual

The code examples of optimization in this manual are conceptual source that complements each explanation of functions. When the code examples are compiled and executed, optimizations may not work as expected. This is because optimizations depend on compilation options and other conditions.

Notation Used in This Manual

Syntax Description Symbols

A syntax description symbol is a symbol that has specific meaning in syntax.

The following syntax symbols are used in this manual.

Symbol name	Symbol	Explanation
Selection symbols	{ }	Only one of the items enclosed in the braces must be selected (items are listed vertically).
		Multiple items are enumerated by this delimiter (items are listed horizontally).
Option symbol	[]	An item enclosed in brackets can be omitted. This symbol includes the meaning of the selection symbol "{ }".
Repeat symbol	...	The item immediately preceding the ellipsis can be specified repeatedly in the syntax.

Parallel

The way of parallelization is thread parallelization unless otherwise described.

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Trademark

- Arm is a registered trademark of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.
- The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board
- Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.
- All other trademarks and product names are the property of their respective owners.
- The trademark notice symbol (TM,(R)) is not necessarily added in the system name and the product name, etc. published in this material.

Date of Publication and Version

Version	Manual code
March 2025, Version 1.14	J2UL-2560-01ENZ0(14)
March 2024, Version 1.13	J2UL-2560-01ENZ0(13)
September 2023, Version 1.12	J2UL-2560-01ENZ0(12)
March 2023, Version 1.11	J2UL-2560-01ENZ0(11)
September 2022, Version 1.10	J2UL-2560-01ENZ0(10)
March 2022, Version 1.9	J2UL-2560-01ENZ0(09)
November 2021, Version 1.8	J2UL-2560-01ENZ0(08)
August 2021, Version 1.7	J2UL-2560-01ENZ0(07)
July 2021, Version 1.6	J2UL-2560-01ENZ0(06)
March 2021, Version 1.5	J2UL-2560-01ENZ0(05)
January 2021, Version 1.4	J2UL-2560-01ENZ0(04)
November 2020, Version 1.3	J2UL-2560-01ENZ0(03)
September 2020, Version 1.2	J2UL-2560-01ENZ0(02)
June 2020, Version 1.1	J2UL-2560-01ENZ0(01)
February 2020, 1st Version	J2UL-2560-01ENZ0(00)

Copyright

Copyright FUJITSU LIMITED 2020-2025

Update History

Changes	Location	Version
Updated the trademarks.	Preface	Version 1.14
Deleted a note.	9.10.2	
Improved the explanation of predefined macros.	6.1.6 9.1.5	Version 1.13
Improved the explanation of the following option: - -x language	9.8.2 D.2	
Added a note.	9.10.2	
Improved the explanation of the following options: - -K{fp_relaxed nofp_relaxed} - -K{preex nopreex} - -N{Rtrap Rnotrap}	2.2.2.5	Version 1.12
Improved the explanation of the following optimization control specifiers: - fission_point - fp_relaxed - nopreex	3.4.1.2	
Improved the explanation of the following options: - -O[n] - -f{fj-fast-matmul fj-no-fast-matmul} - -f{fj-optlib-string fj-no-optlib-string} - -f{lto no-lto}	9.1.2.2.3	
Improved the explanation.	9.1.2.3.2	
Corrected description of supported OpenMP specifications.	1.1 4.3 4.3.2.2 J.3	
Added a note.	1.1	Version 1.11
Corrected the table "Return Values of the Compile Command".	2.1.3	
Corrected the explanation.	8.1.2	
Improved the explanation of the following options: - -f{fj-swp fj-no-swp} - -f{fj-zfill[=N] fj-no-zfill} - -f{lto no-lto}	9.1.2.2.3	
Deleted the following option: - -f{fj-loop-interchange fj-no-loop-interchange}	9.1.2.2.3 9.1.2.3.1	
Added the following optimization control specifiers: - #pragma fj loop swp - #pragma fj loop noswp	9.2.2.1.3	

Changes	Location	Version
Corrected the explanation of the following option: - -fvisibility=internal	9.8.2	
Deleted the following GNU C++ compatible option of Trad Mode: - -fvisibility	D.2	
Improved figures design.	-	
Corrected the explanation.	6.1.6 9.1.5	Version 1.10
Added the following option: - -f{debug-info-for-profiling no-debug-info-for-profiling}	9.1.2.2.1	
Corrected the explanation of the following option: - -f{optimize-sibling-calls no-optimize-sibling-calls}	9.8.2	
Added the section "Restrictions and Notes".	9.10	
Corrected the explanation.	I.1	
Unified notation of "C++ Standard Library".	-	
Improved the explanation.	-	
Improved the explanation of the following options: - -O[n] - -msve-vector-bits={512 scalable} - -f{lto no-lto} - -f{omit-frame-pointer no-omit-frame-pointer} - -f{slp-vectorize no-slp-vectorize}	9.1.2.2.3	Version 1.9
Added the following option: - -m{omit-leaf-frame-pointer no-omit-leaf-frame-pointer}	9.1.2.1 9.1.2.2.3	
Added the explanation.	9.1.2.3.1	
Added the section "SIMD".	9.2.3	
Added the section "Restriction of registers with asm keyword and on an inline asm clobber lists".	C.2.11	
Improved the explanation.	-	
Improved the explanation of the following options: - -K{region_extension noregion_extension} - -K{striping[=N] nostriping}	2.2.2.5	Version 1.8
Added some standard library functions to the list of optimization targets of the -Klib option.	2.2.2.5	
Added the section "Intrinsic functions that SIMD Extensions can be applied to".	3.2.7.5	
Improved the explanation of the following optimization control specifiers: - fission_point - striping	3.4.1.2	
Modified the table "Optimization Identifiers that can be specified for parallelization".	4.2.6.1	
Improved the explanation.	-	

Changes	Location	Version
Improved the description of "Predefined Macro Name".	6.1.6 9.1.5	Version 1.7
Added a note about SIMD built-in function documentation.	9.7	
Added a note.	1.2.1	Version 1.6
Added the following option: - <code>-Kopenmp_loop_variable={private standard}</code>	2.2.2.3 2.2.2.5	
Changed the <code>-K{array_declaration_opt noarray_declaration_opt}</code> option default to the <code>-Knoarray_declaration_opt</code> .	2.2.2.5	
Added the following environment variable: - <code>FCOMP_LINK_FJOB</code>	2.3	
Added the explanation.	3.2.7.4	
Improved the explanation.	3.3.9	
Changed the title.	Chapter 7 9.6	
Added the section "Compile Commands and Required Options at Linking".	7.1	
Added the following options: - <code>-f{fj-regalloc-using-latency fj-no-regalloc-using-latency}</code> - <code>-f{fj-promote-licm-addressing fj-no-promote-licm-addressing}</code> - <code>-f{fj-sched-insn-contiguous fj-no-sched-insn-contiguous}</code>	9.1.2.2.3	
Added a note.	9.4.1.2.4	
Added the section "Link Error (undefined reference to)".	C.2.10	
Deleted a note.	2.1.2	Version 1.5
Improved the explanation about the zfill optimization.	2.2.2.5 3.3.5 3.4.1.2	
Improved the explanation of the following options: - <code>-K{fp_relaxed nofp_relaxed}</code> - <code>-K{ilfunc[={loop procedure}] noilfunc}</code>	2.2.2.5	
Added the following environment variable: - <code>FLIB_L1_SCCR_CNTL</code>	3.5.2.2 I.2 J.4.4	
Added the explanation.	3.6.1	
Improved the explanation about supported specifications.	4.3	
Added the section "Creating Shared Libraries in Clang Mode".	4.3.4.4	
Added the section "Notes on Compilation".	9.1.1.4	
Added the following option: - <code>-msve-vector-bits={512 scalable}</code>	9.1.2.1 9.1.2.2.3	
Added the following options: - <code>-f{fj-interleave-loop-insns[=N] fj-no-interleave-loop-insns}</code> - <code>-f{fj-loop-fission fj-no-loop-fission}</code> - <code>-ffj-loop-fission-threshold=N</code>	9.1.2.2.3 9.1.2.3.1	

Changes	Location	Version
<ul style="list-style-type: none"> - <code>-f{fj-swp fj-no-swp}</code> - <code>-f{fj-zfill[=<i>N</i>] fj-no-zfill}</code> 		
Added an article.	9.1.2.3.4	
Added the section "Notes on Specified SVE Vector Register Size".	9.1.2.3.5	
Added the following optimization control specifiers: <ul style="list-style-type: none"> - <code>#pragma fj loop clone <i>var</i>==<i>n</i></code> - <code>#pragma fj loop loop_fission_target [<i>cl</i>]</code> - <code>#pragma fj loop loop_fission_threshold <i>n</i></code> - <code>#pragma fj loop zfill [<i>N</i>]</code> - <code>#pragma fj loop nozfill</code> 	9.2.2.1.2 9.2.2.1.3	
Added the following optimization control specifiers: <ul style="list-style-type: none"> - <code>#pragma fj loop swp</code> - <code>#pragma fj loop noswp</code> - <code>#pragma clang loop vectorize_width(<i>n</i>, scalable)</code> 	9.2.2.1.2	
Added the following information to the optimization information. <ul style="list-style-type: none"> - Loop fission - Software pipelining - Clone optimization 	9.4.1.2.2	
Changed the operation specification of the sector cache.	I.2 J.4.4	
Improved the explanation.	-	Version 1.4
Added a note.	4.2.2.4 4.3.2.1 J.2.2 J.3.2	
Added a table of commands and required options for interlanguage linking.	Chapter 7 9.6	
Added loop interchange to the optimizations applied by the <code>-O2</code> option or higher.	9.1.2.2.3	
Improved the explanation.	-	
Added a note.	2.1.2	Version 1.3
Added the following options: <ul style="list-style-type: none"> - <code>--linkcoarray</code> - <code>--linkfortran</code> 	2.2.1 2.2.2.4 9.1.2.1 9.1.2.2.4	
Corrected the errors in the explanations of the <code>-N{reordered_variable_stack noreordered_variable_stack}</code> option.	2.2.2.6 E.4	
Improved the explanation about the <code>-N{Rtrap Rnotrap}</code> option.	2.2.2.6	
Improved the explanation about loop unrolling.	3.2.4	
Added examples of compiling and linking.	7.2 9.6.2	
Corrected the errors in the explanations of detected errors.	8.2.1	

Changes	Location	Version
Corrected the errors in the explanations of the following options. <ul style="list-style-type: none"> - <code>-ffj-lst[={p t}]</code> - <code>-ffj-lst-out=file</code> 	9.1.2.2.1 9.4.1.2	
Added the section "Integer Division Exception when the Divisor is Zero".	C.2.9	
Improved the explanation.	-	
Improved code examples.	3.3.6 3.4.1.2 3.5.2.1 3.5.2.2	Version 1.2
Improved the explanation about the parameters aligned and unaligned for the <code>simd</code> specifier.	3.4.1.2 3.4.1.3	
Added the section "Notes of Wrong Erroneous Program".	3.6.3	
Added the following optimization control specifier: <ul style="list-style-type: none"> - <code>#pragma clang loop vectorize(assume_safety)</code> 	9.2.2.1.2	
Added the section "Changing SVE Vector Register Size".	C.2.8	
Improved the explanation.	-	
Added clone optimization to the optimizations applied by the <code>-O3</code> option.	2.2.2.3	Version 1.1
Added the following options: <ul style="list-style-type: none"> - <code>-K{array_declaration_opt noarray_declaration_opt}</code> - <code>-K{extract_stride_store noextract_stride_store}</code> - <code>-K{fp_precision nofp_precision}</code> - <code>-Kswp_policy={auto small large}</code> 	2.2.2.3 2.2.2.5	
Added the following environment variable: <ul style="list-style-type: none"> - <code>FLIB_TRACEBACK_MEM_SIZE</code> 	2.5.1	
Added the following optimization control specifiers: <ul style="list-style-type: none"> - <code>[no]array_declaration_opt</code> - <code>[no]extract_stride_store</code> - <code>[no]fullunroll_pre_simd [n]</code> - <code>swp_policy {auto small large}</code> 	3.4.1.2	
Added the following option: <ul style="list-style-type: none"> - <code>-f{fj-fp-precision fj-no-fp-precision}</code> 	9.1.2.2.3	
Added "Information That Relates To The Register" to the optimization information.	9.4.1.2.2	
Improved the explanation.	-	

All rights reserved.
The information in this manual is subject to change without notice.

Contents

Chapter 1 Overview.....	1
1.1 Configuration of the C Language Processing System.....	1
1.2 How to Use.....	2
1.2.1 Preparation.....	2
1.2.2 Compilation and Linking.....	3
1.2.3 Debugging.....	3
1.2.4 Tuning.....	3
Chapter 2 From Compilation to Execution.....	4
2.1 Compile Command.....	4
2.1.1 Syntax of the Compile Command.....	4
2.1.2 Input Files for the Compile Command.....	4
2.1.3 Return Value of the Compile Command.....	4
2.2 Compiler Options.....	5
2.2.1 Syntax of Compiler Options.....	5
2.2.2 Description of Compiler Options.....	5
2.2.2.1 General Options for Compiler.....	5
2.2.2.2 Options for Messages.....	10
2.2.2.3 Options for Optimization.....	10
2.2.2.4 Options for Language Specifications.....	12
2.2.2.5 -K Option.....	13
2.2.2.6 -N Option.....	32
2.2.3 Notes for Using Compiler Options.....	37
2.3 Environment Variable for Compile Command.....	38
2.4 Compilation Profile File.....	40
2.5 Procedure of Execution.....	41
2.5.1 Environment Variable for Execution.....	41
2.5.2 Notes for Execution.....	42
2.5.2.1 Variable Allocation at Execution.....	43
Chapter 3 Optimization.....	44
3.1 Overview of Optimization.....	44
3.2 Standard Optimization.....	44
3.2.1 Elimination of Common Expressions.....	45
3.2.2 Movement of Invariant Expressions.....	45
3.2.3 Reducing Strength of Operators.....	45
3.2.4 Loop Unrolling.....	46
3.2.5 Loop Blocking.....	47
3.2.6 Software Pipelining.....	48
3.2.7 SIMD.....	49
3.2.7.1 Normal SIMD.....	49
3.2.7.2 SIMD Extension for Loop Containing "if" Statement.....	49
3.2.7.3 List Vector Conversion.....	49
3.2.7.4 SIMD with Redundant Executions for the SIMD Width.....	49
3.2.7.5 Math functions that SIMD Extensions can be applied to	51
3.2.8 Loop Unswitching.....	51
3.2.9 Inline Expansion.....	51
3.3 Extended Optimization.....	51
3.3.1 Optimization by Modifying Evaluation Methods.....	52
3.3.1.1 Advance Evaluation of Invariant Expressions.....	52
3.3.1.2 Arithmetic Evaluation Method Modification.....	52
3.3.2 Optimization of Pointers.....	53
3.3.3 Multi-Operation Function.....	54
3.3.3.1 About Calling of Multi-Operation Functions.....	54
3.3.3.2 Effects of Compiler Option -Kmfunc=3.....	57
3.3.4 Loop Striping.....	58

3.3.5 zfill.....	58
3.3.6 Loop Versioning.....	59
3.3.7 Clone Optimization.....	59
3.3.8 Unroll-and-Jam.....	60
3.3.9 Tree-Height-Reduction Optimization.....	60
3.3.10 Loop Fission.....	61
3.3.10.1 Strip-Mining.....	62
3.3.11 Strict Aliasing.....	63
3.4 Using Optimization Functions.....	64
3.4.1 Using Optimization Control Line (OCL).....	64
3.4.1.1 Types of Optimization Control Lines.....	64
3.4.1.2 Optimization Control Specifier.....	65
3.4.1.3 Notes for Optimization Control Specifiers.....	102
3.5 Software Control of Sector Cache.....	103
3.5.1 Use of Sector Cache.....	103
3.5.2 Controlling Sector Cache via Software.....	103
3.5.2.1 Software Control with Optimization Control Lines.....	104
3.5.2.2 Software Control with Environment Variables and Optimization Control Line.....	105
3.5.2.3 Behavior when an Exceptional Value Is Specified.....	106
3.6 Notes.....	106
3.6.1 Side Effect of Optimizations for Floating-Point Operation.....	106
3.6.2 Notes on Specified SVE Vector Register Size.....	108
3.6.3 Notes of Wrong Erroneous Program.....	108
Chapter 4 Multiprocessing.....	109
4.1 Overview of Multiprocessing.....	109
4.1.1 What is Multiprocessing?.....	109
4.1.2 Effect of Multiprocessing.....	110
4.1.3 Requirements for Effective Multiprocessing.....	110
4.1.4 Parallelization.....	110
4.2 Automatic Parallelization.....	110
4.2.1 Compilation.....	111
4.2.1.1 Compiler Option for Automatic Parallelization.....	111
4.2.2 Execution Process.....	111
4.2.2.1 Number of Threads.....	111
4.2.2.2 Stack Size on Execution.....	111
4.2.2.3 Synchronization Process.....	111
4.2.2.4 CPU Binding for Thread.....	112
4.2.3 Example of Compilation and Execution.....	114
4.2.4 Performance Tuning.....	115
4.2.5 Feature Details on Automatic Parallelization.....	115
4.2.5.1 Targets for Automatic Parallelization.....	115
4.2.5.2 Loop Slicing.....	115
4.2.5.3 Automatic Loop Slicing by Compiler.....	115
4.2.5.4 Loop Interchange and Automatic Loop Slicing.....	116
4.2.5.5 Loop Distribution and Automatic Loop Slicing.....	116
4.2.5.6 Loop Fusion and Automatic Loop Slicing.....	117
4.2.5.7 Loop Reduction.....	117
4.2.5.8 Restrictions on Loop Slicing.....	118
4.2.5.9 Displaying the State of Automatic Parallelization.....	120
4.2.5.10 Parallel Region Extension.....	120
4.2.5.11 Block Distribution and Cyclic Distribution.....	121
4.2.6 Optimization Control Line.....	121
4.2.6.1 Optimization Control Specifier.....	121
4.2.6.2 Automatic Parallelization and Optimization Control Specifiers.....	123
4.2.6.3 Optimization Control Specifiers for Automatic Parallelization.....	123
4.2.7 Notes on Automatic Parallelization.....	133

4.2.7.1 Multiprocessing of Nested Loops.....	134
4.2.7.2 Side Effect of Using -Kparallel,reduction.....	135
4.2.7.3 Examples of Invalid Usage of an Optimization Control Line.....	135
4.2.7.4 Standard Library Function References.....	137
4.3 Parallelization by OpenMP Specification.....	137
4.3.1 Compilation.....	137
4.3.1.1 Compiler Option for OpenMP C Program.....	138
4.3.1.2 Obtaining of Optimization Information.....	138
4.3.1.3 Restriction of OpenMP programs.....	138
4.3.2 Execution Process.....	139
4.3.2.1 Environment Variable at Execution.....	139
4.3.2.2 Environment Variable for OpenMP Specifications.....	141
4.3.2.3 Notes during Execution.....	142
4.3.3 Implementation-Dependent Specifications.....	143
4.3.4 Notes on OpenMP Programming.....	145
4.3.4.1 Implementation of parallel Region and Explicit task Region.....	145
4.3.4.2 Implementation of threadprivate Variable.....	146
4.3.4.3 Automatic Parallelization for OpenMP Programs.....	146
4.3.4.4 Creating Shared Libraries in Clang Mode.....	146
4.4 Runtime Messages.....	147
Chapter 5 Emitting Information.....	149
5.1 Emitting Information at Compilation.....	149
5.1.1 Header.....	149
5.1.1.1 Output Format.....	149
5.1.2 Source List.....	149
5.1.2.1 Output Format.....	149
5.1.2.2 Information Included in Source List.....	150
5.1.2.2.1 Line Number of Source.....	150
5.1.2.2.2 Symbols for Parallelization.....	150
5.1.2.2.3 Symbols for Inline Expansion.....	151
5.1.2.2.4 Number of Loop Unrolling.....	151
5.1.2.2.5 Symbols for Using SIMD Extensions.....	151
5.1.2.2.6 Details Optimization Information.....	151
5.1.2.3 Example of Source Listing.....	154
5.1.2.4 Notes on Information at Compilation (Source List).....	157
5.1.3 Parallelization Messages.....	157
5.1.4 Statistics List.....	157
5.1.4.1 Output Format.....	158
5.1.4.2 Options Which Are Not Output.....	158
5.1.4.3 Options Which Are Interpreted by Compiler.....	158
5.2 Information at Execution.....	158
5.2.1 Runtime Messages.....	158
5.2.2 Trace Back Information.....	159
Chapter 6 Language Specifications.....	160
6.1 Language Specifications Extended in This System.....	160
6.1.1 long long Type.....	160
6.1.2 The pragma Directive.....	161
6.1.3 The ident Directive.....	161
6.1.4 The assert Directive.....	162
6.1.5 The unassert Directive.....	162
6.1.6 Predefined Macro Names.....	162
6.2 Support Status of Language Specifications.....	164
6.2.1 C99 Specifications.....	164
6.2.2 C11 Specifications.....	165
Chapter 7 Notes on Linking with Different Languages and Trad/Clang Modes.....	166

7.1 Compile Commands and Required Options when Linking.....	166
7.1.1 Linking C++ Trad Mode with C++ Clang Mode.....	170
7.1.2 Linking C++ Clang Mode with C++ Clang Mode.....	171
7.1.3 Linking MPI Object Programs and Use Profiler.....	171
7.1.4 Objects Generated in Clang Mode with -flto Option.....	171
7.2 Linking with C++.....	171
7.3 Linking with Fortran.....	173
Chapter 8 Debugging Functions.....	176
8.1 Functions for Debugging.....	176
8.1.1 Subscript Range Checks (subchk Function).....	176
8.1.2 Heap Memory Checks (heapchk Function).....	176
8.1.2.1 Memory Release Check.....	177
8.1.2.2 Buffer Overrun Check.....	177
8.1.2.3 Memory Leak Check.....	177
8.2 Debugging Programs for Abend.....	177
8.2.1 Causes of Abend.....	177
8.2.2 Information Generated when an Abend Occurs.....	178
8.2.2.1 Information Generated for a General Abend.....	178
8.2.2.2 Information of SIGXCPU.....	179
8.2.2.3 Information when an Abend Occurs Again during Abend Processing.....	179
8.3 Hook Function.....	179
8.3.1 User-Defined Function Format.....	179
8.3.2 Notes on Hook Function.....	180
8.3.3 Calling the User-Defined Function from a Specified Location.....	180
8.3.3.1 Notes on Calling from a Specified Location.....	182
8.3.4 Calling the User-Defined Function at Regular Time Interval.....	182
8.3.4.1 Notes on Calling by Regular Time Interval.....	182
8.3.5 Calling from Any Location in the Program.....	182
Chapter 9 Clang Mode.....	183
9.1 From Compilation to Execution.....	183
9.1.1 Compile Command.....	183
9.1.1.1 Syntax of the Compile Command.....	183
9.1.1.2 Input Files for the Compile Command.....	183
9.1.1.3 Return Value of the Compile Command.....	183
9.1.1.4 Notes on Compilation.....	184
9.1.1.4.1 Stack Size on Compilation.....	184
9.1.2 Compiler Options.....	184
9.1.2.1 Syntax of the Compile Command.....	184
9.1.2.2 Description of Compiler Options.....	185
9.1.2.2.1 General Options for Compiler.....	185
9.1.2.2.2 Options for Messages.....	189
9.1.2.2.3 Options for Optimization.....	190
9.1.2.2.4 Options for Language Specifications.....	199
9.1.2.2.5 Options for CPU/Architecture.....	200
9.1.2.2.6 Options for Code Generation.....	202
9.1.2.3 Notes on Compile Options.....	203
9.1.2.3.1 Correspondences of Compile Options in Clang Mode and Trad Mode.....	203
9.1.2.3.2 Side Effect of Optimizations for Floating-Point Operation.....	206
9.1.2.3.3 Note on Using SVE.....	207
9.1.2.3.4 Notes on Using SIMD Built-in Functions.....	208
9.1.2.3.5 Notes on Specified SVE Vector Register Size.....	208
9.1.3 Environment Variable for Compile Command.....	208
9.1.4 Compilation Profile File.....	209
9.1.5 Predefined Macro Names.....	209
9.1.6 Procedure of Execution.....	211
9.1.6.1 Environment Variable for Execution.....	211

9.1.6.2 Notes for Execution.....	211
9.2 Optimization.....	211
9.2.1 Overview of Optimization.....	211
9.2.2 Using Optimization Functions.....	211
9.2.2.1 Using Optimization Control Line (Pragma Directives).....	211
9.2.2.1.1 Types of Optimization Control Lines.....	212
9.2.2.1.2 Optimization Control Specifier.....	212
9.2.2.1.3 Notes on Optimization Control Lines and Optimization Control Specifiers.....	226
9.2.3 SIMD.....	228
9.2.3.1 Math Functions that SIMD Extensions can be applied to.....	228
9.3 Multiprocessing.....	229
9.3.1 Overview of Multiprocessing.....	229
9.3.2 Parallelization by OpenMP Specification.....	229
9.4 Emitting Information.....	230
9.4.1 Emitting Information at Compilation.....	230
9.4.1.1 Header.....	230
9.4.1.2 Source List.....	230
9.4.1.2.1 Output Format.....	230
9.4.1.2.2 Information Included in Source List.....	231
9.4.1.2.3 Example of Source Listing.....	233
9.4.1.2.4 Notes on Information at Compilation (Source List).....	235
9.5 Language Specifications.....	236
9.5.1 Supported Language Standard.....	236
9.5.2 Half-Precision (16 bit) Floating-Point Type.....	236
9.6 Notes on Linking with Different Languages and Trad/Clang Modes.....	237
9.6.1 Compile commands and required options when linking.....	237
9.6.2 Linking with C++.....	237
9.6.3 Linking with Fortran.....	238
9.7 SIMD Built-in Functions.....	241
9.8 Compatibility with GNU C Specifications.....	241
9.8.1 GNU C Extensions.....	241
9.8.2 GNU C Compatible Options.....	241
9.9 Code Coverage.....	243
9.9.1 How to Use Code Coverage.....	244
9.9.2 Necessary Files to Use Code Coverage.....	244
9.9.2.1 The ".gcno" File.....	244
9.9.2.2 The ".gcda" File.....	245
9.9.3 Notes on Code Coverage.....	246
9.10 Restrictions and Notes.....	246
9.10.1 Restrictions of Macros CMPLX, CMPLXF, and CMPLXL in complex.h.....	246
Appendix A Implementation-dependent Specifications.....	248
A.1 Compiling.....	248
A.2 Environment.....	248
A.3 Identifiers.....	248
A.4 Characters.....	248
A.5 Integers.....	249
A.6 Floating-Point Numbers.....	250
A.7 Arrays and Pointers.....	251
A.8 Registers.....	252
A.9 Structures, Unions, Enumerated Types, and Bit Fields.....	252
A.10 Qualifiers.....	253
A.11 Declarators.....	253
A.12 Statements.....	253
A.13 Preprocessor Directives.....	253
A.14 Standard Library Functions.....	254
A.15 OpenMP Specifications.....	254

Appendix B Compilation Limitations.....	255
Appendix C Restrictions and Notes.....	257
C.1 Restriction.....	257
C.2 Notes.....	257
C.2.1 Variable Arguments of Function.....	257
C.2.2 Expression Result of Signed Integer Type.....	257
C.2.3 Undefined Behavior.....	257
C.2.4 Arrays of Variable Length.....	257
C.2.5 Restriction of Register Definition for Floating Point Types with asm Keyword.....	257
C.2.6 Restriction of Floating Point Types Expression as a Register Constraint for asm Operands.....	257
C.2.7 Debug Using Debugger.....	257
C.2.8 Changing SVE Vector Register Size.....	258
C.2.9 Integer Division Exception when the Divisor Is Zero.....	258
C.2.10 Link Error (undefined reference to).....	258
C.2.11 Restriction of registers with asm keyword and on an inline asm clobber lists.....	259
Appendix D Compatibility with GNU C Specifications.....	260
D.1 GNU C Extensions.....	260
D.1.1 Attributes.....	261
D.1.2 Built-in Functions.....	262
D.2 GNU C Compatible Options.....	265
Appendix E Data and Memory Regions.....	270
E.1 Data Size and Alignment.....	270
E.2 Memory Regions.....	270
E.3 Data Allocation.....	270
E.4 Data Allocation to Stack Region.....	271
Appendix F Code Coverage.....	274
F.1 How to Use Code Coverage.....	274
F.2 Necessary Files to Use Code Coverage.....	275
F.2.1 The ".gcno" File.....	275
F.2.2 The ".gcda" File.....	276
F.3 Notes on Code Coverage.....	277
Appendix G Fujitsu Extended Functions.....	278
G.1 How to Use Fujitsu Extended Functions.....	278
G.1.1 Header.....	278
G.1.2 Supported Functions.....	278
Appendix H Runtime Information Output Function.....	279
H.1 Usage of Runtime Information Output Function.....	279
H.1.1 Range of Obtainable Information.....	279
H.1.2 Runtime Environment Variables.....	280
H.2 Output of Runtime Information Output Function.....	280
H.2.1 Output Information.....	280
H.2.2 Output Format.....	281
H.2.3 Output Example.....	281
H.3 Notes on Runtime Information Output Function.....	283
Appendix I Using High-Speed Facility on Job Operation Software.....	285
I.1 Inter-Core Hardware Barrier.....	285
I.1.1 Compilation.....	285
I.1.2 Execution.....	285
I.2 Sector Cache.....	287
Appendix J Fujitsu OpenMP Library.....	288
J.1 Overview of Multiprocessing.....	288

J.2 Automatic Parallelization.....	288
J.2.1 Compilation (Automatic Parallelization).....	288
J.2.2 Execution Process (Automatic Parallelization).....	289
J.2.3 Example of Compilation and Execution.....	292
J.2.4 Performance Tuning.....	292
J.2.5 Automatic Parallelization.....	292
J.2.6 Optimization Control Line.....	293
J.2.7 Notes on Automatic Parallelization.....	293
J.2.8 Linking with Other Multi-Thread Programs.....	293
J.3 Parallelization by OpenMP Specification.....	294
J.3.1 Compilation (OpenMP Parallelization).....	294
J.3.2 Execution Process (OpenMP Parallelization).....	295
J.3.3 Implementation-Dependent Specifications.....	300
J.3.4 Notes on OpenMP Programming.....	303
J.3.5 Linking with Other Multi-Thread Programs.....	303
J.3.6 Debugging OpenMP Programs.....	304
J.4 Using High-Speed Facility on Job Operation Software.....	305
J.4.1 Management of CPU Resources.....	305
J.4.2 CPU Binding.....	306
J.4.3 Inter-Core Hardware Barrier.....	306
J.4.4 Sector Cache.....	308

Chapter 1 Overview

This chapter describes an overview and an outline of how to process C programs using this system.

1.1 Configuration of the C Language Processing System

This system is one of language processing programs. This system consists of the following components:

Compile Command

The compile command is a program which calls the C compiler, the preprocessor, the assembler, and the linker. The compile command converts a program written in the C programming language into an executable file format through the compilation and linking.

There are two types of compile commands in this system,

Table 1.1 Compile Command

Kind	Command name	Description
Cross compiler	<code>fccpx</code>	The command used on the login node
Native compiler	<code>fcc</code>	The command used on the compute node

In this manual, the `fccpx` is used as the name of the compile command. Read the `fccpx` as the `fcc` when you use the native compiler.

C Compiler

Compile command calls C compiler. C compiler creates object programs from C source programs.

C compiler has two modes with different user interfaces. The mode to be used is specified by the option of the compile command.

Table 1.2 Two Modes of C Compiler

Mode	Description
Trad Mode	<p>This mode uses an enhanced compiler based on compilers for K computer and PRIMEHPC FX100 or earlier system.</p> <p>This mode is suitable for maintaining compatibility with the past Fujitsu compiler. Supported specifications are C89/C99/C11 and OpenMP 3.1/Part of OpenMP 4.0/Part of OpenMP 4.5.</p> <p>For specifications of Trad Mode, see "Chapter 2 From Compilation to Execution" to "Chapter 8 Debugging Functions" and "Appendix".</p>
Clang Mode	<p>This mode uses an enhanced compiler based on Clang/LLVM.</p> <p>This mode is suitable for compiling programs using the latest language specification and open source software. Supported specifications are C89/C99/C11 and OpenMP 4.5/Part of OpenMP 5.0.</p> <p>For specifications of Clang Mode, see "Chapter 9 Clang Mode".</p>



Note

Trad Mode and Clang Mode may have different implementation-dependent specifications since these modes are based on the different compilers.

C Library for Multiprocessing

This system provides two libraries for multiprocessing.

Table 1.3 Two Libraries for Multiprocessing

Library Name	Description
LLVM OpenMP Library	This is the library for multiprocessing based on LLVM OpenMP Runtime Library which is an open source software. Supported specifications are OpenMP 4.5/Part of OpenMP 5.0. This library is available in Trad Mode and Clang Mode. For specifications of LLVM OpenMP Library, see " Chapter 4 Multiprocessing ".
Fujitsu OpenMP Library	This is the library for multiprocessing based on Fujitsu OpenMP Library for K computer and PRIMEHPC FX100 or earlier system. This library is suitable for maintaining compatibility with the past Fujitsu OpenMP Library. Supported specifications are OpenMP 3.1/Part of OpenMP 4.0/Part of OpenMP 4.5. This library is only available in Trad Mode. For specifications of Fujitsu OpenMP Library, see " Appendix J Fujitsu OpenMP Library ".

Online Manual

Online manual pages, accessed by the man command, provide information about fccpx(1), fcc(1), fccpx_trad_mode(7), fcc_trad_mode(7), fccpx_clang_mode(7), and fcc_clang_mode(7).

1.2 How to Use

This section gives a simple description of how to use this system.

1.2.1 Preparation

Append the following values to user environment variable before use. "*installation_path*" is the product/software installation path. For "*installation_path*", contact the system administrator.

Environment Variable	Value
PATH	<i>/installation_path/bin</i>
LD_LIBRARY_PATH	<i>/installation_path/lib64</i>
MANPATH	<i>/installation_path/man</i>

Example:

```
$ PATH=/installation_path/bin:$PATH
$ export PATH
$ LD_LIBRARY_PATH=/installation_path/lib64:$LD_LIBRARY_PATH
$ export LD_LIBRARY_PATH
$ MANPATH=/installation_path/man:$MANPATH
$ export MANPATH
```



Note

When setting the environment variable LANG with a locale, ensure that the locale is installed on the node where the program is compiled.

A list of available locales can be found with the "locale -a" command.

Example:

```
$ locale -a
C
:
```

If the locale is not installed, a warning message may be output or a compilation error may occur. In this case, take one of the following actions:

- Specify C for the environment variable LANG
- Ask the system administrator to install the locale

The current locale can be found with the "locale" command.

Example:

```
$ locale
LANG=C
:
```

1.2.2 Compilation and Linking

Use the compile command to compile source code written in the C language.

A wealth of functions for optimizing the object code and other uses can be used at compilation time. These functions can be used by specifying the compiler option in the operands of the compile command.

The compile command supports linking by calling linker (ld command).

Use the compile command to generate the executable program by linking the C language library with the object programs which are generated during compilation.

In this system, the C language specifications used at compilation time are based on the GNU C compiler specification (GNU C Extensions). Part of the GNU extensions to the C language and part of the GNU C compiler options can be used.

For GNU extensions to the C language, refer to the Section "[D.1 GNU C Extensions](#)" in Trad Mode, "[9.8.1 GNU C Extensions](#)" in Clang Mode.

For GNU C compiler options, refer to the Section "[D.2 GNU C Compatible Options](#)" in Trad Mode, the Section "[9.8.2 GNU C Compatible Options](#)" in Clang Mode.

Here is an example of how to compile and link the program `sample.c` with the cross compiler.

The sample program: `sample.c`

```
#include <stdio.h>
int main(void)
{
    (void)printf("Hello, world.\n");
    return 0;
}
```

Compilation and linking:

```
$ fccpx sample.c
```

1.2.3 Debugging

For C programs debugging, use the source-level debugger. Use `gdb` to execute source-level debugger.

To use the debugger effectively, compile the C source code using the `-g` option for the compile command.

1.2.4 Tuning

To tune C programs, use a sampling function provided by the Profiler.

Refer to the "Profiler User's Guide" for the Profiler.

Chapter 2 From Compilation to Execution

This chapter describes the procedure for compiling C source code.

Note that this chapter is for Trad Mode. See "[Chapter 9 Clang Mode](#)" about Clang Mode.

2.1 Compile Command

Use the compile command to compile source code and create an executable program.

The compile command analyzes various operands specified in the compile command line and calls the preprocessor, compiler, assembler (as command), and linker (ld command) as necessary.

2.1.1 Syntax of the Compile Command

You can specify options for the preprocessor, compiler, assembler (as command), and linker (ld command). You can also specify filenames.

Options for the preprocessor and compiler are called compiler options.

Options for the compiler (compile command), assembler (as command), and linker (ld command) can be viewed with the man command.

Table 2.1 Format of the Compile Command

Command Name		Operands
Cross compiler	<code>fccpx</code>	<code>[_option-list] _filename-list</code>
Native compiler	<code>fcc</code>	<code>[_option-list] _filename-list</code>

_: At least one blank is required.

Information

There is no restriction on the order in which *option-list* and *filename-list* are specified.

option-list and *filename-list* can be mixed.

2.1.2 Input Files for the Compile Command

The file types that can be specified as input files for the compile command are as follows:

Table 2.2 Format of Input Files

File Type	File Suffix	Passed To
Headers	<code>.h</code>	Preprocessor ^[a]
	<code>.H</code>	
C source file	<code>.c</code>	Preprocessor and compiler
	<code>.i</code>	
Assembler source file	<code>.s</code>	Assembler
Assembler source file that must be preprocessed	<code>.S</code>	Preprocessor and assembler
Object file	<code>.o</code>	Linker

[a] When -E option is effective.

2.1.3 Return Value of the Compile Command

The possible return values of the compile command are as follows:

Table 2.3 Return Values of the Compile Command

Return Value	Meaning
0	Terminated normally.
Non-zero	An error occurred at compiling or linking.

2.2 Compiler Options

This section describes the format of compiler options and their meanings.

Compiler options may be specified as follows:

- Operand for the compile command
- Environment variable (Refer to "2.3 Environment Variable for Compile Command")
- Compilation Profile File (Refer to "2.4 Compilation Profile File")

Refer to "2.4 Compilation Profile File" for the priority of compiler options.

Usually, options that are unrecognizable to the compile command are ignored and warning message is output. The behavior for unrecognized options can be changed by specifying the environment variable FCOMP_UNRECOGNIZED_OPTION. Refer to "2.3 Environment Variable for Compile Command" for the environment variable FCOMP_UNRECOGNIZED_OPTION.

2.2.1 Syntax of Compiler Options

Compiler options can be specified as operands of the compile command. The format is given below.

- General compiler options

```
[ -# ] [ -### ] [ -A- ] [ -Aname[tokens] ] [ -B{dynamic|static} ] [ -C ] [ -Dname[=tokens] ] [ -E ] [ -H ] [ -Idir ] [ -Ldir ] [ -M ] [ -MD ] [ -MF filename ] [ -MM ] [ -MMD ] [ -MP ] [ -MT target ] [ -Nopt ] [ -P ] [ -Qc ] [ -S ] [ -SSL2 ] [ -SSL2BLAMP ] [ -Uname ] [ -Wtool,arg1[,arg2]\... ] [ -Yitem,dir ] [ -c ] [ {-g|-g0} ] [ -lname ] [ -mt ] [ -o pathname ] [ -shared ]
```

- Message options

```
[ -V ] [ {-help|--help} ] [ -j ] [ -w ]
```

- Optimization options

```
[ -Kopt ] [ -O[n] ] [ -x- ] [ -xfunc1[,func2]\... ] [ -xn ]
```

- Language specifications options

```
[ -ansi ] [ -std=level ] [ --linkcoarray ] [ --linkfortran ]
```

2.2.2 Description of Compiler Options

A description of the compiler options is given below.

2.2.2.1 General Options for Compiler

The following options are general options for the compiler.

-#

Shows commands passed by the compile command, but does not execute.

-###

Shows commands passed by the compile command.

-A-

Invalidates all predefined macros (except those starting with "_") and predefined assertions.

-Aname[(tokens)]

Uses *name* as a predicate for the specified *tokens* in the same way as for an `#assert` preprocessing directive.

The predefined assertions are given below:

- `system(unix)`

-B{dynamic|static}

Specify the kind of the library at linking. `-Bdynamic` is set by default.

These options are passed to the linker.

By using these options together with the `-lname` option, you can specify the kind of each library at linking.

-Bdynamic

Specifies to link with dynamic library or static library.

When the `-lname` option is specified after the `-Bdynamic` option, the linker will first search for a file of the name `libname.so`, and then search for a file of the name `libname.a`.

-Bstatic

Specifies to link with static library.

When the `-lname` option is specified after the `-Bstatic` option, the linker will only search for `libname.a`.



Example

When linking `libxxx.a` and `libyyy.so`

```
$ gccpx a.c -Bstatic -lxxx -Bdynamic -lyyy
```



Note

If the `-Bstatic` option is specified, the linker will search only a static library for not only the static library specified with the `-lname` option, but also standard library (`-lc` etc.) and necessary libraries by the feature. And, the linker error may occur because the necessary library is not found.

Specify the `-Bdynamic` option at the end of the command line when specifying the `-Bstatic` option.

Example 1: When the `-Bdynamic` option is not specified at the end

```
$ gccpx a.c -Bstatic -lxxx
ld: cannot find -lc
```

Example 2: When the `-Bdynamic` option is specified at the end

```
$ gccpx a.c -Bstatic -lxxx -Bdynamic
```

-C

This option executes the preprocessing phase to retain all comments other than those on preprocessing directive lines.

-Dname[=tokens]

Assigns *name* to the specified *tokens* as in the `#define` preprocessor directive.

If the same *name* is specified simultaneously in the `-D` and `-U` options, the *name* is defined following the order of the options.

The predefined macros are shown in "[6.1.6 Predefined Macro Names](#)".

-E

Performs only preprocessing on the specified C source code and sends the results to the standard output.

The output includes the preprocessor directives used in the next step of the processing system.

-H

Writes the pathnames of the valid headers in the current compilation to the standard error.

-I*dir*

Sets the search path for headers with names that do not start with '/' (i.e. headers specified with an absolute path name). The directory specified in *dir* is searched first and then the directory-installed standard headers are searched.

If multiple directories are specified in multiple -I options, the directories are searched in the specified order.

The directory that retrieves headers is performed in the following order:

- The files enclosed in double quotation marks (") are searched in the following order:
 1. The current directory that contains the #include preprocessor directive
 2. Directories specified with the -I option
 3. Directories specified as the environment variable CPATH
 4. Directories specified with the -isystem option
 5. Directories specified as the environment variable C_INCLUDE_PATH
 6. The Install directory that contains headers provided by the compiler
 7. The directory-installed standard headers
 8. Directories specified with the -idirafter option
- The files enclosed in angle brackets (<>) are searched in the following order:
 1. Directories specified with the -I option
 2. Directories specified as the environment variable CPATH
 3. Directories specified with the -isystem option
 4. Directories specified as the environment variable C_INCLUDE_PATH
 5. The Install directory that contains headers provided by the compiler
 6. The directory-installed standard headers
 7. Directories specified with the -idirafter option

If a header is specified with an absolute path name, only the specified absolute path name is searched.

If the specified directory does not exist, this option is invalidated.

-L*dir*

Adds *dir* to the list of directories through which the linker searches for libraries.

This option is passed to the linker.

Libraries are searched in the following order:

1. Directories specified as the argument of the -L option
2. The directory installed libraries provided by the compiler
3. The directory installed standard libraries
4. Directories specified to the environment variable LIBRARY_PATH

-M

Specifies to output the dependency of the sources in the format recognized by the make command.

When this option is valid, only the preprocessing is executed.

-MD

Specifies to output the dependency of the sources in the format recognized by the `make` command.

By default, the result is output to the file with a ".d" suffix.

When the `-E` and `-o` options are specified, the result is output to the file specified by the `-o` option.

When the `-MF` option is specified simultaneously, the result is output to the file specified by the `-MF` option.

-MF *filename*

Specifies the *filename* to output the result of the dependency of the source using the `-M`, `-MM`, `-MD` or `-MMD` options.

-MM

Specifies to output the dependency of the sources in the format recognized by the `make` command. However, a standard header is not contained in the output.

When this option is valid, only the preprocessing is executed.

-MMD

Specifies to output the dependency of the sources in the format to recognized by the `make` command. However, a standard header is not contained in the output.

By default, the result is output to the file with a ".d" suffix.

When the `-E` and `-o` options are specified, the result is output to the file specified by the `-o` option.

When the `-MF` option is specified simultaneously, the result is output to the file specified by the `-MF` option.

-MP

Specifies to append the phony target of the dependency to the output of the result produced by the `-M`, `-MM`, `-MD` or `-MMD` option.

-MT *target*

Specifies to change the *target* of the dependency in the output of the result produced by the `-M`, `-MM`, `-MD` or `-MMD` option.

-Nopt

For *opt*, specify one of the following:

```
{ {Rtrap|Rnotrap} | {cancel_overtime_compilation|nocancel_overtime_compilation} | check_cache_arraysize | {clang|
noclang} | {coverage|nocoverage} | {exceptions|noexceptions} | {fjcex|nofjcex} | {fjprof|nofjprof} | {hook_func|
nohook_func} | {hook_time|nohook_time} | {libomp|fjomplib} | {line|noline} | lst[={p|t}] | lst_out=file |
profile_dir=dir_name | quickdbg[={subchk|nosubchk|heapchk|noheapchk|inf_detail|inf_simple}] |
{reordered_variable_stack|noreordered_variable_stack} | {rt_tune|rt_notune} | rt_tune_func | rt_tune_loop[={all|
innermost}] | {setvalue[=setarg]|nosetvalue} | src | sta }
```

Multiple *opts* can be specified to "-N" option using commas to separate them. For example,

```
-Nsrc,sta
```

can be specified instead of specifying

```
-Nsrc -Nsta
```

For details, see Section "[2.2.2.6 -N Option](#)".

-P

This option specifies not to include linemarkers in the output of the `-E` option.

-Qc

In *c*, specify *y* or *n*. If `-Qy` is specified, information identifying the various compile tools called is written to the output file. If `-Qn` is specified, this information is not written. By default, `-Qy` is set.

-S

Compiles the specified C source file and leaves the assembly-language output in a corresponding file with the suffix ".s".

This option is invalidated when the -E option is specified simultaneously.

-SSL2

Specifies to link C-SSL II, C-SSL II thread-parallel capabilities, and BLAS/LAPACK.

-SSL2BLAMP

Specifies to link C-SSL II, C-SSL II thread-parallel capabilities, and BLAS/LAPACK thread-parallel versions.

This option just replaces the sequential BLAS/LAPACK from -SSL2 with the corresponding thread-parallel versions.

-Uname

Invalidates the definition of *name* as specified in #undef preprocessor directive.

If the same *name* is specified simultaneously in the -D and -U options, the *name* is invalidated following the order of the options.

-Wtool,arg1[,arg2]...

Directs that the specified arguments *arg1*[,*arg2*]... are to be passed as separate arguments to *tool*. An individual argument must be separated by a comma only.

tool can be one of the following characters:

p	Preprocessor
0	Compiler
a	Assembler
l	Linker

-Yitem,dir

Specifies the new directory *dir* as the position for *item*.

For *item*, specify one of the following characters:

0	Changes the path name of the compiler to <i>dir</i> /ccpcomp _x (cross compiler). Changes the path name of the compiler to <i>dir</i> /ccpcom (native compiler).
a	Changes the path name of the assembler to <i>dir</i> /as.
l (the lowercase letter L)	Changes the path name of the linker to <i>dir</i> /ld.
M	Changes the directory of the message files to <i>dir</i> .

If more than one -Y options are specified for an *item*, the last specified option is valid.

-c

Suppresses the linking phase. The object files generated up to then are not deleted.

This option is invalidated when the -E or -S option is specified simultaneously.

{-g|-g0}

The -g option generates additional information used by the debugger.

The -g0 option invalidates the -g option.

-g0 is set by default.

Refer to "[C.2.7 Debug Using Debugger](#)" about the notes on the debugger in Trad Mode.

-lname

Specifies to link a library named lib*name*.so or lib*name*.a.

The position of this option in the command line is important, since libraries are searched for in the order in which the other libraries and object files appear in the command line. This option must be specified after source files.

This option is passed to the linker.

-mt

Creates a thread safe object.

The compiler option **-mt** is needed if an object program compiled with it exists in the command line as an input file.

This option is also needed if the program is multi-threaded.

-o *pathname*

Creates a file of the name specified in *pathname*.

- If the **-c** option is set, an object file is created with the name *pathname*.
- If the **-shared** option is set, a shared object is created with the name *pathname*.
- If the **-S** option is set, an assembler source file is created with the name *pathname*.
- In other cases, an executable program is created with the name *pathname*, instead of the default `a.out`.

-shared

Specifies to create a shared object. This option is passed to the linker.

2.2.2.2 Options for Messages

The options for messages are given below.

-V

Specifies to emit the version and release information of each command of the C compiler to the standard error.

{-help|--help}

Specifies to output help information.

-j

Suppresses warnings on local automatic variables that are used before their values are set.

-w

Suppresses the output of warning messages.

2.2.2.3 Options for Optimization

The optimization options are described below.

-Kopt

For *opt*, specify one of the following:

{ {PIC|pic} | {SVE|NOSVE} | {alias_const|noalias_const} | {align_loops[=*N*]|noalign_loops} | *archi* | {array_declaration_opt|noarray_declaration_opt} | {array_private|noarray_private} | assume={shortloop|noshortloop|memory_bandwidth|nomemory_bandwidth|time_saving_compilation|notime_saving_compilation} | cmodel={small|large} | {const|noconst} | *cpu* | {dynamic_iteration|nodynamic_iteration} | {eval|noeval} | {eval_concurrent|eval_noconcurrent} | {extract_stride_store|noextract_stride_store} | fast | {fast_matmul|nofast_matmul} | {fconst|nofconst} | {fenv_access|nofenv_access} | {fp_contract|nofp_contract} | {fp_precision|nofp_precision} | {fp_relaxed|nofp_relaxed} | {fsimple|nofsimple} | {fz|nofz} | {hpctag|nohpctag} | {ilfunc[={loop|procedure}]]|noilfunc} | instance=*N* | {largepage|nolargepage} | {lib|nolib} | {loop_blocking[=*N*]|loop_noblocking} | {loop_fission|loop_nofission} | {loop_fission_stripmining[={*N*|L1|L2}]]|loop_nofission_stripmining} | loop_fission_threshold=*N* | {loop_fusion|loop_nofusion} | {loop_interchange|loop_nointerchange} | {loop_part_parallel|loop_nopart_parallel} | {loop_part_simd|loop_nopart_simd} | {loop_perfect_nest|loop_noperfect_nest} | {loop_versioning|loop_noversioning} | lootype={f|n|s} | memalias|nomemalias | {mfunc[={1|2|3}]]|nomfunc} | noprefetch | {ocl|noocl} | {omitfp|noomitfp} | {openmp|noopenmp} | {openmp_assume_norecurrence|openmp_noassume_norecurrence} | {openmp_collapse_except_innermost|openmp_nocollapse_except_innermost} | openmp_loop_variable={private|standard} |

```
{openmp_ordered_reduction|openmp_noordered_reduction} | {openmp_simd|noopenmp_simd} | {optlib_string|
nooptlib_string} | {optmsg[={1|2}]|nooptmsg} | {parallel|nparallel} | {parallel_fp_precision|parallel_nofp_precision} |
parallel_iteration=N|parallel_strong | {pc_relative_literal_loads|nopc_relative_literal_loads} | {plt|noplt} | {preex|nopreex} |
prefetch_cache_level={1|2|all} | {prefetch_conditional|prefetch_noconditional} | {prefetch_indirect|prefetch_noindirect} |
{prefetch_infer|prefetch_noinfer} | prefetch_iteration=N| prefetch_iteration_L2=N| prefetch_line=N| prefetch_line_L2=N|
{prefetch_sequential[={auto|soft}]|prefetch_nosequential} | {prefetch_stride[={soft|hard_auto|hard_always}]|
prefetch_nostride} | {prefetch_strong|prefetch_nostrong} | {prefetch_strong_L2|prefetch_nostrong_L2} | {preload|nopreload}
| {rdconv[={1|2}]|nordconv} | {reduction|noreduction} | {region_extension|noregion_extension} | {restp[={all|arg|restrict}]|
norestp} | {sch_post_ra|nosch_post_ra} | {sch_pre_ra|nosch_pre_ra} | {sibling_calls|nosibling_calls} | {simd[={1|2|auto}]|
nosimd} | {simd_packed_promotion|simd_nopacked_promotion} | {simd_reduction_product|simd_noreduction_product} |
simd_reg_size={128|256|512|agnostic} | {simd_uncounted_loop|simd_nouncounted_loop} | {simd_use_multiple_structures|
simd_nouse_multiple_structures} | {strict_aliasing|nostrict_aliasing} | {striping[=N]|nostriping} | {swp|noswp} |
{swp_freq_rate=N|swp_ireg_rate=N|swp_preg_rate=N} | swp_policy={auto|small|large} | swp_strong | swp_weak |
tls_size={12|24|32|48} | {unroll[=N]|nounroll} | {unroll_and_jam[=N]|nounroll_and_jam} | visimpact | {zfill[=N]|nozfill} }
```

Multiple *opts* can be specified to "-K" option using commas to separate them. For example,

```
-Kfast,parallel
```

can be specified instead of specifying

```
-Kfast -Kparallel
```

For more details, see Section ["2.2.2.5 -K Option"](#).

-O[*n*]

In *n*, specify the level of optimization as 0, 1, 2 or 3. If the -O option is specified without *n*, -O2 is set. As the optimization level increases, the execution time decreases and the compilation time increases. The higher levels of optimization functionally include the lower levels of optimization.

If the -O[*n*] option is not specified, -O0 is set.

This option is effective to C source files only. For information about C source files, see Section ["2.1.2 Input Files for the Compile Command"](#).

- Optimization level 0

No optimization is performed.

- Optimization level 1

Basic optimization is performed. Moreover, the -Knoalias_const option is set. This may reduce the object size and the execution time compared with optimization level 0.

- Optimization level 2

In addition to the optimization of level 1, the following optimizations are performed:

- Loop unrolling (Equivalent to specifying -Kunroll)
- Software pipelining (Equivalent to specifying -Kswp)
- Loop blocking (Equivalent to specifying -Kloop_blocking)
- Loop fusion (Equivalent to specifying -Kloop_fusion)
- Loop fission (Equivalent to specifying -Kloop_fission)
- Loop interchange (Equivalent to specifying -Kloop_interchange)
- Using prefetch instructions (Equivalent to specifying -Kprefetch_sequential,prefetch_cache_level=all)
- SIMD extend instruction conversion (Equivalent to specifying -Ksimd=auto)
- Adjusting loop alignment (Equivalent to specifying -Kalign_loops)
- Optimization of the sibling call (Equivalent to specifying -Ksibling_calls)

- Repeated application of optimization functions

Repeated application of optimization functions means that the optimization functions performed in optimization level 1 are repeatedly performed until there is no room for further optimization.

- Optimization is performed on the assumption that expressions of signed integer type whose size is 4-byte or less do not overflow when the expressions increase or decrease by a constant value as the loop repeats (Equivalent to specifying `-Krdconv=1`)

This may result in an increased object program size.

This level is specified to pursue the execution performance improvement of the object program.

- Optimization level 3

In addition to the optimization of level 2, the following optimizations are performed:

- Unrolling of nested loops
- Transforming loops into perfectly nested loops to promote optimization such as loop interchange (Equivalent to specifying `-Kloop_perfect_nest`)
- Loop unswitching
- Clone optimization
- Inline expansion (Equivalent to specifying `-x-`)

This level is specified to pursue the execution performance improvement of the object program.

-x-

Inline expansion instead of function calls is performed for all defined functions in which the number of initialized declarations and executable statements after macro expansion is not greater than that allowed by the compiler (n).

This option is effective only if the `-O1` option or higher is set.

If both this option and the `-xn` option are specified, the one specified last is effective.

This option determines whether to perform inline expansion by using the value of n . Therefore, the inline function does not become the target of the inline expansion when the compiler judges that inline expansion should not be performed.

See Section "[3.2.9 Inline Expansion](#)" for about inline expansion.

-xfunc1[,func2]...

Inline expansion of function calls is performed for the functions *func1*[,*func2*]... defined in the C source code.

This option is effective only if the `-O1` option or higher is set.

-xn

Inline expansion is performed for functions in which the number of initialized declarations and executed statements is equal to or less than the specified number n .

n can be a number from 0 to 2147483647.

If both this option and the `-x-` option are specified, the one specified last is effective.

When the `-x0` option is specified, the `-x-`, `-xn`, and `-xfunc1`[,*func2*]... options are ignored, so inline expansion is invalidated. However, the one specified last is effective.

This option is effective only if the `-O1` option or higher is set.

The inline function becomes the target of the inline expansion regardless of the value of n .

2.2.2.4 Options for Language Specifications

This section explains the options relating to language specifications.

-ansi

This option is equivalent to the `-std=c89` option.

See the `-std=level` option for details.

-std=*level*

This option specifies the level of language specifications to be interpreted by the compiler (including the preprocessor). For *level*, specify c89, c99, c11, gnu89, gnu99, or gnu11. -std=gnu11 is set by default.

-std=c89

-std=gnu89

In addition to the C89 specifications, GNU C Extensions is used at compilation time.

-std=c99

-std=gnu99

In addition to the C99 specifications, GNU C Extensions is used at compilation time.

For more detail about the C99 specifications, refer to "[6.2.1 C99 Specifications](#)".

-std=c11

-std=gnu11

In addition to the C11 specifications, GNU C Extensions is used at compilation time.

For more detail about the C11 specifications, refer to "[6.2.2 C11 Specifications](#)".

Some of the predefined macros are changed by the -std=*level* option. For predefined macro names, see "[6.1.6 Predefined Macro Names](#)".

--linkcoarray

Directs to search for a library required for linkage with Fortran that does use the COARRAY specification.

If this option is not specified, search processing is not performed, enabling a reduction in compilation time.

--linkfortran

Directs to search for a library required for linkage with Fortran that does not use the COARRAY specification.

If this option is not specified, search processing is not performed, enabling a reduction in compilation time.

2.2.2.5 -K Option

-K{PIC|pic}

Specifies to generate position-independent code (PIC). The -KPIC and -Kpic options are equivalent.

Note that the -K{PIC|pic} and -Kmodel=large options cannot be set simultaneously.

This option takes effect only during compilation.

-K{SVE|NOSVE}

-KSVE specifies to output object files using SVE, which is an Armv8-A extension. -KSVE is set by default.

Specify the -KNOSVE option to generate object files for a processor that does not support SVE.

-K{alias_const|noalias_const}

-Knoalias_const specifies data optimization pointed to by const pointers. This assumes that the const pointers are not defined by other pointers. The -Kalias_const option invalidates the -Knoalias_const option. -Knoalias_const is set by default.

The -Kalias_const and -Knoalias_const options require that the -O1 option or higher is set.

The -Knoalias_const option is invalidated if the -Knoconst option is set.

-K{align_loops[=*N*]|noalign_loops}

-Kalign_loops specifies to align loops to a power of two byte boundary. The -Knoalign_loops option invalidates the -Kalign_loops option. *N* is the number of bytes for the alignment boundary. The value that can be specified for *N* is a power of two (from 1 to 32768) or 0. If a value is not specified or 0 is specified for *N*, the compiler will automatically determine a value. The -Knoalign_loops is set by default when the -O1 option is set. The -Kalign_loops option is set by default when the -O2 option or higher is set.

Specifying -Knoalign_loops is equivalent to specifying -Kalign_loops=1.

The -Kalign_loops and -Knoalign_loops options require that the -O1 option or higher is set.

-Karchi

Specifies the architecture. In *archi*, specify one of ARMV8_A, ARMV8_1_A, ARMV8_2_A, or ARMV8_3_A. -KARMV8_3_A is set by default.

-KARMV8_A

This option specifies to generate object files using instructions in Armv8-A.

-KARMV8_1_A

This option specifies to generate object files using instructions in Armv8-A and Armv8.1-A.

-KARMV8_2_A

This option specifies to generate object files using instructions in Armv8-A, Armv8.1-A, and Armv8.2-A.

-KARMV8_3_A

This option specifies to generate object files using instructions in Armv8-A, Armv8.1-A, Armv8.2-A, and Armv8.3-A.

-K{array_declaration_opt|noarray_declaration_opt}

-Karray_declaration_opt specifies to perform the optimization such as SIMDization assuming that the array subscript does not exceed the range of the array declaration. The -Knoarray_declaration_opt option invalidates the -Karray_declaration_opt option.

The -Knoarray_declaration_opt is set by default.

The result of the execution cannot be guaranteed if the -Karray_declaration_opt option is set when the above assumption is not based.

The -Karray_declaration_opt and -Knoarray_declaration_opt options require that the -O1 option or higher is set.

-K{array_private|noarray_private}

-Karray_private specifies to localize arrays in a loop. The -Knoarray_private option invalidates the -Karray_private option. -Knoarray_private is set by default.

The -Karray_private and -Knoarray_private options require that the -Kparallel option is set.

-Kassume={shortloop|noshortloop|memory_bandwidth|nomemory_bandwidth|time_saving_compilation|notime_saving_compilation}

These options specify whether or not to control optimization considering features of the program. Multiple -Kassume options can be specified at the same time.

-Kassume=noshortloop,assume=nomemory_bandwidth,assume=notime_saving_compilation is set by default. The compiler performs optimizations under the following policy.

- Assume that the iteration count is large when the iteration count of the innermost loop is unknown at compilation.
- Solve the bottleneck of CPU operation preferentially.
- Give priority to increase the speed of the executable program rather than decrease of the compilation time.

These options require that the -O1 option or higher is set.

-Kassume={shortloop|noshortloop}

-Kassume=shortloop specifies to assume that iteration counts are small when the iteration count of the innermost loop in the program is unknown at compilation. -Kassume=noshortloop specifies to assume that iteration counts are not small. -Kassume=noshortloop is set by default.

When -Kassume=shortloop option is specified, optimizations such as automatic parallelization, loop unrolling, and software pipelining may be controlled or invalidated.

-Kassume={memory_bandwidth|nomemory_bandwidth}

-Kassume=memory_bandwidth specifies to assume that innermost loops in the program has a memory bandwidth bottleneck. -Kassume=nomemory_bandwidth specifies to assume that the innermost loop in the program does not have a memory bandwidth bottleneck. -Kassume=nomemory_bandwidth is set by default.

When -Kassume=memory_bandwidth option is specified, optimizations such as zfill optimization promotion and software pipelining may be controlled or invalidated.

-Kassume={time_saving_compilation|notime_saving_compilation}

-Kassume=time_saving_compilation specifies to control optimization for decreasing compilation time of the program. **-Kassume=notime_saving_compilation** specifies to optimize with giving priority to the speed of the executable program rather than decrease of the compilation time. **-Kassume=notime_saving_compilation** is set by default.

-Kcmodel={small|large}

This option specifies the possible largest size of the text area and the static data area in an executable program or a shared object. **-Kcmodel=small** is set by default.

Note that the **-K{PIC|pic}** and **-Kcmodel=large** options cannot be set simultaneously.

-Kcmodel=small

The total size of the text area and the static data area is limited to 4GB at linking. This option creates an efficient object program.

-Kcmodel=large

Only the size of the text area is limited to 4GB at linking. This option is used when the static data area is large and an error occurs at linking.

-K{const|noconst}

-Kconst specifies to treat `const` qualified data as constant. The **-Knoconst** option invalidates the **-Kconst** option. **-Kconst** is set by default.

-Kcpu

Specifies the target processor. In *cpu*, specify A64FX or GENERIC_CPU. **-KA64FX** is set by default.

-KA64FX

-KA64FX specifies to output object files for the A64FX processor. The **-Khpctag** option is effective when the **-KA64FX** option is set.

-KGENERIC_CPU

-KGENERIC_CPU specifies to output the object file for the Arm processor.

-K{dynamic_iteration|nodynamic_iteration}

In a multiloop composed of parallel executable loops, the **-Kdynamic_iteration** option specifies to select the loop dynamically (at execution) depending on the execution environment. The nest level of the loop that can be dynamically selected is up to three from the outside of the multiloop. The **-Knodynamic_iteration** option specifies to select the outermost loop of the multiloop to execute in parallel. **-Knodynamic_iteration** is set by default.

The **-Kdynamic_iteration** and **-Knodynamic_iteration** options require that the **-Kparallel** option is set.

-K{eval|noeval}

-Keval performs the optimization that changes the method of operator evaluation. The **-Knoeval** option invalidates the **-Keval** option. **-Knoeval** is set by default.

If the **-Keval** option is set, side effects (calculation errors or runtime exceptions) may occur in the execution results and produce unexpected results. See Section "[3.6.1 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

If the **-Keval** option is set, the **-Kfsimple** option also takes effect. Moreover, the **-Kreduction** option also takes effect if **-Kparallel** is set.

If **-Keval** and **-Ksimd[={ 1|2|auto}]** options are set, the **-Kfsimple** and **-Ksimd_reduction_product** options also takes effect.

If the **-Knoeval** option is set, the **-Knofsimple**, **-Knoreduction**, and **-Ksimd_noreduction_product** options are set.

The **-Keval** and **-Knoeval** options require that the **-O1** option or higher is set.

Before using the **-Keval** option, see Section "[3.3.1.2 Arithmetic Evaluation Method Modification](#)".

-K{eval_concurrent|eval_noconcurrent}

-Keval_concurrent option specifies to give priority to the instruction-level parallelism in the tree-height-reduction optimization. **-Keval_noconcurrent** option specifies to suppress the instruction-level parallelism and specifies to give priority to utilizing FMA instructions in the tree-height-reduction optimization. **-Keval_noconcurrent** option is set by default.

The **-Keval_concurrent** and **-Keval_noconcurrent** options require that the **-O1** option or higher is effective and the **-Keval** option is effective.

The effect of the `-Keval_concurrent` option is expected when the loop iteration count is small and therefore the software pipelining is not applied.

See section [3.3.9 Tree-Height-Reduction Optimization](#) for the tree-height-reduction optimization.

`-K{extract_stride_store|noextract_stride_store}`

`-Kextract_stride_store` option specifies to use scalar instructions for stride access store in the target loop when using SIMD extensions. The `-Knoextract_stride_store` option specifies to use SIMD instructions for stride access store in the target loop when using SIMD extensions. `-Kextract_stride_store` option is set by default.

When the `-Kextract_stride_store` option is specified, the function of "[3.2.7.4 SIMD with Redundant Executions for the SIMD Width](#)" is suppressed for loops to which this function is applied.

Note that the size of the object program and compile time may increase if the `-Kextract_stride_store` option is specified.

The `-Kextract_stride_store` and `-Knoextract_stride_store` options require that the `-O2` option or higher option is set.

`-Kfast`

Specifies the optimizations for high speed execution on the target machine.

This option is equivalent to the following:

`-O3 -Keval,fast_matmul,fp_contract,fp_relaxed,fz,ilfunc,mfunc,omitfp,simd_packed_promotion`

This option must be set at both compilation and linking.

When this option is set, side effects may occur in the execution results due to effect of the `-Keval`, `-Kfast_matmul`, `-Kfp_contract`, `-Kfp_relaxed`, `-Kilfunc`, and `-Kmfunc` options. See Section "[3.6.1 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

`-K{fast_matmul|nofast_matmul}`

The compiler option `-Kfast_matmul` performs the optimization that converts the loop of matrix multiplication into a high speed library call. The compiler option `-Knofast_matmul` invalidates the compiler option `-Kfast_matmul`. The compiler option `-Knofast_matmul` is set by default.

When the compiler option `-Kfast_matmul` is set, side effects (calculation errors) may occur in the execution results. See Section "[3.6.1 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

The compiler options `-Kfast_matmul` and `-Knofast_matmul` require that the compiler option `-O2` or higher option is set.

The compiler option `-Kfast_matmul` is needed if an object program compiled with it exists in the command line as an input file.

`-K{fconst|nofconst}`

`-Kfconst` specifies that the data type of a floating-point constant without a suffix is float. The `-Knofconst` option invalidates the `-Kfconst` option. `-Knofconst` is set by default.

`-K{fenv_access|nofenv_access}`

`-Kfenv_access` specifies that a program may access the floating-point environment to test flags or run under non-default modes. The `-Knofenv_access` option invalidates the `-Kfenv_access` option. `-Knofenv_access` is set by default.

The `-Kfenv_access` option disables some floating-point optimization.

`-K{fp_contract|nofp_contract}`

`-Kfp_contract` specifies to perform optimization using Floating-Point Multiply-Add/Subtract instructions. The `-Knofp_contract` option invalidates the `-Kfp_contract` option. `-Knofp_contract` is set by default.

When the `-Kfp_contract` option is set, side effects (calculation errors in rounding error extent) may occur in the execution results and produce unexpected results. See Section "[3.6.1 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

The `-Kfp_contract` and `-Knofp_contract` options require that the `-O1` option or higher is set.

`-K{fp_precision|nofp_precision}`

`-Kfp_precision` specifies to induce the combination of optimization options that do not cause calculation errors in floating-point operations. The `-Knofp_precision` option invalidates the `-Kfp_precision` option. `-Knofp_precision` is set by default.

The `-Kfp_precision` option is equivalent to replacing the `-Kfp_precision` option with the following:

-Knoeval,nofast_matmul,nofp_contract,nofp_relaxed,nofz,noifunc,nomfunc,parallel_fp_precision

If the -Kfp_precision option is effective, some optimizations are limited and the execution performance may decrease.

The -Knofp_precision option does not affect the individual options induced by -Kfp_precision option, even if they are specified together.

-K{fp_relaxed|nofp_relaxed}

-Kfp_relaxed specifies to convert floating point division operations and `sqrt` functions into the reciprocal approximation operation with the reciprocal approximation instructions and the Floating-Point Multiply-Add/Subtract instructions. This conversion is applied to the single-precision and double-precision real type calculation. The -Knofp_relaxed option invalidates the -Kfp_relaxed option. -Knofp_relaxed is set by default.

When the -Kfp_relaxed option is set, side effects may occur in the execution results. Also, this optimization may result in generating a floating-point exception whether the -NRtrap option is valid or not. See Section "[3.6.1 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

The -Kfp_relaxed and -Knofp_relaxed options require that the -O1 option or higher is set.

When the -NRtrap option and either of the -Knosimd option or the -KNOSVE option are effective, the optimization to convert a `sqrt` function into reciprocal approximation instructions is suppressed even though the -Kfp_relaxed option is set. Therefore, the execution performance may decrease as compared with when the -NRnotrap option is effective. Note that the reciprocal approximation instructions generated by the -Kfp_relaxed option may be evaluated in advance even if the -Knopreex option is enabled. And a floating-point exception may occur when the -Knopreex, -Kfp_relaxed, and -NRtrap options are valid.

-K{fsimple|nofsimple}

-Kfsimple specifies to simplify floating-point operation. For example, `x*0` operation is simplified to 0. The -Knofsimple option invalidates the -Kfsimple option. -Knofsimple is set by default.

When -Kfsimple is specified, side effects may occur in the execution results. See Section "[3.6.1 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

The -Kfsimple and -Knofsimple options require that the -O1 option or higher is set.

-K{fz|nofz}

-Kfz option specifies to use flush-to-zero mode. The -Knofz option specifies not to use flush-to-zero mode. The default is -Knofz.

When -Kfz option is specified, flush-to-zero mode replaces a denormalized number with zero with the same sign if a result or a source operand is a denormalized number. Moreover, side effects may occur in the execution results. See Section "[3.6.1 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization. Some programs may get more performance.

-Kfz and -Knofz options must be set at linking.

-Kfz is effective only when the -O1 option or higher is set.

-K{hpctag|nohpctag}

-Khpctag specifies to use the HPC tag address override function of the A64FX processor. The -Knopctag option invalidates the -Khpctag option. -Khpctag is set by default.

Using the HPC tag address override function, the Sector cache and the hardware prefetch assistance function become effective.

The -Khpctag and -Knopctag options require that the -KA64FX option is set.

The -Khpctag and -Knopctag options should be specified when programs are compiled and linked.

-K{ilfunc[={loop|procedure}]]noifunc}

-Kilfunc performs the inline expansion for the math functions. The -Knoifunc option invalidates the -Kilfunc option.

The following shows the functions for which inline expansion can be performed.

- Single-precision and double-precision real type function: `atan`, `atan2`, `cos`, `exp`, `log`, `log10`, `pow`, `sin`, and `tan`
- Single-precision and double-precision complex type function: `abs` and `exp`

The -Klib option is required for the -Kilfunc option to take effect.

If the argument `= {loop|procedure}` is omitted, `-Kilfunc=procedure` is set.

-Knoifunc is set by default.

If the compiler determines that the optimization is not promoted due to such as data type and the presence of function calls, inline expansion is not applied.

Inline expansion is applied by the -Kilfunc option, side effects may occur in the execution result. Also, this optimization may result in generating a floating-point exception when -NRtrap is not valid. See Section "3.6.1 Side Effect of Optimizations for Floating-Point Operation" for the side effect of optimization.

The -Kilfunc and -Knoifunc options require that the -O1 option or higher is set.

-Kilfunc=loop

Specifies to perform the inline expansion for the math functions in a loop.

-Kilfunc=procedure

Specifies to perform the inline expansion for the math functions in a function.

-Kinstance=*N*

Specifies the number of threads for parallel execution as *N*. *N* should be from 2 to 512.

This option requires that the -Kparallel option is set.

When the number of threads at run-time does not equal the value of *N*, the following message is output and execution is discontinued.

```
jwe1040i-s This program cannot be executed, because the number of threads specified by -Kinstance=N
option and the actual number of threads are not equal.
```

Before using this option, see Section "4.2.7.1 Multiprocessing of Nested Loops".

-K{largepage|nolargepage}

-Klargepage creates an executable program which uses the large page function. The -Knolargepage option invalidates the -Klargepage option. -Klargepage is set by default.

The -Klargepage and -Knolargepage option must be set at linking.

-K{lib|nolib}

-Klib promotes optimization by recognizing the operation of standard library functions. The -Knolib option invalidates the -Klib option. -Knolib is set by default when the -O0 option is set. -Klib option is set by default when the -O2 option or higher is set.

If a user-defined function with the same name as a standard library function is used, unexpected results may occur.

The -Klib and -Knolib options require that the -O1 option or higher is set.

The recognized standard library functions are as follows:

```
abort, abs, acos, acosf, acosh, acoshf, asin, asinf, asinh, asinhf, atan, atan2, atan2f, atanf,
atanh, atanhf, calloc, cbrt, cbrtf, ceil, ceilf, clearerr, copysign, copysignf, cos, cosf, cosh,
coshf, csqrt, csqrtf, erf, erfc, erfcf, erff, exit, exp, exp2, exp2f, expf, expm1, expm1f, fabs,
fabsf, fclose, fdim, fdimf, feof, ferror, fflush, fgetc, fgetpos, fgets, floor, floorf, fma, fmaf,
fmax, fmaxf, fmin, fminf, fmod, fmodf, fputc, fputs, free, frexp, frexpf, fseek, fwrite, getenv,
hypot, hypotf, ilogb, ilogbf, isalnum, isalpha, iscntrl, isdigit, isgraph, islower, ispunct,
isspace, isupper, isxdigit, ldexp, ldexpf, lgamma, lgammaf, llrint, llrintf, llround, llroundf,
log, log10, log10f, loglp, loglpf, log2, log2f, logb, logbf, logf, lrint, lrintf, lround, lroundf,
malloc, memchr, memcmp, memcpy, memmove, memset, modf, nearbyint, nearbyintf, nextafter,
nextafterf, nexttoward, nexttowardf, perror, pow, powf, printf, putchar, rand, realloc, remainder,
remainderf, remove, remquo, remquoof, rename, rint, rintf, round, roundf, scalbln, scalblnf, scalbn,
scalbnf, scanf, setvbuf, sin, sinf, sinh, sinh, sprintf, sqrt, sqrtf, srand, sscanf, strcat,
strchr, strcmp, strcpy, strcspn, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strrchr,
strspn, strstr, strtod, strtok, strtol, strtoul, system, tan, tanf, tanh, tanhf, tgamma, tgammaf,
tolower, toupper, trunc, truncf, vprintf, vsprintf
```

-K{loop_blocking[=*N*]|loop_noblocking}

-Kloop_blocking performs optimization by loop blocking. The -Kloop_noblocking option invalidates the -Kloop_blocking option. -Kloop_noblocking is always applied when the -O1 option is effective. -Kloop_blocking is set by default when the -O2 option or higher is set.

N indicates the block size which should be from 2 to 10000. If N is omitted, the compiler automatically determines a suitable value for N .

The `-Kloop_blocking` and `-Kloop_noblocking` options require that the `-O1` option or higher is set.

For details about the loop blocking, see Section "[3.2.5 Loop Blocking](#)".

`-K{loop_fission|loop_nofission}`

`-Kloop_fission` specifies to perform the loop fission optimization for promotion of the software pipelining, improvement of the cache memory efficiency, and the resolution of register shortage. The `-Kloop_nofission` option invalidates the `-Kloop_fission` option. `-Kloop_nofission` is always applied when the `-O1` option is set. `-Kloop_fission` is set by default when the `-O2` option or higher is set.

The target loops and fission points of the loop fission optimization are specified by the following optimization control specifiers.

- `loop_fission_target` specifier

This optimization control specifier fissions the specified loop automatically.

- `fission_point` specifier

This optimization control specifier fissions the loop at the specified executable statement.

See Section "[3.4.1 Using Optimization Control Line \(OCL\)](#)" for information about `loop_fission_target` and `fission_point` specifiers. Note that you must specify the `-Kocl` option to validate optimization control specifiers.

The `-Kloop_fission` and `-Kloop_nofission` options require that the `-O1` option or higher is set.

The `-Kloop_fission` and `-Kloop_nofission` options work only when used in combination with above optimization control specifiers.

For details about this function, see Section "[3.3.10 Loop Fission](#)".

`-K{loop_fission_stripmining[={N|L1|L2}]|loop_nofission_stripmining}`

`-Kloop_fission_stripmining` specifies to perform optimization of strip-mining at the automatic loop fission. The `-Kloop_fission_stripmining` option invalidates the `-Kloop_nofission_stripmining` option.

If the argument $=\{N|L1|L2\}$ is omitted, the compiler choose the strip length automatically. `-Kloop_nofission_stripmining` is set by default.

By this optimization, improvement of the cache memory efficiency is expected for the data accessed between fissioned loops.

The `-Kloop_fission_stripmining` option requires that the `-Kloop_fission` option, the `-Kocl` option, and the `-O2` option or higher are set and the `loop_fission_target` specifier is specified in the optimization control line.

For details about this function, see Section "[3.3.10.1 Strip-Mining](#)".

`-Kloop_fission_stripmining=N`

N indicates the strip length. N should be an integer value from 2 to 100000000.

`-Kloop_fission_stripmining=L1`

The strip length is adjusted to the size of the level 1 cache for the cache memory efficiency.

`-Kloop_fission_stripmining=L2`

The strip length is adjusted to the size of the level 2 cache for the cache memory efficiency.

`-Kloop_fission_threshold=N`

Specifies the threshold N to decide the granularity of loops after automatic loop fission. N is an integer value from 1 to 100. If the value of N is reduced, the fissioned loops tend to become small and the number of the fissioned loops tends to increase. `-Kloop_fission_threshold=50` is set by default.

This option requires that the `-Kloop_fission` option, the `-Kocl` option, and the `-O2` option or higher are set and the `loop_fission_target` specifier is specified in the optimization control line.

`-K{loop_fusion|loop_nofusion}`

`-Kloop_fusion` fuses neighboring loops. The `-Kloop_nofusion` option invalidates the `-Kloop_fusion` option. `-Kloop_nofusion` is always applied when the `-O1` option is set. `-Kloop_fusion` is set by default when the `-O2` option or higher is set.

The `-Kloop_fusion` and `-Kloop_nofusion` options require that the `-O1` option or higher is set.

-K{loop_interchange|loop_nointerchange}

-Kloop_interchange exchanges loops. The -Kloop_nointerchange option invalidates the -Kloop_interchange option. -Kloop_nointerchange is always applied when the -O1 option is set. -Kloop_interchange is set by default when the -O2 option or higher is set.

The -Kloop_interchange and -Kloop_nointerchange options require that the -O1 option or higher is set.

-K{loop_part_parallel|loop_nopart_parallel}

-Kloop_part_parallel specifies that when a loop contains statements to which parallelization can be applied and statements to which parallelization cannot be applied, the loop is divided into two or more loops and automatic parallelization is applied to one of the loops. This optimization is applied to the innermost loop. The -Kloop_nopart_parallel option invalidates the -Kloop_part_parallel option. -Kloop_nopart_parallel is set by default.

Note that compile time and execution time may increase.

The -Kloop_part_parallel and -Kloop_nopart_parallel options require that the -Kparallel option is set.

-K{loop_part_simd|loop_nopart_simd}

-Kloop_part_simd specifies that when a loop contains instructions to which SIMD extensions can be applied and instructions to which SIMD extensions cannot be applied, the loop is divided into two or more loops and SIMD extensions are applied to one of the loops. This optimization is applied to the innermost loop. The -Kloop_nopart_simd option invalidates the -Kloop_part_simd option. -Kloop_nopart_simd is set by default.

Note that compile time and execution time may increase.

The -Kloop_part_simd and -Kloop_nopart_simd options require that the -Ksimd[={ 1|2|auto}] option is set.

-K{loop_perfect_nest|loop_noperfect_nest}

-Kloop_perfect_nest performs optimization which fissions the imperfectly nested loop into the perfectly nested loops. The -Kloop_noperfect_nest option invalidates the -Kloop_perfect_nest option. The -Kloop_noperfect_nest is set by default when the -O2 option is set, and the -Kloop_perfect_nest is set by default when the -O3 option is set.

This optimization may promote optimizations such as the loop interchange and loop collapsing.

The -Kloop_perfect_nest and -Kloop_noperfect_nest options require that the -O2 option or higher is set.

-K{loop_versioning|loop_noversioning}

-Kloop_versioning performs optimization by the loop versioning. The -Kloop_noversioning option invalidates the -Kloop_versioning option. -Kloop_noversioning is set by default.

The loop versioning may promote optimizations such as SIMD extension, software pipelining, or automatic parallelization.

Note that the size of the object program and compile time may increase because the loop versioning generates two loops. Moreover, the execution performance of the program may decrease due to the overhead of judgement for the choice of generated loops.

The -Kloop_versioning and -Kloop_noversioning options require that the -O2 option or higher is set.

For details about the loop versioning, see Section "[3.3.6 Loop Versioning](#)".

-Klooptype={f|n|s}

Specifies loop continuity conditions and direction. This option may promote optimizations. -Klooptype=f is set by default.

-Klooptype=f

Declares that the program does not contain an infinite loop but may contain a wrap-around loop.

-Klooptype=n

Declares that the program may contain an infinite loop or wrap-around loop.

-Klooptype=s

Declares that the program does not contain an infinite or wrap-around loop.

-K{memalias|nomemalias}

-Kmemalias specifies that in the case of indirect memory access through a pointer, when the accessing types are different, no memory alias is set. The -Knomemalias option invalidates the -Kmemalias option. -Knomemalias is set by default.

The -Kmemalias option may promote optimizations.

The -Kmemalias and -Knomemalias options require that the -O1 option or higher is set.

The -Kmemalias and -Knomemalias options are deprecated and will be removed in the future.

-K{mfunc={1|2|3}}[nomfunc]

-Kmfunc specifies to apply the optimization which changes a function into a multi-operation function. A multi-operation function is a function achieving improved execution performance using one tuned function call instead of performing multiple functions. If the argument ={1|2|3} is omitted, -Kmfunc=1 is set. The -Knomfunc option invalidates the -Kmfunc option. -Knomfunc is set by default.

When the -Kmfunc option is set, side effects may occur in the execution result. See Section "[3.6.1 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

The -Kmfunc and -Knomfunc options require that the -O2 option or higher is set.

The -Klib option is required for the -Kmfunc option to take effect.

The following table shows the function of the target.

Functions	Float Type	Double Type
acos	* (a)	* (a)
asin	* (a)	* (a)
atan	*	*
atan2	*	*
cos	*	*
erf	* (a)	* (a)
erfc	* (a)	* (a)
exp	*	*
log	*	*
log10	*	*
sin	*	*
pow	*	*

*: To be a target

-: Not to be a target

a) These multi-operation functions execute sequential instructions internally. The execution performance improves because optimizations, such as branch optimization, are applied to the internal instructions. The improvement is inferior to that of the normal multi-operation functions.

-Kmfunc=1

A multi-operation function of multiplicities which is the same as the SIMD width is used.

-Kmfunc=2

In addition to the function of -Kmfunc=1, a multi-operation function of multiplicities automatically determined by the compiler is also used. This function needs a large stack area.

-Kmfunc=3

In addition to the function of -Kmfunc=2, a multi-operation function of multiplicities automatically determined by the compiler is also used in a loop which contains an "if" statement or similar. This function needs a large stack area. The execution performance may be worse than with -Kmfunc=1 or -Kmfunc=2 when the true ratio of the "if" statement is low.

Moreover, this optimization has side effects which may cause abend. See Section "[3.3.3.2 Effects of Compiler Option -Kmfunc=3](#)" for more information about the side effects.

-Knoprefetch

Specifies to generate an object without a prefetch instruction.

-K{ocl|noocl}

-Kocl specifies that the optimization control line (OCL) is effective. The -Knoocl option invalidates the -Kocl option. -Kocl is set by default.

The -Kocl and -Knoocl options require that the -O1 option or higher is set.

-K{omitfp|noomitfp}

The -Komitfp option performs the optimization that the frame pointer register is not kept for a function call. The -Knomitfp option invalidates the -Komitfp option. -Knomitfp is set by default.

Note that when the -Komitfp option is specified, the trace back information is not kept.

The -Komitfp and -Knomitfp options require that the -O1 option or higher is set.

-K{openmp|noopenmp}

-Kopenmp specifies to enable the Specification of OpenMP Application Program Interface. The -Knoopenmp option invalidates the -Kopenmp option. -Knoopenmp is set by default.

When the -Kopenmp option is specified, -mt is assumed.

The -Kopenmp option is needed if an object program compiled with it exists in the command line as an input file.

When the -O1 option or less is effective, SIMD Extensions are not used even if simd construct or declare simd construct is effective.

-K{openmp_assume_norecurrence|openmp_noassume_norecurrence}

These options direct whether or not to promote optimizations by assuming that array elements of the chunk size have no data dependency over iteration for the loop with the OpenMP "for" directive. -Kopenmp_noassume_norecurrence is set by default.

When -Kopenmp_assume_norecurrence is effective, SIMD extensions, software pipelining, and other optimizations are promoted.

The -Kopenmp_assume_norecurrence and -Kopenmp_noassume_norecurrence options are applied to the innermost loops with the OpenMP "for" directive.

The -Kopenmp_assume_norecurrence and -Kopenmp_noassume_norecurrence options are effective only when the -Kopenmp option and -O2 option or higher is set.

-K{openmp_collapse_except_innermost|openmp_nocollapse_except_innermost}

In the loop construct of OpenMP, this option specifies whether to remove the loop from the object of collapse when it meets all the following conditions:

- The collapse clause that includes the innermost loop is specified.
- When the compiler can judge that there is a possibility that the execution performance decreases by collapse that includes the innermost loop at the compile time.

When the -Kopenmp_collapse_except_innermost is specified, the loop is removed from the object of collapse. -Kopenmp_nocollapse_except_innermost is set by default.

The decrease of execution performance by collapse might be able to be prevented by -Kopenmp_collapse_except_innermost option.

The -Kopenmp_collapse_except_innermost and -Kopenmp_nocollapse_except_innermost options are effective only when the -Kopenmp option is set.

The loop removed from the object of collapse can be known from the diagnostic message by setting the -Koptmsg=2 option at the compile time.

-Kopenmp_loop_variable={private|standard}

Indicates whether the sequential loop control variable in the OpenMP parallel syntax should be treated as private or shared. The -Kopenmp_loop_variable=private option is default.

This option is effective only when the -Kopenmp option is set.

-Kopenmp_loop_variable=private

Indicates the sequential loop control variable in the OpenMP parallel syntax is treated as `private`.

The handling of the loop control variable does not conform to the standard.

-Kopenmp_loop_variable=standard

Indicates the sequential loop control variable in the OpenMP parallel syntax is treated as `shared`.

The handling of the loop control variable conforms to the standard.

-K{openmp_ordered_reduction|openmp_noordered_reduction}

-Kopenmp_ordered_reduction specifies to fix the order of the reduction operations same as in the numerical order of the threads at the end of the region for which the `reduction` clause of OpenMP was specified. The **-Kopenmp_noordered_reduction** option invalidates the **-Kopenmp_ordered_reduction** option. **-Kopenmp_noordered_reduction** is set by default.

If the number of threads used is identical, by fixing the order of the reduction operations same as in the numerical order of the threads, identical results will be always obtained. However, rounding errors may occur when the operation order is changed by the effect of the scheduling of a loop construct with a dynamic or a guided schedule kind, or a `sections` construct. Note that the execution performance may decrease compared to when the **-Kopenmp_ordered_reduction** option is invalidated.

The **-Kopenmp_ordered_reduction** and **-Kopenmp_noordered_reduction** options require that the **-Kopenmp** option is set.

-K{openmp_simd|noopenmp_simd}

-Kopenmp_simd specifies that only the `simd` construct and the `declare simd` construct of OpenMP are enabled.

-Knoopenmp_simd is set by default.

When the **-Kopenmp_simd** option is specified, only the SIMD Extensions based on the OpenMP specifications are applied, and the parallelization is not applied.

When the **-Knoopenmp_simd** option is specified, the **-Kopenmp_simd** option is disabled.

When the **-O1** option or less is enabled, SIMD Extensions are not used even if `simd` construct or `declare simd` construct is enabled.

-K{optlib_string|nooptlib_string}

-Koptlib_string specifies the compiler to link the library of the optimized version statically about the string handling functions. The **-Knooptlib_string** option invalidates the **-Koptlib_string** option. **-Knooptlib_string** is set by default.

-Koptlib_string option is effective only when the **-KSVE** option and **-KA64FX** option are set.

These options must be set at linking.

The following shows the string handling functions for which the library of the optimized version can be linked.

<code>bcopy, bzero, memchr, memcmp, memccpy, memcpy, memmove, memset, strcat, strcmp, strcpy, strlen, strncmp, strncpy, strncat</code>
--

-K{optmsg[={1|2}]|nooptmsg}

-Koptmsg specifies to output the messages of applied optimizations. If the argument `={1|2}` is omitted, **-Koptmsg=1** is set. The **-Knooptmsg** option invalidates the **-Koptmsg** option. **-Knooptmsg** is set by default.

The **-Koptmsg** and **-Knooptmsg** options require that the **-O1** option or higher is set.

-Koptmsg=1

This option outputs messages about applied optimizations with side effects on the execution result.

-Koptmsg=2

In addition to the output of **-Koptmsg=1**, messages are output about optimizations such as automatic parallelization, SIMD extend instruction conversion, and loop unrolling.

-K{parallel|noparallel}

The compiler option **-Kparallel** specifies to perform automatic parallelization. However, if the effect of parallel execution is not expected, automatic parallelization is not performed. The compiler option **-Knoparallel** invalidates the compiler option **-Kparallel**. The compiler option **-Knoparallel** is set by default.

If the compiler option `-Kparallel` is set, the compiler options `-O2`, `-Kregion_extension`, `-Kloop_part_parallel`, `-Kloop_perfect_nest`, and `-mt` also take effect. Note that if the compiler option `-O3` is set at the same time, the compiler option `-O3` takes effect.

The compiler option `-Kparallel` is invalidated if the compiler option `-O0` or `-O1` is set.

The compiler option `-Kparallel` is needed if an object program compiled with it exists in the command line as an input file.

`-K{parallel_fp_precision|parallel_nofp_precision}`

`-Kparallel_fp_precision` specifies that the compiler controls to apply optimizations considering calculation error of a floating type or a complex type operation that is caused by the difference of the parallel number of threads. The `-Kparallel_nofp_precision` option invalidates the `-Kparallel_fp_precision` option. `-Kparallel_nofp_precision` is set by default.

When the `-Kparallel_fp_precision` option is specified, the compiler applies optimizations within a range in which no calculation errors occur.

The `-Kparallel_fp_precision` and `-Kparallel_nofp_precision` options require that the `-Kparallel` or `-Kopenmp` option is set.

When the `-Kparallel_fp_precision` option and `-Kopenmp` option are set, `-Kopenmp_ordered_reduction` option is valid.

When the `-Kparallel_fp_precision` option is set, the execution performance may decrease because the part of optimization is restricted.

Note that the calculation error of a floating type or a complex type operation that is caused by the difference of the parallel number of threads even if the `-Kparallel_fp_precision` option is valid when the reduction clause of OpenMP is specified.

`-Kparallel_iteration=N`

Specifies that parallelization targets only the loop which has proved that the iteration counts is *N* or more at compilation. *N* can be specified from 1 to 2147483647.

This option requires that the `-Kparallel` option is set.

`-Kparallel_strong`

Specifies to perform automatic parallelization for all loops that can be made parallel without estimating parallelization effects.

This option sets the `-Keval` and `-Kpreex` options.

Aside from this point, the function and notes are same as for the `-Kparallel` option.

`-K{pc_relative_literal_loads|nopc_relative_literal_loads}`

`-Kpc_relative_literal_loads` option specifies to limit the size of text data area in the function to 1MB, and access to the literal pool by one instruction. The `-Knopc_relative_literal_loads` option invalidates the `-Kpc_relative_literal_loads` option. `-Knopc_relative_literal_loads` is set by default.

`-Kpc_relative_literal_loads` and `-Knopc_relative_literal_loads` options are effective at compilation time only.

`-K{plt|noplt}`

`-Kplt` specifies to use Procedure Linkage Table (PLT) for accessing a global symbol in the position-independent code (PIC). `-Knoplt` option invalidates the `-Kplt` option. `-Kplt` is set by default.

`-Kplt` and `-Knoplt` options are effective only if the `-KPIC` option or the `-Kpic` option is set.

`-K{preex|nopreex}`

`-Kpreex` specifies to evaluate invariant expressions in advance. The `-Knopreex` option invalidates the `-Kpreex` option. `-Knopreex` is set by default.

When the `-Kpreex` option is set, side effects (calculation error or runtime exceptions) may occur in the execution results because instructions that may not to be executed based on the logic of the program are likely to be executed. For detail of the side effects, see Section "[3.3.1.1 Advance Evaluation of Invariant Expressions](#)".

The `-Kpreex` and `-Knopreex` options require that the `-O1` option or higher is set. Note that a floating-point exception may occur when the `-Knopreex`, `-Kfp_relaxed`, and `-NRtrap` options are valid.

`-Kprefetch_cache_level={1|2|all}`

Specifies the data cache-level of prefetch instructions. By default, `-Kprefetch_cache_level=all` is set.

This option requires that at least one of the `-Kprefetch_indirect`, `-Kprefetch_sequential`, or `-Kprefetch_stride` options is set.

-Kprefetch_cache_level=1

Data is prefetched in the first level cache. Normal prefetch instructions are used.

-Kprefetch_cache_level=2

Data is prefetched in the second level cache.

-Kprefetch_cache_level=all

Both -Kprefetch_cache_level=1 and -Kprefetch_cache_level=2 functions are effective. By using two levels of prefetch instructions, the prefetch function becomes more sophisticated.

-K{prefetch_conditional|prefetch_noconditional}

-Kprefetch_conditional specifies to generate prefetch instructions for array data in the block included in an "if" statement and a "switch" statement. The -Kprefetch_noconditional option invalidates the -Kprefetch_conditional option. -Kprefetch_noconditional is set by default.

The -Kprefetch_conditional and -Kprefetch_noconditional options require that at least one of the -Kprefetch_indirect, -Kprefetch_sequential, or -Kprefetch_stride options is set.

-K{prefetch_indirect|prefetch_noindirect}

-Kprefetch_indirect specifies to generate prefetch instructions for array data accessed indirectly (list access) within a loop. The -Kprefetch_noindirect option invalidates the -Kprefetch_indirect option. -Kprefetch_noindirect is set by default.

The -Kprefetch_indirect and -Kprefetch_noindirect options require that the -O1 option or higher is set.

-K{prefetch_infer|prefetch_noinfer}

-Kprefetch_infer specifies to generate the prefetch instructions under the assumption that the memory access is continuous. The -Kprefetch_noinfer option invalidates the -Kprefetch_infer option. -Kprefetch_noinfer is set by default.

The -Kprefetch_infer and -Kprefetch_noinfer options require that at least one of the -Kprefetch_indirect, -Kprefetch_sequential, or -Kprefetch_stride options is set.

-Kprefetch_iteration=N

Specifies to prefetch data which is referred to or defined in a loop after *N* iteration(s). *N* should be from 1 to 10000. If the optimization using SIMD extensions is applied to a loop, specify the loop iteration count after using SIMD extensions for *N*.

This option specifies to prefetch data to only the first level cache.

This option requires that at least one of the -Kprefetch_indirect, -Kprefetch_sequential, or -Kprefetch_stride options is set, and the -Kprefetch_cache_level=1 or -Kprefetch_cache_level=all is set.

This option cannot be specified simultaneously with the -Kprefetch_line option.

-Kprefetch_iteration_L2=N

Specifies to prefetch data which is referred to or defined in a loop after *N* iteration(s). *N* should be from 1 to 10000. If the optimization using SIMD extensions is applied to a loop, specify the loop iteration count after using SIMD extensions for *N*.

This option specifies to prefetch data to only the second level cache.

This option requires that at least one of the -Kprefetch_indirect, -Kprefetch_sequential, or -Kprefetch_stride options is set, and -Kprefetch_cache_level=2 or -Kprefetch_cache_level=all is set.

This option cannot be specified simultaneously with the -Kprefetch_line_L2 option.

-Kprefetch_line=N

Specifies to prefetch data which is referred to or defined in a line after *N* line(s). *N* should be from 1 to 100.

This option specifies to prefetch data to only the first level cache.

This option requires that at least one of the -Kprefetch_indirect or -Kprefetch_sequential options is set, and -Kprefetch_cache_level=1 or -Kprefetch_cache_level=all is set.

This option cannot be specified simultaneously with the -Kprefetch_iteration option.

-Kprefetch_line_L2=N

Specifies to prefetch data which is referred to or defined in a line after *N* line(s). *N* should be from 1 to 100. This option specifies to prefetch data to only the second level cache.

This option requires that at least one of the -Kprefetch_indirect or -Kprefetch_sequential options is set, and -Kprefetch_cache_level=2 or -Kprefetch_cache_level=all is set.

This option cannot be specified simultaneously with the -Kprefetch_iteration_L2 option.

-K{prefetch_sequential[={auto|soft}]]prefetch_nosequential}

-Kprefetch_sequential specifies to generate prefetch instructions for prefetch array data accessed sequentially within a loop. The -Kprefetch_nosequential option invalidates the -Kprefetch_sequential option. When the -O1 option is set, -Kprefetch_nosequential is set by default. When the -O2 option or higher is set, -Kprefetch_sequential is set by default.

If the argument = {auto|soft} is omitted, -Kprefetch_sequential=auto is assumed.

The -Kprefetch_sequential and -Kprefetch_nosequential options require that the -O1 option or higher is set.

-Kprefetch_sequential=auto

The compiler automatically selects whether to use the hardware prefetch function or to output the prefetch instruction for array data accessed sequentially within a loop.

-Kprefetch_sequential=soft

The compiler specifies to output the prefetch instruction for array data accessed sequentially within a loop without using the hardware prefetch function.

-K{prefetch_stride[={soft|hard_auto|hard_always}]]prefetch_nostride}

-Kprefetch_stride specifies to perform the prefetch for array data with a stride larger than the cache line size in its loop. This includes loops with prefetch addresses not defined at compilation. The -Kprefetch_nostride option invalidates the -Kprefetch_stride option. -Kprefetch_nostride is set by default.

If "= {soft|hard_auto|hard_always}" of the -Kprefetch_stride option is omitted, the -Kprefetch_stride=soft option is set by default.

The -Kprefetch_stride=soft and -Kprefetch_nostride options require that the -O1 option or higher is set.

The -Kprefetch_stride=hard_auto and -Kprefetch_stride=hard_always options require that the -Khpctag option and the -O1 option or higher are set.

-Kprefetch_stride=soft

The -Kprefetch_stride=soft option specifies to generate prefetch instructions and perform the prefetch.

-Kprefetch_stride=hard_auto

The -Kprefetch_stride=hard_auto option specifies to perform the prefetch using the hardware stride prefetcher.

This option sets the stride prefetcher to prefetch only data that is not on cache.

-Kprefetch_stride=hard_always

The -Kprefetch_stride=hard_always option specifies to perform the prefetch using the hardware stride prefetcher.

In contrast with the -Kprefetch_stride=hard_auto option, this option sets the stride prefetcher to always prefetch.

-K{prefetch_strong|prefetch_nostrong}

-Kprefetch_strong specifies to generate strong prefetch instructions. The -Kprefetch_nostrong option invalidates the -Kprefetch_strong option. -Kprefetch_strong is set by default.

The -Kprefetch_strong option specifies to prefetch data to only the first level cache.

The -Kprefetch_strong and -Kprefetch_nostrong options require that at least one of the -Kprefetch_indirect, -Kprefetch_sequential, or -Kprefetch_stride options is set, and the -Kprefetch_cache_level=1 or -Kprefetch_cache_level=all option is set.

The -Kprefetch_nostrong option requires that the -Khpctag option is set.

-K{prefetch_strong_L2|prefetch_nostrong_L2}

-Kprefetch_strong_L2 specifies to generate strong prefetch instructions. The -Kprefetch_nostrong_L2 option invalidates the -Kprefetch_strong_L2 option. -Kprefetch_strong_L2 is set by default.

The `-Kprefetch_strong_L2` option specifies to prefetch data to only the second level cache.

The `-Kprefetch_strong_L2` and `-Kprefetch_nostrong_L2` options require that at least one of the `-Kprefetch_indirect`, `-Kprefetch_sequential`, or `-Kprefetch_stride` options is set, and the `-Kprefetch_cache_level=2` or `-Kprefetch_cache_level=all` option is set.

The `-Kprefetch_nostrong_L2` option requires that the `-Khpctag` option is set.

`-K{preload|nopreload}`

`-Kpreload` specifies to perform the speculative execution of load instructions. By specifying the `-Kpreload` option, the optimization of instruction scheduling is promoted, and the execution performance is expected to improve.

The `-Knopreload` option invalidates the `-Kpreload` option. The `-Knopreload` is set by default.

Note that this optimization may cause interruption of execution of the program by the execution of load instructions referring to illegal areas that are not to be executed logically in the program.

The `-Kpreload` option is effective only when the `-O1` option or higher is set.

`-K{rdconv[={1|2}]]nordconv}`

`-Krdconv` specifies to perform on the assumption that expressions of integer type whose size is 4-byte or less do not overflow or wrap around when the expressions increase or decrease by a constant value as the loop repeats.

The `-Knordconv` option invalidates the `-Krdconv` option. If `"={1|2}"` of `-Krdconv` is omitted, the `-Krdconv=1` option is assumed to be set.

`-Krdconv=1` is set by default when the `-O2` or higher option is set.

If the expression overflows or wrap around, the result may differ depending on the presence or absence of the `-Krdconv` option.

These options require that the `-O2` option or higher is set.

`-Krdconv=1`

This option specifies to perform optimizations on the assumption that expressions of signed integer type whose size is 4-byte or less do not overflow when the expressions increase or decrease by a constant value as the loop repeats.

`-Krdconv=2`

In addition to the features of the `-Krdconv=1` option, this option specifies to perform on the assumption that expressions of unsigned integer type whose size is 4-byte or less do not wrap around when the expressions increases or decreases by a constant value as the loop repeats.

`-K{reduction|noreduction}`

`-Kreduction` specifies to perform the reduction optimization. The `-Knoreduction` option invalidates the `-Kreduction` option. `-Knoreduction` is set by default.

When the `-Kreduction` option is set, side effects (calculation errors) may occur in the execution results. See Section ["3.6.1 Side Effect of Optimizations for Floating-Point Operation"](#) for the side effect of optimization.

The `-Kreduction` and `-Knoreduction` options require that the `-Kparallel` option is set.

Before using the `-Kreduction` option, see Section ["4.2.1.1 Compiler Option for Automatic Parallelization"](#).

`-K{region_extension|noregion_extension}`

`-Kregion_extension` specifies to perform the optimization that extends the parallel region to reduce the overhead caused by automatic parallelization.

If the `-Kregion_extension` option is specified, the execution performance may be degraded on loops with a small parallelization effect.

The `-Kregion_extension` and `-Knoregion_extension` options require that the `-Kparallel` option is set.

For details, see Section ["4.2.5.10 Parallel Region Extension"](#).

`-K{restp[={all|arg|restrict}]]norestp}`

`-Krestp` specifies to perform optimization of restricted pointers. The `-Knorestp` option invalidates the `-Krestp` option. `-Krestp=restrict` is set by default.

If the `-Krestp` option is set, side effects (calculation errors or runtime exceptions) may occur in the execution results.

The `-Krestp` and `-Knorestp` options require that the `-O1` option or higher is set.

Before using the `-Krestp` option, see Section "[3.3.2 Optimization of Pointers](#)".

`-Krestp=all`

Specifies to perform the optimization of restricted pointers assuming that the `restrict` qualifier is specified for all pointers.

`-Krestp=arg`

Specifies to perform optimization of restricted pointers assuming that the `restrict` qualifier is specified for the pointers in arguments.

`-Krestp=restrict`

Specifies to perform optimization of restricted pointers only for the `restrict` qualified pointers.

`-Krestp`

Specifies to perform optimization of restricted pointers for the `restrict` qualified pointers and for the pointers in arguments.

`-K{sch_post_ra|nosch_post_ra}`

`-Ksch_post_ra` specifies to perform instruction scheduling after register allocation. The `-Knosch_post_ra` option invalidates the `-Ksch_post_ra` option. When the `-O0` option is set, `-Knosch_post_ra` is set by default. When the `-O1` option or higher is set, `-Ksch_post_ra` is set by default.

The compiler rearranges execution instructions to improve the execution performance when the `-Ksch_post_ra` option is effective. This optimization does not increase saving and restoring instructions for registers to and from the memory.

`-K{sch_pre_ra|nosch_pre_ra}`

`-Ksch_pre_ra` specifies to perform instruction scheduling before register allocation. The `-Knosch_pre_ra` option invalidates the `-Ksch_pre_ra` option. When the `-O0` option is set, the default is `-Knosch_pre_ra`. When the `-O1` option or higher is set, the default is `-Ksch_pre_ra`.

The compiler rearranges execution instructions to improve the execution performance when the `-Ksch_pre_ra` option is effective. Note that the execution performance may decrease by increasing saving and restoring instructions for registers to and from the memory.

`-K{sibling_calls|nosibling_calls}`

`-Ksibling_calls` specifies to optimize sibling calls. The `-Knosibling_calls` option invalidates the `-Ksibling_calls` option. `-Knosibling_calls` is set by default when the `-O1` option is set. `-Ksibling_calls` is set by default when the `-O2` or higher option is set.

When the `-Ksibling_calls` option is set, the trace back information is not kept.

The `-Ksibling_calls` and `-Knosibling_calls` options require that the `-O1` option or higher is set.

`-K{simd[={1|2|auto}]}|nosimd}`

`-Ksimd` specifies to perform optimization using the SIMD Extensions for loop.

If the argument `= {1|2|auto}` is omitted, `-Ksimd=auto` is assumed. The `-Knosimd` option invalidates the `-Ksimd` option.

When the `-O2` option or higher is set, `-Ksimd=auto` is set by default.

`-Kloop_part_simd` option is also effective when the `-Ksimd[={1|2|auto}]` option is set.

The `-Ksimd` and `-Knosimd` options require that the `-O2` option or higher is set.

For details about SIMD, see Section "[3.2.7 SIMD](#)".

`-Ksimd=1`

Specifies optimization that uses the SIMD Extensions for loop excluding conditional branch (like `"if"` statement).

`-Ksimd=2`

In addition to the function of `-Ksimd=1`, `-Ksimd=2` specifies optimization that uses the SIMD Extensions for loop including conditional branch.

The generated code has redundant instructions in conditional branches, so execution time may increase according to the true-ratio of the condition.

-Ksimd=2 may produce side effects (calculation errors or runtime exceptions) in the execution results because instructions that may not be executed based on the logic of the program are likely to be executed, as the result of the speculative execution for the expressions in the "if" statement.

-Ksimd=auto

The compiler automatically determines whether to use SIMD Extensions for the loop. SIMD Extensions are promoted for loops that contain "if" statement.

-K{simd_packed_promotion|simd_nopacked_promotion}

-Ksimd_packed_promotion specifies to perform optimization promoting packed SIMD assuming that index calculations of array elements of both single-precision floating-point type and 4-byte integer type do not exceed 4-byte range. -Ksimd_nopacked_promotion is set by default.

When the -Ksimd_packed_promotion is effective and the index calculations exceed 4-byte range, the program execution may be aborted or the execution result may be incorrect by referring illegal area.

-Ksimd_packed_promotion and -Ksimd_nopacked_promotion options require that the -Ksimd={1|2|auto} option is set.

-K{simd_reduction_product|simd_noreduction_product}

-Ksimd_reduction_product specifies to perform SIMD extensions to the reduction operation of product. The -Ksimd_noreduction_product option invalidates the -Ksimd_reduction_product option. -Ksimd_noreduction_product is set by default.

When the -Ksimd_reduction_product option is set, side effects (calculation errors) may occur in the execution results. See Section ["3.6.1 Side Effect of Optimizations for Floating-Point Operation"](#) for the side effect of optimization.

The -Ksimd_reduction_product and -Ksimd_noreduction_product options require that the -Ksimd={1|2|auto} option is set.

This option cannot be specified with the -Ksimd_reg_size=agnostic option at the same time.

-Ksimd_reg_size={128|256|512|agnostic}

Specifies the size of the SVE vector register. Units are bits.

When the -Ksimd_reg_size={128|256|512} option is specified, optimizations are performed on the assumption that the vector register size is a fixed value specified as the option at compilation time. Therefore, optimizations are promoted and the improvement of the execution performance is expected. However, the generated executable program works normally only on CPU architecture which has the same size of the SVE vector register as the size specified at compilation time. For details, see Section ["3.6.2 Notes on Specified SVE Vector Register Size"](#).

When specifies -Ksimd_reg_size=agnostic, the SVE vector register is not considered to be a specific size, and the executable program decides the vector register size at execution time. The executable program does not depend on the SVE vector register size on the CPU architecture. Note that the execution performance might decrease compared with the case of -Ksimd_reg_size={128|256|512}.

-Ksimd_reg_size=512 is set by default.

This option is effective when the -KSVE option is set.

-K{simd_uncounted_loop|simd_nouncounted_loop}

-Ksimd_uncounted_loop specifies to create objects that use SIMD extension instructions for statements in a while loop, do-while loop, if-goto loop, and a "for" loop with jumps out of the loop. The -Ksimd_nouncounted_loop option invalidates the -Ksimd_uncounted_loop option. -Ksimd_nouncounted_loop is set by default.

This option is effective when the -Ksimd={1|2|auto} and -KSVE options are set.

-K{simd_use_multiple_structures|simd_nouse_multiple_structures}

-Ksimd_use_multiple_structures specifies to use the SVE Load Multiple Structures and SVE Store Multiple Structures instructions when using SIMD extensions. The execution performance is improved by utilizing the above instructions for the load and store when using SIMD extensions. Note that performance may decrease depending on data alignment. The -Ksimd_nouse_multiple_structures option invalidates the -Ksimd_use_multiple_structures option.

-Ksimd_use_multiple_structures is set by default.

This option is effective when the -Ksimd={1|2|auto} and the -KSVE options are set.

-K{strict_aliasing|nostrict_aliasing}

The **-Kstrict_aliasing** option specifies to allow optimizations with considering overlaps of memory regions according to the strict aliasing rules defined by language standard. **-Knostrict_aliasing** option is set by default.

The **-Kstrict_aliasing** and **-Knostrict_aliasing** options require that the **-O2** option or higher is set.

See section [3.3.11 Strict Aliasing](#) when specifying the **-Kstrict_aliasing** option.

-K{striping[=*N*]|nostriping}

-Kstriping specifies to apply loop striping. *N* is the striping size which should be from 2 to 100. If the argument **=*N*** is omitted, **-Kstriping=2** is assumed. The **-Knostriping** option invalidates the **-Kstriping** option. **-Knostriping** is always applied when the **-O1** option is set. The **-Knostriping** option is set by default when the **-O2** or higher option is set.

The **-Kstriping** and **-Knostriping** options requires that the **-O1** option or higher is set.

For details about loop striping, see Section "[3.3.4 Loop Striping](#)".

-K{swp|noswp}

-Kswp specifies to apply software pipelining. However, if the effect of software pipelining is not expected, software pipelining is not performed. The **-Knoswp** option invalidates the **-Kswp** option. **-Knoswp** is always applied when the **-O1** option is set. **-Kswp** is set by default when the **-O2** or higher option is set.

The **-Kswp** and **-Knoswp** options require that the **-O1** option or higher is set.

If the **-Kswp** option is specified with the **-Kswp_weak** or **-Kswp_strong** option, the one specified last is effective.

For details about software pipelining, see Section "[3.2.6 Software Pipelining](#)".

-K{swp_freq_rate=*N*|swp_ireg_rate=*N*|swp_preg_rate=*N*}

Specifies the rate (percentage) about the following registers that can be used by software pipelining.

- Floating-point register and SVE vector register
- Integer register
- SVE predicate register

N should be an integer value number from 1 to 1000. **-Kswp_freq_rate=100,swp_ireg_rate=100,swp_preg_rate=100** is set by default.

The application of the software pipelining can be adjusted by changing the condition of the number of registers. If the software pipelining is not applied due to shortage of register, it may be applied by specifying integer values that are larger than 100.

Specifying this option may increase saving and restoring instructions for registers to and from the memory, and the execution performance may decrease.

This option is effective only if the **-O2** option or higher is set.

-Kswp_freq_rate=*N*

Specifies that *N*% of the floating-point register and the SVE vector register is available when applying the software pipelining.

-Kswp_ireg_rate=*N*

Specifies that *N*% of the integer register is available when applying the software pipelining.

-Kswp_preg_rate=*N*

Specifies that *N*% of the SVE predicate register is available when applying the software pipelining.

-Kswp_policy={auto|small|large}

Specifies a policy to select an instruction scheduling algorithm used in software pipelining.

Software pipelining is performed by the **-Kswp** option, **-Kswp_weak** option, **-Kswp_strong** option, or the corresponding optimization control lines.

-Kswp_policy=auto is set by default.

-Kswp_policy=auto

The compiler automatically selects a fit algorithm for each loop.

-Kswp_policy=small

An algorithm fit for a small loop, such as a loop with low register pressure, is used.

-Kswp_policy=large

An algorithm fit for a large loop, such as a loop with high register pressure, is used.

-Kswp_strong

Specifies to perform software pipelining for more loops by easing its applicable condition.

This option may increase requirement of time and memory significantly at compilation time.

If this option is specified with the **-Kswp** or **-Kswp_weak** option, the one specified last is effective.

Aside from this point, the functions and notes are the same as for the **-Kswp** option.

-Kswp_weak

Specifies to adjust software pipelining for the target loop and reduce overlap of instructions in the loop.

When the loop iteration count is uncertain and small, the effect of the optimization is expected because the loop iteration count required to execute the software-pipelined loop becomes small.

This option may decrease the execution performance because the overlaps of instructions become small.

If this option is specified with the **-Kswp** or **-Kswp_strong** option, the one specified last is effective.

Aside from this point, the functions and notes are the same as for the **-Kswp** option.

-Ktls_size={12|24|32|48}

Specifies the size of an offset necessary for the access to Thread-Local Storage. Units are bits.

As the size of Thread-Local Storage, **-Ktls_size=12** (4K bytes), **-Ktls_size=24** (16M bytes), **-Ktls_size=32** (4G bytes) or **-Ktls_size=48** (256T bytes) can be specified.

When the size of the Thread-Local Storage exceeds the range of the offset, an error occurs at link time.

These options are invalidated when the **-KPIC** or **-Kpic** option is specified simultaneously.

-K{unroll[=*N*]|nounroll}

-Kunroll specifies to apply loop unrolling. *N* is the upper limit of the unrolling expansion number which should be from 2 to 100. If *=N* is omitted, the compiler automatically determines a suitable value for *N*. The **-Knounroll** option invalidates the **-Kunroll** option. **-Knounroll** is set by default when the **-O1** option is set. **-Kunroll** is set by default when the **-O2** or higher option is set.

The **-Kunroll** and **-Knounroll** options require that the **-O1** option or higher is set.

For details about loop unrolling, see Section "[3.2.4 Loop Unrolling](#)".

-K{unroll_and_jam[=*N*]|nounroll_and_jam}

-Kunroll_and_jam specifies to apply unroll-and-jam. The **-Knounroll_and_jam** option invalidates the **-Kunroll_and_jam** option. However, the optimization is suppressed in the following case:

- Assuming that the optimization is not effective.
- Assuming that there is a data dependency over iterations of the loops.

N is the upper limit of the unrolling expansion number which should be an integer value from 2 to 100. If *=N* is omitted, the compiler automatically determines a value for *N*. **-Knounroll_and_jam** option is set by default.

The **-Kunroll_and_jam** and **-Knounroll_and_jam** options require that the **-O2** option or higher is set.

Unroll-and-jam promotes removing common expression and improves the execution performance. However, the increase of data stream and change of the order of data access may decrease the cache efficiency and the execution performance.

This optimization is not expected to be effective for all the loops in the program, so it is recommended to use the optimization control specifier **unroll_and_jam** or **unroll_and_jam_force** for each loop rather than the **-Kunroll_and_jam** option for the entire program.

For details about unroll-and-jam, see Section "[3.3.8 Unroll-and-Jam](#)".

-Kvisimpact

Specifies to generate an optimized object for VISIMPACT (Virtual Single Processor by Integrated Multicore Parallel Architecture).

This option is equivalent to the following options:

-Kfast,parallel

This option must be set at both compilation and linking.

See Section "[3.6.1 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

-K{zfill[=*N*]|nozfill}

The -Kzfill option specifies to perform zfill optimization. The zfill optimization speeds up write operations for array data that is only written in a loop, by using an instruction that allocates space on the cache for writing (DC ZVA) without loading data from the memory. The zfill optimization works on the data *N*blocks ahead of the address pointed to by the target store instruction where one block is 256 byte-long and *N* is an integer value between 1 and 100. If a value is not specified for *N*, the compiler will automatically determine a value. The -Knozfill option invalidates the -Kzfill option. -Knozfill is always applied when the -O1 option is effective. -Knozfill is set by default when the -O2 option or higher is set.

The -Kzfill option requires that the -KA64FX option and the -O2 option or higher are set.

Note that if an object program compiled with the -Kzfill option is executed on a CPU other than the one on which a single cache write operation of a DC ZVA instruction is 256 bytes, the execution may be terminated abnormally or an incorrect result may occur. The amount of the cache write operation for a DC ZVA instruction on an A64FX processor is 256 bytes.

Performance may also be reduced under the following conditions:

- The program is not affected by memory bandwidth bottleneck.
- When the loop iteration count is too few.
- When the block size is explicitly specified with the -Kzfill=*N* option and the memory size where the data is written in the loop is smaller than *N*blocks.

This optimization is not expected to be effective for all the loops in the program, so it is recommended to use the optimization control specifier zfill for each loop rather than the -Kzfill option for the entire program.

For details about zfill, see Section "[3.3.5 zfill](#)".

2.2.2.6 -N Option

-N{Rtrap|Rnotrap}

-NRtrap specifies to catch signals. The -NRnotrap option invalidates the -NRtrap option. -NRnotrap is set by default. When the -NRtrap option is specified, the following signals are caught and the messages are output.

Signal	Message
SIGILL, SIGBUS, SIGSEGV	jwe0019i-u
SIGXCPU	jwe0017i-u
SIGFPE	jwe1017i-u

The floating-point underflow exception is caught when the environment variable FLIB_EXCEPT=u is specified.

Note that the integer divide by zero is not detected in the Fujitsu CPU A64FX with Arm architecture. See Section "[C.2.9 Integer Division Exception when the Divisor Is Zero](#)" for details.

For details about each signal, see Section "[8.2.1 Causes of Abend](#)".

If the -NRtrap option is used together with -Kpreex option and -Ksimd=2 option, the speculative execution may cause exceptions that would not normally occur.

When the compiler options -Kfp_relaxed and -NRtrap are effective and either of the compiler option -Knosimd or -KNOSVE is effective, a floating-point exception may occur. However, to avoid a floating-point divide-by-zero exception for sqrt(0.0), the optimization to convert to reciprocal approximation instructions is suppressed for the sqrt function. Therefore, the execution performance may decrease as compared with when the compiler option -NRnotrap is effective.

To enable the -NRtrap option, it must be set at both compilation and linking.

-N{cancel_overtime_compilation|nocancel_overtime_compilation}

These options specify whether to cancel the compilation if the compiler forecasts that it takes a long time (24 hours or more as a guide) to compile the program. When the -Ncancel_overtime_compilation option is specified, the compilation is canceled. The -Ncancel_overtime_compilation is set by default.

-Ncheck_cache_arraysize

The -Ncheck_cache_arraysize option inspects the sizes of arrays during the compilation and issue a warning message if the sizes could be a cause of an impact to execution performance.

This function decides that an array could be a cause of a cache conflict if its size is a multiple of that of second level cache. The array can be rewritten so as not to have a size of multiple of that of the cache in order to avoid the impact.

The warning message is not issued for a variable-length array. Also, this function cannot judge correctly when the Sector cache is active.

-N{clang|noclang}

-Nclang option specifies that the objects are generated using Clang Mode. -Nnoclang option invalidates -Nclang option. -Nnoclang is set by default.

For details about Clang Mode, see "[Chapter 9 Clang Mode](#)".

-N{coverage|nocoverage}

The -Ncoverage option specifies to generate the information for the code coverage. -Ncoverage option should be specified when programs are compiled and linked. -Nnocoverage is set by default.

If the -Ncoverage option specifies, the execution performance may decrease due to the instructions that measures the execution time are added in the object program.

For details about the code coverage, see section "[Appendix F Code Coverage](#)".

-N{exceptions|noexceptions}

The -Nexceptions option defines the __EXCEPTIONS macro. The -Nnoexceptions option invalidates the -Nexceptions option. -Nnoexceptions is set by default.

-N{fjcex|nofjcex}

-Nfjcex specifies to use the Fujitsu Extended Functions in this system. The -Nnofjcex option invalidates the -Nfjcex option. -Nnofjcex is set by default.

The -Nfjcex option must be set at both compilation and linking.

For details about the extended functions, see "[Appendix G Fujitsu Extended Functions](#)".

-N{fjprof|nofjprof}

-Nfjprof option specifies to enable the Profiler function. The -Nnofjprof option invalidates the -Nfjprof option. These options must be set at linking. The default is -Nfjprof.

See the "Profiler User's Guide" for information on the Profiler.

-N{hook_func|nohook_func}

-Nhook_func specifies to use the hook function that is called from a specified location. The -Nnohook_func option invalidates the -Nhook_func option. -Nnohook_func is set by default.

If the -Nhook_func option is set, the user-defined function is called from the following locations:

- Program entry and exit
- Function entry and exit
- Parallel region (OpenMP or automatic parallelization) entry and exit

This option must be set when compiling and linking programs.

See Section "[8.3 Hook Function](#)", for information about the hook function.

-N{hook_time|nohook_time}

-Nhook_time specifies to use a hook function that is called at regular time interval. The -Nnohook_time option invalidates the -Nhook_time option. -Nnohook_time is set by default.

If the -Nhook_time option is set, the user-defined function is called at regular time interval.

The interval time to call the user-defined function is specified by environment variable FLIB_HOOK_TIME. If environment variable FLIB_HOOK_TIME is not specified, the user-defined function is called at the interval of a minute.

For details about the environment variable FLIB_HOOK_TIME, see Section "[2.5.1 Environment Variable for Execution](#)".

This option must be set at linking.

See Section "[8.3 Hook Function](#)", for information about the hook function.

-N{libomp|fjompilib}

Specifies the library for multiprocessing. -Nlibomp option is set by default. If -Nclang option is set, -Nfjompilib option is disabled and -Nlibomp option is enabled. These options must be set at linking.

-Nlibomp

Specifies to use LLVM OpenMP Library for multiprocessing. See "[Chapter 4 Multiprocessing](#)" for LLVM OpenMP Library.

-Nfjompilib

Specifies to use Fujitsu OpenMP Library for multiprocessing. See "[Appendix J Fujitsu OpenMP Library](#)" for Fujitsu OpenMP Library.

-N{line|noline}

-Nline generates additional information required to use the execution time sampling function provided with the Profiler. The -Nnoline option invalidates the -Nline option.

-Nline is set by default.

-Nlst[={p|t}]

Specifies to output compilation information to file(s) whose name is with suffix ".lst". When more than one C source files are specified, the file names are "*each-file-name.lst*".

When the -Nlst option is specified without arguments, -Nlst=p is set.

For details on the output format on statements specified with the OpenMP C directives, see "[Chapter 4 Multiprocessing](#)".

-Nlst=p

Specifies to write compilation information that consists of source list and statistics list.

The source list includes annotation of optimization which shows the situation of automatic parallelization, inline expansion, loop unrolling, etc.

-Nlst=t

Specifies to write detail optimization information to source list in addition to the function of -Nlst=p.

-Nlst_out=*file*

Specifies the filename to output compilation information.

When this option is specified, the -Nlst=p option is also effective.

-Nprofile_dir=*dir_name*

Specifies storage location directory of the .gcda file which is necessary for the use of the code coverage. For *dir_name*, the storage location directory name is specified by the relative path or the absolute path.

The .gcda file is generated when the executable program linked with object programs containing information for coverage is executed. If the directory specified as *dir_name* does not exist at the execution time, the directory is generated.

This option requires that the -Ncoverage option and the -S or -c option are set.

For details about the code coverage, see section "[Appendix F Code Coverage](#)".

-Nquickdbg={subchk|nosubchk|heapchk|noheapchk|inf_detail|inf_simple}}

The **-Nquickdbg** option is used to debug C programs. The debugging function includes embedded information for debugging to a C object file, and checks are performed automatically during execution. Checks are performed if the **-Nquickdbg=subchk** or **-Nquickdbg=heapchk** option is set.

Multiple debugging functions can be used in combination by specifying multiple **-Nquickdbg** values. The **-Nquickdbg=inf_detail** and **-Nquickdbg=inf_simple** options are enabled with at least one of the **-Nquickdbg**, **-Nquickdbg=subchk**, or **-Nquickdbg=heapchk** options.

Omission of an argument is equivalent to specifying **-Nquickdbg=subchk,quickdbg=heapchk**.

If the **-Nquickdbg=subchk** or **-Nquickdbg=heapchk** option is enabled, the **-Nquickdbg=inf_detail** option is also enabled.

This option must be set when compiling and linking programs.

See Section "[8.1 Functions for Debugging](#)" for details about this function.

-Nquickdbg={subchk|nosubchk}

Specifies whether to check the validity of the referenced subscript range against the declared array size when arrays are referenced. See Section "[8.1.1 Subscript Range Checks \(subchk Function\)](#)" for details.

-Nquickdbg=subchk

The subscript range is checked. If an error is detected, a diagnostic message is output.

-Nquickdbg=nosubchk

The subscript range is not checked.

-Nquickdbg={heapchk|noheapchk}

Specifies whether to check the heap for improper use of memory, buffer overrun, and memory leaks. See Section "[8.1.2 Heap Memory Checks \(heapchk Function\)](#)" for details.

-Nquickdbg=heapchk

The heap is checked. If an error is detected, a diagnostic message is output.

If allocation and release of a heap memory exist in different compilation units, the **-Nquickdbg=heapchk** option must be specified for both compilation units.

-Nquickdbg=noheapchk

The heap is not checked.

-Nquickdbg={inf_detail|inf_simple}

Specifies the information to be included in diagnostic messages output when errors are detected. See Section "[8.1 Functions for Debugging](#)" for details.

-Nquickdbg=inf_detail

In addition to the message and line number where the error occurred, information to help identify the cause is output, such as the variable name where the error occurred.

-Nquickdbg=inf_simple

The message indicating the error and the line number where the error occurred is output in diagnostic messages.

-N{reordered_variable_stack|noreordered_variable_stack}

The **-Nreordered_variable_stack** option directs the compiler the order in which to allocate the automatic variables to the stack area.

If the **-Nreordered_variable_stack** option is set, the allocation order of automatic variables is determined in the following order.

1. Descending order of their alignments
2. Descending order of data sizes if the alignments are equal
3. The order of appearance of the declaration statements in the source program if both the alignments and data sizes are equal

The stack area of the entire program can be reduced by allocating the automatic variables in descending order of their alignments.

When the **-Nnoreordered_variable_stack** option is specified, automatic variables are allocated in order of the appearance of the declaration statements in the source program. **-Nnoreordered_variable_stack** is set by default.

The order of allocation is not guaranteed when the -Nnoline, -g0, or -Kocl option is set.

For details, see Section "E.4 Data Allocation to Stack Region".

-N{rt_tune|rt_notune}

-Nrt_tune specifies to output the runtime information. The -Nrt_notune option invalidates the -Nrt_tune option. -Nrt_notune is set by default.

The runtime information can be used for the tuning of a user program. Refer to the "[Appendix H Runtime Information Output Function](#)" for details.

-Nrt_tune_func

In addition to the -Nrt_tune output, runtime information about user-defined functions is output.

If the -Nrt_tune_func option is set, the -Nrt_tune option also takes effect. When the -Nrt_notune option is specified after this option, runtime information is not output.

Refer to the "[Appendix H Runtime Information Output Function](#)" for details.

-Nrt_tune_loop[={all|innermost}]

In addition to the -Nrt_tune output, runtime information about loops is output. If the argument ={all|innermost} is omitted, -Nrt_tune_loop=all is assumed.

If the -Nrt_tune_loop option is set, the -Nrt_tune option also takes effect. When the -Nrt_notune option is specified after this option, runtime information is not output.

Refer to the "[Appendix H Runtime Information Output Function](#)" for details.

-Nrt_tune_loop=all

Runtime information about all loops is output.

-Nrt_tune_loop=innermost

Runtime information about the innermost loops is output.

-N{setvalue[=setarg]|nosetvalue}

-Nsetvalue specifies to automatically fill up data allocated in heap or stack area with 0 value. -Nnosetvalue is set by default.

Note that execution time may increase by the -Nsetvalue option.

For *setarg*, specify one of the following:

{ {heap|noheap} | {stack|nostack} | {scalar|noscalar} | {array|noarray} | {struct|nostruct} }

If the argument =*setarg* is omitted, -Nsetvalue=heap, setvalue=stack is assumed.

The -Nsetvalue option is invalidated if the -Nquickdbg option is set.

-Nsetvalue={heap|noheap}

-Nsetvalue=heap specifies to automatically fill up data allocated in heap area with 0 value. -Nsetvalue=noheap is set by default.

-Nsetvalue=heap fills up following area with 0 value:

- areas allocated by malloc function

-Nsetvalue={stack|nostack}

-Nsetvalue=stack specifies to automatically fill up uninitialized variables allocated in stack area with 0 value. -Nsetvalue=nostack is set by default.

-Nsetvalue=stack fills up following variables with 0 value:

- automatic variables

However, variables specified by private clause or lastprivate clause of OpenMP specification are not filled up with 0 value, even if the -Nsetvalue=stack option is effective.

The -Nsetvalue=stack option is equivalent to the following:

-Nsetvalue=stack, setvalue=scalar, setvalue=array, setvalue=struct

-Nsetvalue={scalar|noscalar}

-Nsetvalue=scalar specifies to automatically fill up uninitialized scalar type variables allocated in stack area with 0 value. -Nsetvalue=noscalar is set by default.

-Nsetvalue=scalar option requires that the -Nsetvalue=stack option is set.

-Nsetvalue={array|noarray}

-Nsetvalue=array specifies to automatically fill up uninitialized array type variables allocated in stack area with 0 value. -Nsetvalue=noarray is set by default.

Note that variable-length array in C99 is excluded from the target of the -Nsetvalue=array option.

-Nsetvalue=array option requires that the -Nsetvalue=stack option is set.

-Nsetvalue={struct|nostruct}

-Nsetvalue=struct specifies to automatically fill up uninitialized struct and union type variables allocated in stack area with 0 value.

-Nsetvalue=nostruct is set by default.

-Nsetvalue=struct option requires that the -Nsetvalue=stack option is set.



Example

How to specify -Nsetvalue option

To fill up only uninitialized scalar type variables with 0 value allocated in stack area, specify as follows:

```
-Nsetvalue=stack,setvalue=noarray,setvalue=nostruct
```

-Nsrc

Specifies to output source list to the standard output.

The source list includes annotation of optimization which shows the situation of automatic parallelization, inline expansion, loop unrolling, etc.

Note that when the -Nlst or -Nlst_out=*file* option is specified, the source list is output to the *file*.

For details on the output format on statements specified with the OpenMP C directives, see "[Chapter 4 Multiprocessing](#)".

-Nsta

Specifies to output statistics list to the standard output.

Note that when the -Nlst or -Nlst_out=*file* option is specified, the statistics list is output to the *file*.

2.2.3 Notes for Using Compiler Options

This section provides notes on compiler options.

"-K" option and "-N" option

Multiple options can be specified after the "-K" option using commas to separate them, or they can be specified in two or more separate "-K" options. "-N" option works in the same way.

In the following example, although the options are specified differently, the two commands are equivalent.

Example:

```
$ fccpx -o pgm.out -Kfast -Kparallel -Nsrc -Nsta pgm.c
```

```
$ fccpx -o pgm.out -Kfast,parallel -Nsrc,sta pgm.c
```

Ban of Blank Space

Options which are specified after the "-K" and "-N" options cannot have any blank spaces.

In the following example, the "parallel" option is invalid because there is a blank space in front of the option.

Example:

Invalid Specification:

```
$ fccpx -Kfast, parallel pgm.c
```

Correct Specification:

```
$ fccpx -Kfast,parallel pgm.c
```

2.3 Environment Variable for Compile Command

This section explains the environment variables that the compile command recognizes.

`fccpx_ENV`

`fcc_ENV`

Environment variables to set compiler options. These are valid in Trad/Clang Mode.

The environment variable `fccpx_ENV` is for the cross compiler and the environment variable `fcc_ENV` is for the native compiler.

The user can specify Trad/Clang Mode common compiler options for the value of these environment variables. Refer to "[9.1.2.3.1 Correspondences of Compile Options in Clang Mode and Trad Mode](#)" for a description of the common compiler options for Trad/Clang Mode.

Compiler options specified to these environment variables are valid in Trad/Clang Mode. It is useful to set compiler options that you always specify for these environment variables.

Refer to "[2.4 Compilation Profile File](#)" for the priority of compiler options.



Example

The following two examples are equivalent:

Example 1: Specifying compiler options to environment variable `fccpx_ENV`

```
$ fccpx_ENV="-Kfast -I/usr/prv/include"
$ export fccpx_ENV
$ fccpx a.c
```

Example 2: Specifying compiler options to operands of the compile command

```
$ fccpx -Kfast -I/usr/prv/include a.c
```

`fccpx_trad_ENV`

`fcc_trad_ENV`

Environment variables to set compiler options. These are valid only in Trad Mode.

The environment variable `fccpx_trad_ENV` is for the cross compiler and the environment variable `fcc_trad_ENV` is for the native compiler.

The user can specify Trad Mode specific and Trad/Clang Mode common compiler options for the value of these environment variables. Refer to "[9.1.2.3.1 Correspondences of Compile Options in Clang Mode and Trad Mode](#)" for a description of the common compiler options for Trad/Clang Mode.

Compiler options specified to these environment variables are valid only in Trad Mode.

Refer to "[2.4 Compilation Profile File](#)" for the priority of compiler options.

`FCOMP_LINK_FJOB`

In this system, original objects of this system are usually combined at linking, but they are not combined under the following conditions.

- The `-L` option which is passed to the linker by this system is directly specified on the compile command to pass the linker by user.

This can result in a link error (undefined reference to), which can be avoided by setting the environment variable FCOMP_LINK_FJOB.

These are valid in Trad/Clang Mode. The environment variable FCOMP_LINK_FJOB can have any value. When the environment variable FCOMP_LINK_FJOB is set, original objects of this system are combined at linking.

For the example of a link error, see "C.2.10 Link Error (undefined reference to)".

Example

Example of using FCOMP_LINK_FJOB environment variable:

```
$ export FCOMP_LINK_FJOB=true
```

FCOMP_UNRECOGNIZED_OPTION

The behavior for unrecognized compiler options can be changed by specifying the environment variable FCOMP_UNRECOGNIZED_OPTION. This is valid only in Trad Mode.

Specify either warning or error to the environment variable FCOMP_UNRECOGNIZED_OPTION. If the environment variable FCOMP_UNRECOGNIZED_OPTION is not set or an invalid value is set to it, warning is set.

Value	Description
warning	A warning message is output for an unrecognized compiler option and the compilation is continued.
error	An error message is output for an unrecognized compiler option and the compilation is canceled.

Example

Example 1: Specifying warning to the environment variable FCOMP_UNRECOGNIZED_OPTION

```
$ FCOMP_UNRECOGNIZED_OPTION=warning
$ export FCOMP_UNRECOGNIZED_OPTION
$ fccpx -unrecognized_option a.c
fccpx: warning: -unrecognized_option is unrecognized option.
$ echo $?
0
$
```

Example 2: Specifying error to the environment variable FCOMP_UNRECOGNIZED_OPTION

```
$ FCOMP_UNRECOGNIZED_OPTION=error
$ export FCOMP_UNRECOGNIZED_OPTION
$ fccpx -unrecognized_option a.c
fccpx: error: -unrecognized_option is unrecognized option.
$ echo $?
1
$
```

CPATH

C_INCLUDE_PATH

The directory that retrieves headers with names that do not start with '/' can be added by specifying a value for the environment variable CPATH or C_INCLUDE_PATH. These are valid in Trad/Clang Mode.

To specify directory and other directories, separate the arguments by a colon (:).

For details about the order of the search of the header, see the -I option in Section "2.2.2.1 General Options for Compiler".



Example

Example of using CPATH and C_INCLUDE_PATH environment variables:

```
$ CPATH="/cpath_inc:/cpath_inc_2"
$ export CPATH
$ C_INCLUDE_PATH="/c_include_path:/c_include_path_2"
$ export C_INCLUDE_PATH
```

LIBRARY_PATH

The directory to be searched for libraries at linking can be added by specifying a value for the environment variable `LIBRARY_PATH`. This is valid in Trad/Clang Mode.

Multiple directories can be specified using colons to separate them.

Libraries are searched in the following order:

1. Directories specified as the argument of the `-L` option
2. The directory installed libraries provided by the compiler
3. The directory installed standard libraries
4. Directories specified to the environment variable `LIBRARY_PATH`



Example

Example of using `LIBRARY_PATH` environment variable:

```
$ LIBRARY_PATH="/usr/local/lib64:/usr/local_2/lib64"
$ export LIBRARY_PATH
```

TMPDIR

The temporary directory used by the compile command can be changed by specifying a value for the environment variable `TMPDIR`. This is valid in Trad/Clang Mode.

If the environment variable `TMPDIR` is not set, `/tmp` is used. To avoid output to a common directory, set the local directory to the environment variable `TMPDIR`.



Example

Example of using `TMPDIR` variable:

```
$ TMPDIR=/usr/local/tmp
$ export TMPDIR
```

2.4 Compilation Profile File

The default values of compiler options can be changed by specifying the compilation profile file.

The following compilation profile files are available:

Table 2.4 Compilation Profile files (Trad Mode)

Kind	Mode	File Name
Cross compiler	Trad/Clang Mode common	/etc/opt/FJSVxtclanga/fccpx_PROF
	Trad Mode specific	/etc/opt/FJSVxtclanga/fccpx_trad_PROF

Kind	Mode	File Name
Native compiler	Trad/Clang Mode common	/etc/opt/FJSVxtclanga/fcc_PROF
	Trad Mode specific	/etc/opt/FJSVxtclanga/fcc_trad_PROF

Common compilation profile files for Trad/Clang Mode can specify common compiler options for Trad/Clang Mode. Refer to "[9.1.2.3.1 Correspondences of Compile Options in Clang Mode and Trad Mode](#)" for a description of the common compiler options for Trad/Clang Mode.

Trad Mode specific compilation profile files can specify Trad Mode specific and Trad/Clang common compiler options.



Note

For contents about the settings in compilation profile files, contact the system administrator.

The format of a compilation profile file is as follows:

- Blank characters not enclosed in quotation marks (") or single quotation marks (') are not significant.
- Either single or double quotation marks can be used to begin and end character strings. A character string is terminated by the same delimiter used to begin the string. A character string is also terminated by the end of a line.
- A symbol (#) that is not part of a character string starts a comment. Any characters from the # to the end of the line are treated as part of the comment.



Example

Example of a compilation profile file specification

```
#Default options
-w -O
```

Compiler options are determined by the following priorities:

1. Operand of the compile command
2. Environment variable to set compiler options (mode specific)
3. Environment variable to set compiler options (mode common)
4. Compilation profile file (mode specific)
5. Compilation profile file (mode common)
6. Default value

2.5 Procedure of Execution

This section explains the procedure of executing C language programs.

2.5.1 Environment Variable for Execution

The environment variables shown in "[Table 2.5 Environment Variables for Execution](#)" control the behavior of execution.

For details about OpenMP environment variables, see Section "[4.3.2.2 Environment Variable for OpenMP Specifications](#)".

Table 2.5 Environment Variables for Execution

Environment Variable Name	Operands	Description
FLIB_C_MESSAGE	NO_MESSAGE	A message is output to standard error when an error occurs during execution of the user program. However, if this environment variable is set, an error message is not output.

Environment Variable Name	Operands	Description
		<p>For details of the error message, refer to "Fortran/C/C++ Runtime Messages".</p> <p>When a program was compiled and linked with the -NRtrap option, this environment variable is invalidated.</p> <p> Note</p> <p>.....</p> <p>This environment variable is invalidated for the error message of LLVM OpenMP Library. See "4.4 Runtime Messages" for the message of LLVM OpenMP Library.</p> <p>.....</p>
FLIB_EXCEPT	u	If the -NRtrap option is set at compilation and linking, the floating-point underflow exception (jwe0012i-u) is caught.
FLIB_HEAPCHK_VALUE	hex	Changes the check values used in checks for buffer overrun. See Section "8.1.2.2 Buffer Overrun Check" for details.
FLIB_HOOK_TIME	time	The user-defined function is called at interval of <i>time</i> millisecond(s). <i>time</i> can be a value between 0 and 2147483647. If 0 is specified for <i>time</i> , calling at regular interval is disabled. See Section "8.3 Hook Function", for information about the hook function.
FLIB_RTINFO	(none)	Outputs runtime information. For details, see "Appendix H Runtime Information Output Function".
FLIB_RTINFO_CSV	file	In the Runtime Information Output Function, outputs the fetched information to a CSV file. <i>file</i> is optional. Any file name can be specified as <i>file</i> of the environment variable. If <i>file</i> is omitted, flib_rtinfo.csv is assumed. For details, see "Appendix H Runtime Information Output Function".
FLIB_RTINFO_NOFUNC	(none)	In the Runtime Information Output Function, even if the compiler option -Nrt_tune_func is effective, the output of information for each function is suppressed. For details, see "Appendix H Runtime Information Output Function".
FLIB_RTINFO_NOLOOP	(none)	In the Runtime Information Output Function, even if the compiler option -Nrt_tune_loop is effective, the output of information for each loop is suppressed. For details, see "Appendix H Runtime Information Output Function".
FLIB_TRACEBACK_MEM_SIZE	size	<p>This environment variable changes the size of the heap memory used to debug information output to the trace back map information.</p> <p>An integer value from 1 to 128 can be specified as <i>size</i>. The unit for specifying <i>size</i> is MiB.</p> <p>If no environment variable is specified, or if the value of <i>size</i> is invalid, the default value of this system will be used as the value of <i>size</i>. The default value is 16 MiB.</p> <p>See "5.2.2 Trace Back Information" for the trace back map.</p>

2.5.2 Notes for Execution

When executing programs created on this system, note the following.

2.5.2.1 Variable Allocation at Execution

Programs created by this system allocate the local and private variables in each function to the stack region. When much more space is required for local and private variables in the function, the stack area needs to be extended to the appropriate size.

The stack area of the process can be set by `ulimit` (bash built-in command), etc.

Chapter 3 Optimization

This chapter describes the optimization functions and how to use them.

3.1 Overview of Optimization

Optimization aims to generate object modules (instructions and data areas) that allow the program to execute as quickly as possible. The compiler optimization functions perform the following tasks while generating object modules from C source code.

Deletion

If there are statements or expressions that give the same result, any redundant portions are deleted. Statements or expressions that would be meaningless if executed are also deleted.

Movement

Statements or expressions within loops that give the same result no matter how many times they are repeated are moved outside the loop. When the ultimate result is known without repeated execution, only the required portions are left inside the loop, with the remaining portion moved outside the loop.

Modification

The operator of an arithmetic expression may be changed within a range in which no calculation errors occur. In addition, the data subject to operation may also be modified. For example, for a variable to which a constant value is assigned once and its value remains thereafter unchanged, the reference to that variable is changed to a reference to the constant.

Expansion

When standard library functions or user-defined functions are called, whenever possible, the body of the function is expanded in the location where it was called. This obviates the need to pass arguments and function values or use instructions for calling and return. Moreover, the function's code is optimized as a single unit with the calling function, decreasing execution time.

Execution

If the result of the execution of a statement or the calculated result of an expression is known at compilation time, the instructions that execute the statement or calculate the expression are omitted and only the result of the execution or calculation is used at compilation time.

Other

The functions and characteristics of hardware (for example type of instructions, type and number of registers, addressing methods) are employed as much as possible in order to generate instructions and data areas for fast execution.

Optimization functions can shorten execution time. However, compilation time and work area increase. In addition, depending on the type of optimization, the object modules may be greatly enlarged, and some types of optimization may cause unexpected execution results. To perform optimization, select the appropriate compiler options.

Standard optimization specifies the optimization level and extended optimization supplements standard optimization. These optimization levels are applied by specifying compilation options.



Point

By default, optimization does not cause unexpected results. This is because optimizations with the potential to affect the execution result must be specified by the user via the appropriate optimization option, and are not executed by default.

3.2 Standard Optimization

When an optimization level between level 1 (-O1 option) and level 3 (-O3 option) is specified, standard optimization is performed. This section describes standard optimization.

3.2.1 Elimination of Common Expressions

If two expressions that give equal calculation results (common expressions) are present, the result of the former expression can be used in the latter expression without calculation.

Elimination of common expressions applies to arithmetic operations, relational operations, logical operations, and some standard library functions.

Elimination of common expressions is performed in optimization level 1 (-O1 option) or higher.

An example of elimination of common expressions is given below.

Example: Elimination of Common Expressions

<pre>a = x * y + c; ... b = x * y + d;</pre>	->	<pre>t = x * y; a = t + c; ... b = t + d;</pre>
--	----	---

The $x*y$ portion of the expression on the right hand side of the assignment is a common expression. The code is changed so that the second calculation is eliminated and the result of the first calculation t is used instead. t is a variable generated by the compiler.

3.2.2 Movement of Invariant Expressions

Expressions with values that do not change within a loop (invariant expressions) are moved outside the loop.

Movement of invariant expressions applies to arithmetic operations, relational operations, logical operations, and some standard library functions.

Movement of invariant expressions is performed in optimization level 1 (-O1 option) or higher.

An example of the movement of invariant expressions is given below.

Example: Movement of Invariant Expressions

<pre>for(i = 0; i < n; i++) { y = a[j] * 2; x = x + y * z; }</pre>	->	<pre>y = a[j] * 2; t = y * z; for(i = 0; i < n; i++) { x = x + t; }</pre>
---	----	--

The entire statement $y=a[j]*2$ and the $y*z$ portion of the expression are moved outside the loop. t is a variable generated by the compiler.

The objects that are usually moved by this optimization are statements or parts of statements that are always executed within the loop. However, if the -Kpreex option is specified, invariant expressions in statements that are selectively executed within the loop depending on a decision statement are also moved. To differentiate this optimization from the normal movement of invariant expressions, it is called advance evaluation of invariant expressions. This optimization can further reduce execution time. However, side effects may occur due to the movement. For details of the side effects, see Section "[3.3.1.1 Advance Evaluation of Invariant Expressions](#)".

3.2.3 Reducing Strength of Operators

The "strength" of operators describes the relative amount of execution time required for operation. A "strong" operator requires a large amount of execution time. Typically, addition and subtraction have the same strength. Multiplication is stronger than addition or subtraction and division is even stronger than multiplication. In addition, type conversion between the integer and float types is stronger than addition or subtraction but weaker than multiplication. You can reduce execution time by changing strong operators to weaker ones; for example, by changing multiplication to addition or subtraction.

Reducing the strength of operators is performed in optimization level 1 (-O1 option) or higher.

Induction Variable

This optimization applies to variables of type integer with values that change regularly and incrementally within a loop. This type of variable is called an induction variable.

An example of reducing strength is given below.

Example 1: Reducing the Strength of Operators (Reducing Multiplication to Addition)

```
for(i = 1; i < 10; i++) {  
    ...  
    j = k + i * 100;  
    ...  
}
```

->

```
t = 100;  
for(i = 1; i < 10; i++) {  
    ...  
    j = k + t;  
    ...  
    t = t + 100;  
}
```

Optimization is performed on the operation $i * 100$ on the derivative variable i , so the operation $i * 100$ within the loop is converted to the addition $t = t + 100$. t is a variable generated by the compiler.

Example 2: Reducing the Strength of Operators (Reducing Type Conversion to Addition)

```
for(i = 1; i < 10; i++) {  
    ...  
    x = (float)i;  
    ...  
}
```

->

```
t = 1.0;  
for(i = 1; i < 10; i++) {  
    ...  
    x = t;  
    ...  
    t = t + 1.0;  
}
```

Optimization is performed on the operation $(\text{float})i$ on the derivative variable i , so the operation of type conversion from integer to float is converted to the addition of float type numbers $t = t + 1.0$. t is a `float` type variable generated by the compiler.

3.2.4 Loop Unrolling

Loop unrolling is a type of optimization in which all of the statements executed within a loop are expanded N times within the loop (where N is called the "multiplicity"), and instead, the number of iterations of the loop is reduced to $1/N$. However, if the `Ksimd[={ 1|2|auto}]` option is set and the loop has been optimized using SIMD Extended instructions, the executable statement is unrolled "`SIMD Width`"* N , and the loop iteration count is reduced to "`SIMD Width`"* N .

This optimization is performed on loops which contain no jump from the inside to the outside and no jump from the outside to the inside.

The multiplicity N is an optimal value determined by the compiler depending on the number of iterations of the loop, the number and type of statements executed within the loop and the data types used. In addition, the multiplicity can be specified in the source code with the optimization control line (OCL). For details on this function, see Section "3.4.1 Using Optimization Control Line (OCL)".

Since the number of iterations is reduced and the executed statements that are expanded are optimized as a group within the loop, a faster object is obtained. However, since the statements executed within the loops are expanded, the size of the object module increases.

Loop unrolling works before and after SIMD.

- To apply SIMD or the software pipelining on the outer loop, the inner loop is fully unrolled before SIMD when the iteration count of the inner loop is small. This is called full unrolling before SIMD.
- The inner loop is unrolled or fully unrolled to promote the optimization such as common equations after SIMD.

Loop unrolling is performed in optimization level 2 (-O2 option) or higher.

Loop unrolling works before and after SIMD when `-Kunroll[=n]` option or the unroll specifier is the effective loop.

Loop unrolling can be suppressed by specifying `-Knounroll` option or the nounroll specifier.

You can control the behavior of full unrolling before SIMD by specifying the `fullunroll_pre_simd` or `nofullunroll_pre_simd` specifier. If the `fullunroll_pre_simd` or `nofullunroll_pre_simd` specifier is specified for the loop that has the `-Kunroll[=n]` option, `-Knounroll` option, `unroll` specifier, or `nounroll` specifier enabled, the `fullunroll_pre_simd` or `nofullunroll_pre_simd` specifier takes precedence in full unrolling before SIMD.

Moreover, the message to show the optimized loop and the multiplicity can be output by specifying the `-Koptmsg=2` option.

An example of loop unrolling is given below.

Example: Loop unrolling

```
int i, j, k;
float a[400][400], b[400][400], c[400][400];
for(i = 0; i < 400; i++) {
    for(j = 0; j < 400; j++) {
        c[i][j] = 0.0;
        for(k = 0; k < 400; k++) {
            c[i][j] += a[i][k] * b[k][j];
        }
    }
}
```

For example, if the multiplicity is 4, the innermost loop is converted to the following format in the process of generating the object:

```
for(k = 0; k < 400; k += 4) {
    c[i][j] += a[i][k] * b[k][j];
    c[i][j] += a[i][k+1] * b[k+1][j];
    c[i][j] += a[i][k+2] * b[k+2][j];
    c[i][j] += a[i][k+3] * b[k+3][j];
}
```

3.2.5 Loop Blocking

Loop blocking is the optimization to subdivide the access for array by multiplexing the loop in block size. As a result, the localization of data access is improved, and then efficient use of the cache is promoted.

This optimization is performed on loops which contain no jump from the inside to the outside and no jump from the outside to the inside.

Loop blocking is performed in optimization level 2 (`-O2` option). It can be suppressed by specifying the `-Kloop_noblocking` option.

The message to show the optimized loop can be output by specifying the `-Koptmsg=2` option.

An example of loop blocking is given below.

Example: Loop blocking

```
#define N 200
int a_array[N][N];
int b_array[N][N];

void sub() {
    int i, j;
    for(i = 0; i < N; i++) {
        for(j = 0; j < N; j++) {
            a_array[i][j] = b_array[j][i];
        }
    }
}
```

->

```
#define MIN(x,y) ((x<y)?x:y)
#define BLOCK 2
#define N 200
int a_array[N][N];
int b_array[N][N];

void sub() {
    int i, j, ii, jj;
    for(ii = 0; ii < N; ii += BLOCK) {
        for(jj = 0; jj < N; jj += BLOCK) {
            for(i = ii; i < MIN(N, (ii+BLOCK)); i++) {
                for(j = jj; j < MIN(N, (jj+BLOCK)); j++) {
                    a_array[i][j] = b_array[j][i];
                }
            }
        }
    }
}
```

3.2.6 Software Pipelining

Software pipelining schedules instructions in loops to execute as parallel as possible.

Software pipelining can be suppressed by specifying the compile-time option `-Knoswp`.

Software pipelining performs an instruction scheduling to arrange instructions of a particular iteration in a loop with instructions of the following iterations, and reshapes the loop. Thus software pipelining requires enough loop iteration count. Other optimizations affect the required iteration count because it depends on instructions in the loop.

If the original loop is short, software pipelining may be less effective because instruction scheduling is performed moderately to keep required iteration count small.

When software pipelining is applied to a loop whose iteration count is variable, as shown in the following example, the compiler inserts a branch instruction to choose the software-pipelined loop or the original loop in case the iteration count is short. It is decided at run time whether the software-pipelined loop is chosen. The size of the object program will increase.

Example:

```
void sub (int N) {  
    (The original loop: the iteration count is N.)  
}
```

[Optimized pseudo-code]

```
void sub (int N) {  
    if (Is N enough?) {  
        (The software-pipelined loop)  
    } else {  
        (The original loop: the iteration count is N.)  
    }  
}
```

If the compile-time option `-Koptmsg=2` is specified, optimization messages about results of software pipelining are output with the line numbers of loops. When software pipelining is applied to a loop, optimization messages `jwd8204o-i` and `jwd8205o-i` are output at once. When software pipelining is not applied to a loop, one of optimization messages from `jwd8662o-i` to `jwd8674o-i` is output. Optimization messages about software pipelining are not output when the `-Knoswp` option is specified, or when a loop disappears because of other optimizations before software pipelining, such as full unrolling.

The optimization message `jwd8205o-i` shows the minimum iteration count required to choose the software-pipelined loop. Make sure to consider following optimizations applied to the same loop because the target loop of software pipelining is the innermost loop; when loop collapse, loop interchange, thread parallelization, or inline expansion is applied, the iteration count of the target loop of software pipelining is changed as shown in the following table. If this is the case, the value shown in the optimization message `jwd8205o-i` should be compared with the iteration count in the table.

Optimization applied to the same loop	Message number	The iteration count of the target loop to which software pipelining is applied
Loop collapse	<code>jwd8330o-i</code>	Product of the iteration counts of the collapsed loops.
Loop interchange	<code>jwd8211o-i</code>	The iteration count of the innermost loop after loop interchange.
Thread parallelization	<code>jwd5001p-i</code> , etc.	Quotient of the iteration count divided by the number of threads.
Inline expansion	<code>jwd8101o-i</code>	The iteration count of the innermost loop after inline expansion.

As for the iteration count shown by optimization message `jwd8205o-i`, the value is output with taking account of the iteration counts executed at the same time by SIMD. Note that when the `-Ksimd_reg_size=agnostic` option is effective, the value is output as if the SVE vector register size is 128 bits is output, because the SVE vector register size is unknown at compilation.

3.2.7 SIMD

3.2.7.1 Normal SIMD

The compiler may apply optimizations using SIMD extensions. Using SIMD extensions, multiple operations of the same kind are executed at once.

This optimization is performed on loops which contain no jump from the inside to the outside and no jump from the outside to the inside.

This optimization is enabled by the -O2 option or higher. And this optimization is suppressed by the -Knosimd option. Moreover, the message to show the optimized loop can be output by specifying -Koptmsg=2 option.

3.2.7.2 SIMD Extension for Loop Containing "if" Statement

When -Ksimd={2|auto} option is specified, SIMD extension can be applied for the loop that contains "if" statement. The conditional execution instructions are only store instruction and move instruction in CPU architecture, and other instructions are executed speculatively (statements in the "if" statement are executed even if the condition of the "if" statement is false). Therefore the true rate of condition of the "if" statement is high, the performance may be improved. However, side effects, such as exceptions at execution, may occur in the execution results because instructions are executed speculatively.

3.2.7.3 List Vector Conversion

When the true rate of the "if" statement is low, and the "if" statement in loop contains many instructions, the performance may be improved by applying list vector conversion.

The list vector conversion generates the following two loops:

1. The loop to save the value of loop control variable into a new array when the condition of the original "if" statement is true.
2. The loop to execute the statements of the original "if" statement, and its loop iteration count is equal to the number of the new array element.

The loop of 2. becomes list access method, but the loop becomes the target of SIMD extension and software pipelining.

Example of list vector conversion is shown in the following. The list vector conversion is applied by specifying the optimization control line. For details about `simd_listv` specifier, see Section "3.4.1 Using Optimization Control Line (OCL)".

Example:

```
for (i=0; i<n; ++i) {  
#pragma statement simd_listv  
    if (m[i]) {  
        a[i] = b[i] + c[i];  
    }  
}
```

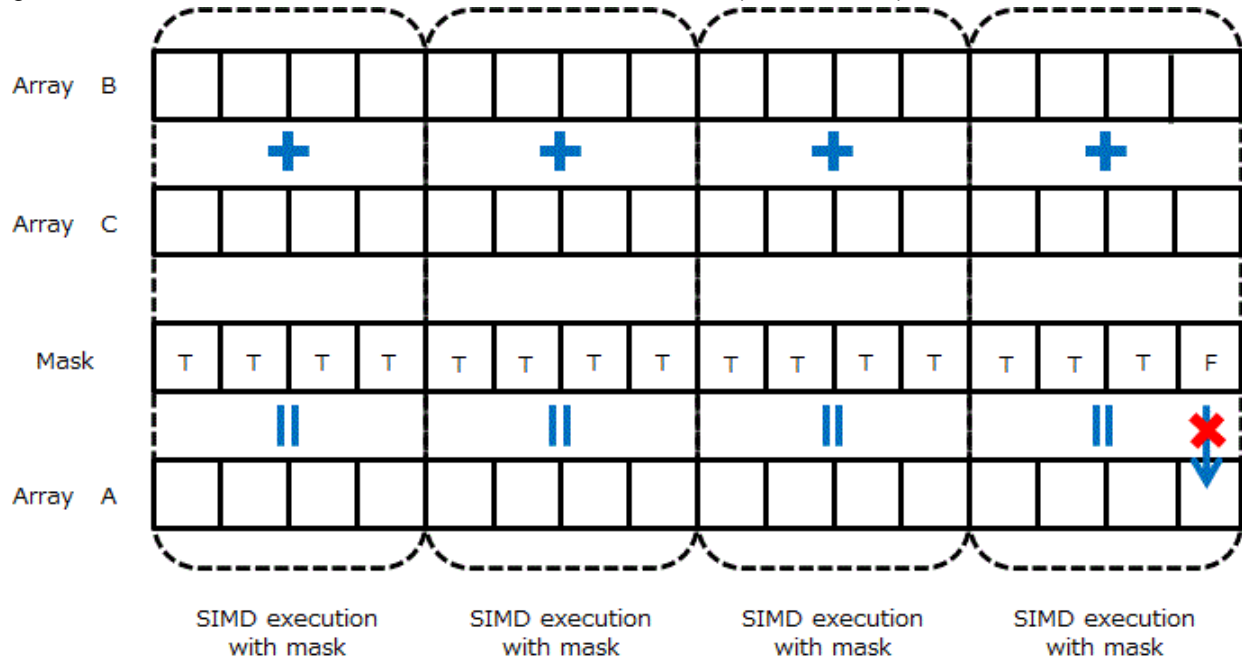
->

```
j = 0;  
for (i=0; i<n; ++i) {  
    if (m[i]) {  
        idx[j] = i;  
        j = j + 1;  
    }  
}  
for (k=0; k<j-1; ++k) {  
    i = idx[k];  
    a[i] = b[i] + c[i];  
}
```

3.2.7.4 SIMD with Redundant Executions for the SIMD Width

When the loop iteration count is not a multiple of the SIMD width, this function generates loops which use SIMD extensions over the whole iterations by using SIMD instructions with masks. (Refer to the example and figures below.) When SIMD is performed, this function is applied by default. The following figure is the image of this function.

Figure 3.1 SIMD with redundant executions for the SIMD width (4-wide SIMD)



[Explanation]

The whole iterations are executed by using SIMD instructions with masks.

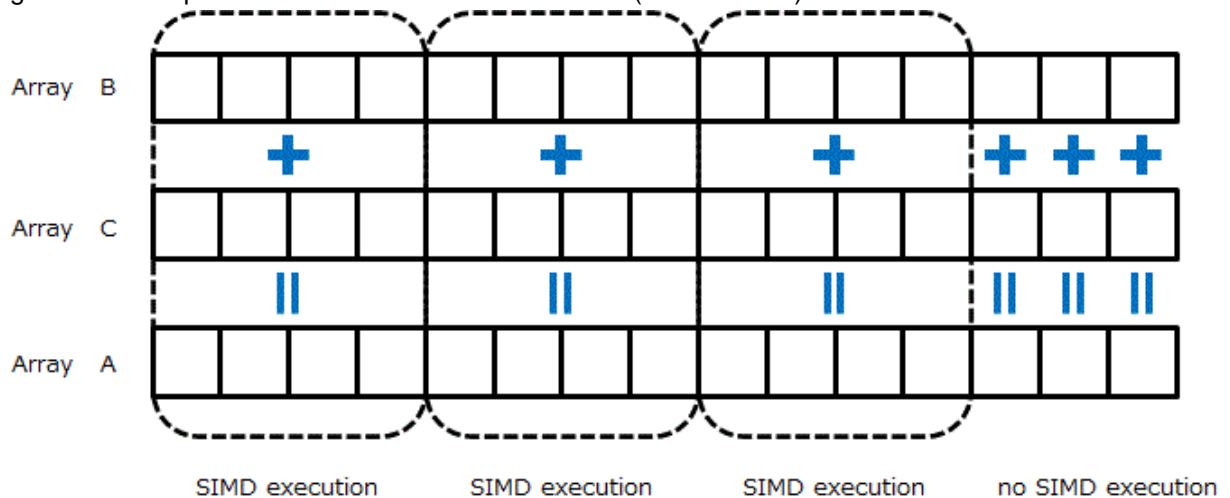
The loops whose remainder dividing the loop iteration count by the SIMD width tend to be 1 or 2 may not obtain the effect of optimization. In order to suppress this function, specify the `simd_noredundant_vl` specifier in the optimization control line.

When the function is not applied to the source program in the example, SIMD instructions are used as shown in "Figure 3.2 The optimization control line is not effective (4-wide SIMD)". The remaining iterations, which correspond the remainder after dividing the loop iteration count by the SIMD width, do not use SIMD instructions.

Example: The optimization control line (OCL) is specified

```
#pragma loop simd_noredundant_vl
for(i=0; i<m; i++) {
    a[i] = b[i]+c[i];
}
```

Figure 3.2 The optimization control line is not effective (4-wide SIMD)



[Explanation]

When the loop control variable `i` is from 13 to 15, SIMD instructions are not used.

This function can be suppressed by specifying -Kextract_stride_store option.

3.2.7.5 Math functions that SIMD Extensions can be applied to

The following shows the math functions that SIMD extensions can be applied to when -Klib option is effective.

```
abs, acos, acosf, acosh, acoshf, asin, asinf, asinh, asinhf, atan, atan2, atan2f, atanf, atanh,
atanhf, cbrt, cbrtf, ceil, ceilf, cimag, cimagf, conj, conjf, copysign, copysignf, cos, cosf, cosh,
coshf, creal, crealf, erf, erfc, erfcf, erff, exp, exp2, exp2f, expf, fabs, fabsf, floor, floorf,
fmax, fmaxf, fmin, fminf, lgamma, lgammaf, log, log10, log10f, log2, log2f, logf, nearbyint,
nearbyintf, pow, powf, rint, rintf, round, roundf, sin, sinf, sinh, sinhf, sqrt, sqrtf, tan, tanf,
tanh, tanhf, tgamma, tgammaf, trunc, truncf
```

See also the description of the -Kilfunc and -Kmfunc options in the section ["2.2 Compiler Options"](#) for information on math functions that SIMD extensions can be applied to.

3.2.8 Loop Unswitching

Loop unswitching is an optimization to modify loop by putting out the "if" statement out of the loop and creating a loop in each "TRUE" and "FALSE" blocks of the "if" statement in order to promote some optimizations such as instruction scheduling if the condition of the "if" statement is invariant in a loop.

This optimization is enabled by specifying the -O3 or -Kparallel option.

This optimization is disabled by specifying the -O2 option or less, and invalidating -Kparallel option.

```
void foo(double a[], int b, int c, int n) {
    int i;
    for(i = 0; i < n; i++) {
        if(b == c) {
            a[i] = 0;
        } else {
            a[i] = 1;
        }
    }
}
```

->

```
void foo(double a[], int b, int c, int n) {
    int i;
    if(b == c) {
        for(i = 0; i < n; i++) {
            a[i] = 0;
        }
    } else {
        for(i = 0; i < n; i++) {
            a[i] = 1;
        }
    }
}
```

3.2.9 Inline Expansion

Using inline expansion, when a function is referenced (called), the body of the function is expanded directly in the place where the function was called. Expansion eliminates the passing of arguments and function values and the overhead for calling and returning from the function (saving and restoring registers, branches). In addition, bringing the expanded portion together with the calling portion promotes other types of optimization. Therefore, execution time can be greatly reduced, but the size of the object module will increase.

Inline expansion is performed in optimization level 3 (-O3 option). It can be suppressed by specifying the -x0 option.

The message to show the name and its location of the expanded function can be output by specifying the -Koptmsg=2 option.

3.3 Extended Optimization

When one of the extended function options is specified regardless of optimization, extended optimization is performed in addition to the standard optimization.

This section describes the functions comprising extended optimization.

Depending on the extended optimization function, the size of the object modules (instructions and data areas) generated by the compiler may be greatly enlarged or side effects may occur in the execution results.

3.3.1 Optimization by Modifying Evaluation Methods

This section describes optimization by advance evaluation of invariant expressions and arithmetic evaluation method modification.

3.3.1.1 Advance Evaluation of Invariant Expressions

Invariant expressions in statements executed selectively within a loop (for example if they depend on an "if" statement or similar) are moved outside the loop. Advance evaluation of invariant expressions can reduce execution time, however may also cause side effects.

The advance evaluation of invariant expressions can be controlled with the -Kpreex and -Knopreex options.

In some cases, advance evaluation of invariant expressions causes the execution of instructions that should not be executed according to the logic of the program, resulting in errors. However, they will not affect the results of execution or their precision.

Moving the execution location may cause an unexpected standard library function call, invalid access to an element of an array, or a division-by-zero error. Examples of the effects that may occur due to advance evaluation of invariant expressions are given below.

Example 1: Referencing of Array Elements

<pre>int a[10], b[10]; ... for(i = 0; i < 10; i++) { if(j < 10) { a[i] = b[j] * f; } }</pre>	->	<pre>int a[10], b[10]; ... t = b[j] * f; for(i = 0; i < 10; i++) { if(j < 10) { a[i] = t; } }</pre>
--	----	---

In the program on the left, when the variable *j* is less than 10, array element *b[j]* is referenced, so the declared range of array *b* is not exceeded.

However, when array element *b[j]* is moved outside the loop due to advance evaluation of invariant expressions, it is referenced regardless of the value of *j*. As a result, if *j* is 10 or greater, the declared range of the array is exceeded when referenced. If the value of *j* is extremely large, an illegal area will be referenced and program execution will be aborted. *t* is a variable of type *int* generated by the compiler.

Example 2: Division

<pre>int a[100], b, f; ... for(i = 0; i < n; i++) { if(f != 0) { a[i] = b / f; } else { a[i] = 0; } }</pre>	->	<pre>int a[100], b, f; ... t = b / f; for(i = 0; i < n; i++) { if(f != 0) { a[i] = t; } else { a[i] = 0; } }</pre>
--	----	---

In the program on the left, the division is performed only when the variable *f* is not 0, so the error by division exception will not occur.

Since the division *b/f* is moved outside the loop due to the advance evaluation of invariant expressions, the division is performed regardless of the value of *f*. As a result, if the variable *f* is 0, program execution will be aborted. *t* is an *int* type variable generated by the compiler.

3.3.1.2 Arithmetic Evaluation Method Modification

The method of evaluating arithmetic expressions is changed to shorten execution time. This modification may cause side effects.

Modification of the Operation Sequence

The operation sequence is modified in order to promote compile-time operations on constants, movement of invariant expressions, instruction scheduling, and other types of optimization.

There may be cases in which calculations that did not give exceptions (overflow or underflow) in the operation sequence prior to the modification will do so as a result of the modification. In addition, in floating-point operations, the precision of the operation results may differ depending on the number of valid digits in each operation term.

Modification of the operation sequence can be controlled with the -Keval and -Knoeval options.

An example of using this optimization is given below.

Example: Modification of the operation sequence

<pre>double a[100], b[100], c[100]; double x, y; ... for(i = 0; i < 100; i++) { a[i] = b[i] * x * c[i] * y; }</pre>	->	<pre>double a[100], b[100], c[100]; double x, y; ... t = x * y; for(i = 0; i < 100; i++) { a[i] = b[i] * c[i] * t; }</pre>
--	----	---

Once the operations among invariant terms are collected, $x*y$ is moved outside the loop as an invariant expression.

Even though the calculation on the left operates as designed and does not generate exceptions, depending on the values of the arrays `b` and `c` and the variables `x` and `y`, the calculation on the right may generate exceptions or the intended result may not be obtained. `t` is a `double` type variable generated by the compiler.

Note that programs that generate exceptions or change the precision of operation results due to modification of the operation sequence are naturally programs that process values near the limit values or perform precision-sensitive calculations. Therefore, even if side effects occur due to the implementation of this optimization, this optimization may not be the basic reason.

Conversion of Division to Multiplication

The division of invariant floating type data within a loop is changed to multiplication. This conversion increases execution speed. Note that this conversion may affect the precision of the execution results.

The conversion of division to multiplication can be controlled with the -Keval and -Knoeval options.

An example of this optimization and its effects is given below.

Example: Conversion of Division to Multiplication

<pre>float a[10], b[10], v; ... for(i = 0; i < 10; i++) { a[i] = b[i] / v; }</pre>	->	<pre>float a[10], b[10], v; ... t = 1.0 / v; for(i = 0; i < 10; i++) { a[i] = b[i] * t; }</pre>
---	----	--

The operation `b[i]/v` within the loop is converted to the multiplication `b[i]*t`. `t` is a variable of type `float` generated by the compiler.

In the program on the right, `1.0/v` is moved outside the loop and changed to multiplication of `t`, so there is potential for calculation errors.

3.3.2 Optimization of Pointers

When pointers are not being optimized, the compiler performs on the safe side assuming that it is not known what the pointer variable is pointing to.

In contrast, when the optimization of pointers is in effect, the compiler performs with the following prerequisites:

- Prerequisite 1

No other pointer variable points to the same area.

- Prerequisite 2

Situations in which an area is accessed with a pointer variable are not mixed with those in which the same area is accessed directly without a pointer.

For example, if the areas accessed by pointer variables are limited to those allocated dynamically with the `malloc` function or similar, prerequisite 2 is satisfied.

When these prerequisites are met, a high degree of optimization can be applied; for example, the elimination of common expressions and the movement of invariant expressions.

However, if optimization is performed on programs that do not satisfy the above prerequisites, the user may not obtain the intended results.

The optimization of pointers can be controlled with the `-Krestp` and `-Knorestp` options.

Example 1: Optimization of Pointers (Explanation of Prerequisite 1)

<pre>a1 = *p + 100; *q = 0; a2 = *p + 100;</pre>	->	<pre>a1 = *p + 100; *q = 0; a2 = a1; /*Changed by elimination of common expressions*/</pre>
--	----	---

Optimization is performed under the premise that the pointer variable `p` and the pointer variable `q` do not point to the same area. As a result, `*p+100` is considered to be a common expression.

If the pointer variable `p` and the pointer variable `q` do point to the same area, unexpected results may occur.

Example 2: Optimization of Pointers (Explanation of Prerequisite 2)

<pre>a1 = *p + b; x = 0; a2 = *p + b;</pre>	->	<pre>a1 = *p + b; x = 0; a2 = a1; /* Changed by elimination of common expressions */</pre>
---	----	--

Optimization is performed under the premise that the pointer variable `p` does not point to any of the variables `a1`, `a2`, `b`, or `x`. As a result, `*p+b` is considered to be a common expression.

If the pointer variable `p` points to the variable `a1` or the variable `x`, unexpected results may occur.

Note that these prerequisites are applied to headers, too.

3.3.3 Multi-Operation Function

A multi-operation function is a function achieving improved execution performance by performing multiple functions of the same type using one call.

When the `-Kmfunc` option is specified or the `mfunc` specifier is effective, the compiler analyzes and replaces a loop with a multi-operation function.

For details about the `-Kmfunc` option and the functions targeted by multi-operation functions, see Section "[2.2.2.5 -K Option](#)".

For details about the `mfunc` specifier, see Section "[3.4.1 Using Optimization Control Line \(OCL\)](#)".

3.3.3.1 About Calling of Multi-Operation Functions

When the compiler cannot analyze complicated loops, the user program can call a multi-operation function directly.

Note the following:

- Include "`fjmfunc.h`" in the program.
- If a user program calls a multi-operation function directly, the argument used to return the result and the memory used for other arguments should be separate. If the memory addresses overlap, execution may be incorrect.
- Specify the `-Kmfunc` option when the program is linked.
- The argument check may be omitted in the multi-operation function for high-speed. Therefore, the user program may be terminated abnormally when the special value (NaN and Inf, etc.) defined by IEEE 754 is input.

"[Table 3.1 List of direct calling multi-operation function](#)" lists the multi-operation functions that can be called directly.

Table 3.1 List of direct calling multi-operation function

Function	Type Argument	Calling Format	Content of Calculation
acos	long n; float x[n],y[n];	void v_acos(x,&n,y);	for(i=0; i<n; i++) y[i] = acosf(x[i]);
	long n; double x[n],y[n];	void v_dacos(x,&n,y);	for(i=0; i<n; i++) y[i] = acos(x[i]);
asin	long n; float x[n],y[n];	void v_asin(x,&n,y);	for(i=0; i<n; i++) y[i] = asinf(x[i]);
	long n; double x[n],y[n];	void v_dasin(x,&n,y);	for(i=0; i<n; i++) y[i] = asin(x[i]);
atan	long n; float x[n],y[n];	void v_atan(x,&n,y);	for(i=0; i<n; i++) y[i] = atanf(x[i]);
	long n; double x[n],y[n];	void v_datan(x,&n,y);	for(i=0; i<n; i++) y[i] = atan(x[i]);
atan2	long n; float x1[n],x2[n],y[n];	void v_atan2(x1,x2,&n,y);	for(i=0; i<n; i++) y[i] = atan2f(x1[i],x2[i]);
	long n; double x1[n],x2[n],y[n];	void v_datan2(x1,x2,&n,y);	for(i=0; i<n; i++) y[i] = atan2(x1[i],x2[i]);
erf	long n; float x[n],y[n];	void v_erf(x,&n,y);	for(i=0; i<n; i++) y[i] = erff(x[i]);
	long n; double x[n],y[n];	void v_derf(x,&n,y);	for(i=0; i<n; i++) y[i] = erf(x[i]);
erfc	long n; float x[n],y[n];	void v_erfc(x,&n,y);	for(i=0; i<n; i++) y[i] = erfcf(x[i]);
	long n; double x[n],y[n];	void v_derfc(x,&n,y);	for(i=0; i<n; i++) y[i] = erfc(x[i]);
exp	long n; float x[n],y[n];	void v_exp(x,&n,y);	for(i=0; i<n; i++) y[i] = expf(x[i]);
	long n; double x[n],y[n];	void v_dexp(x,&n,y);	for(i=0; i<n; i++) y[i] = exp(x[i]);
exp10	long n; float x[n],y[n];	void v_exp10(x,&n,y);	for(i=0; i<n; i++) y[i] = exp10f(x[i]);
	long n; double x[n],y[n];	void v_dexp10(x,&n,y);	for(i=0; i<n; i++) y[i] = exp10(x[i]);
log	long n; float x[n],y[n];	void v_log(x,&n,y);	for(i=0; i<n; i++) y[i] = logf(x[i]);
	long n; double x[n],y[n];	void v_dlog(x,&n,y);	for(i=0; i<n; i++) y[i] = log(x[i]);
log10	long n; float x[n],y[n];	void v_log10(x,&n,y);	for(i=0; i<n; i++) y[i] = log10f(x[i]);
	long n; double x[n],y[n];	void v_dlog10(x,&n,y);	for(i=0; i<n; i++) y[i] = log10(x[i]);

Function	Type Argument	Calling Format	Content of Calculation
sin	long n; float x[n],y[n];	void v_sin(x,&n,y);	for(i=0; i<n; i++) y[i] = sinf(x[i]);
	long n; double x[n],y[n];	void v_dsin(x,&n,y);	for(i=0; i<n; i++) y[i] = sin(x[i]);
cos	long n; float x[n],y[n];	void v_cos(x,&n,y);	for(i=0; i<n; i++) y[i] = cosf(x[i]);
	long n; double x[n],y[n];	void v_dcos(x,&n,y);	for(i=0; i<n; i++) y[i] = cos(x[i]);
sincos	long n; float x[n],y1[n],y2[n];	void v_scn(x,&n,y1,y2);	for(i=0; i<n; i++) sincosf(x[i], &y1[i], &y2[i]);
	long n; double x[n],y1[n],y2[n];	void v_dscn(x,&n,y1,y2);	for(i=0; i<n; i++) sincos(x[i], &y1[i], &y2[i]);
pow	long n; float x1[n],x2[n],y[n];	void v_pow(x1,x2,&n,y);	for(i=0; i<n; i++) y[i] = powf(x1[i], x2[i]);
	long n; float x1[n],x2[n],y[n],a;	void v_powl(x,&a,&n,y);	for(i=0; i<n; i++) y[i] = powf(x[i], a);
	long n; double x1[n],x2[n],y[n];	void v_dpow(x1,x2,&n,y);	for(i=0; i<n; i++) y[i] = pow(x1[i], x2[i]);
	long n; double x1[n],x2[n],y[n],a;	void v_dpowl(x,&a,&n,y);	for(i=0; i<n; i++) y[i] = pow(x[i], a);

The following is an example of calling a multi-operation function.

Example 1:

```
#include <math.h>
#define N 1000
...
double a;
double x[N], y[N], z[N];
int i;
...
for(i = 0; i < N; i++) {
    y[i] = exp(x[i]);
    z[i] = pow(x[i], a);
}
```

Direct calling a multi-operation function

```
#include <fjfunc.h>
#define N 1000
...
long n = N;
double a;
double x[N], y[N], z[N];
...
v_dexp(x, &n, y);
v_dpowl(x, &a, &n, z);
```


Example 2:

When function calling exists in "if" statement and array in function can compress an array only value with true, it can use a multi-operation function.

```
#include <math.h>
#define N 1000
...
    double a, x, y;
    int i;
...
    for(i = 0; i < N; i++) {
        x = ...
        if(x > a) {
            y = y + sin(x);
        }
    }
}
```

Direct calling multi-operation function

```
#include <fjmfnc.h>
#define N 1000
...
    double a, x, y;
    double wx[N], wy[N];
    int i;
    long j;
...
    j = 0;
    for(i = 0; i < N; i++) {
        x = ...
        if(x > a) {
            wx[j] = x;
            j++;
        }
    }
    v_dsin(wx, &j, wy);
    for(i = 0; i < j; i++) {
        y = y + wy[i];
    }
}
```

3.3.3.2 Effects of Compiler Option -Kmfnc=3

When the -Kmfnc=3 option is specified, a multi-operation function is used in the loop which contains an "if" statement. This may cause an abend as shown in the following example.

The following example also illustrates the effects of the -Kmfnc=3 option.

Example:

```
double x[1000], y[1000];
int i;
...
    for(i = 0; i < 2000; i++) {
        if(i < 1000) {
            y[i] = exp(x[i]);
        }
    }
}
```

Explanation:

In the above example, when the condition $i < 1000$ is true, the `exp` function is executed. Since the value of the argument in the `exp` function is less than 1000, the value of subscripts of array `x` and `y` never exceed the declared range.

However, if a multi-operation function is used in the loop, array `x` exceeds the declared range because the array elements for the iteration count of the "for" loop are used as the argument of the `exp` function regardless of the condition of the "if" statement. In this case, a storage protection exception, and an abnormal end may occur.

In this case, do not specify the compiler option `-Kmfunc=3`.

3.3.4 Loop Striping

Using loop striping all of the statements executed within a loop are expanded up to N times within the loop (where N is called the "striping size"). The overhead for loop iteration is reduced, and software pipelining may be promoted.

This optimization is performed on loops which contain no jump from the inside to the outside and no jump from the outside to the inside.

Example: Loop Striping

When the striping size is 4 by specifying `-Kstriping=4`, the loop is expanded as follows:

<pre>for(i = 0; i < N; i++) { a[i] = b[i] + c[i]; }</pre>	<p>-></p>	<pre>for(i = 0; i < N; i += 4) { tmp_b1 = b[i]; tmp_b2 = b[i+1]; tmp_b3 = b[i+2]; tmp_b4 = b[i+3]; tmp_c1 = c[i]; tmp_c2 = c[i+1]; tmp_c3 = c[i+2]; tmp_c4 = c[i+3]; tmp_a1 = tmp_b1 + tmp_c1; tmp_a2 = tmp_b2 + tmp_c2; tmp_a3 = tmp_b3 + tmp_c3; tmp_a4 = tmp_b4 + tmp_c4; a[i] = tmp_a1; a[i+1] = tmp_a2; a[i+2] = tmp_a3; a[i+3] = tmp_a4; }</pre>
--	--------------	---

In loop striping optimization, the statements executed within the loops are expanded. Therefore, the size of the object program increases as loop unrolling. Further, the compile time may increase. Note that the execution performance may decrease because the number of registers used are increased.

Loop striping can be suppressed by specifying the `-Knostriping` option.

Moreover, the message to show the optimized loop and the multiplicity can be output by specifying the `-Koptmsg=2` option.

3.3.5 zfill

The `zfill` optimization speeds up write operations for array data by using an instruction that allocates space on the cache for writing (DC ZVA) without loading data from the memory. The `zfill` is applied to array data to be written in a loop.

However, the `zfill` is not applied to array data which is referred in the same loop, accessed non-sequentially, or stored in an "if" statement. Further, if the `zfill` optimization is applied, prefetch instructions to the second level cache are not generated.

The `zfill` optimization works on the data N blocks ahead of the address pointed to by the target store instruction where one block is 256 byte-long and N is an integer value between 1 and 100. If a value is not specified for N , the compiler will automatically determine a value.

The following optimizations are suppressed because the loop is modified that data is always stored to the cache lines allocated by the `zfill` optimization:

- Loop unrolling
- Loop striping

Further, performance may also be reduced under the following conditions:

- The program is not affected by memory bandwidth bottleneck.

- When the loop iteration count is too few.
- When the block size is explicitly specified with the `-Kzfill=Nooption` and the memory size where the data is written in the loop is smaller than N blocks.

Do not specify the `-Kzfill` option if the execution performance is likely to decrease as a result.

This optimization is not expected to be effective for all the loops in the program, so it is recommended to use the optimization control specifier `zfill` for each loop rather than the `-Kzfill` option for the entire program.

3.3.6 Loop Versioning

The loop versioning generates two loops to which optimization is applied and is not applied. The object program created by the compiler judges the data dependency of array at execution time and selects either loop.

The optimization, such as SIMD, software pipelining or automatic parallelization, is promoted by loop versioning.

The size of the object program and compile time may increase because the loop versioning generates two loops.

The execution performance may decrease because the processing for judgement is overhead.

The loop versioning is applied only to the innermost loop which contains a single array whose data dependency is unknown. The loop versioning may decrease the overhead because of the unknown dependency.

Moreover, the message to show where optimized by the loop versioning can be output by specifying the `-Koptmsg=2` option.

The following example shows the loop versioning.

Example:

```
void f(double *a, double *b, int n) {
    for (int i = 0; i < n; i++) {
        a[i] += b[i];
    }
}
```

Optimized pseudo-code

```
void f(double *a, double *b, int n) {
    if ((&a[n-1] < &b[0]) || (&b[n-1] < &a[0])) {
#pragma loop norecurrence
        for (int i = 0; i < n; i++) { /* optimized loop */
            a[i] += b[i];
        }
    } else {
        for (int i = 0; i < n; i++) { /* no-optimized loop */
            a[i] += b[i];
        }
    }
}
```

It is unknown at compile time that the arrays "a" and "b" are overlapped. When the `-Kloop_versioning` option is specified, the compiler generates two loops with "if" statement for judging the overlap of the arrays at execution time. If the array "a" and "b" are found to have no data dependency, SIMD may be promoted for the loop.

3.3.7 Clone Optimization

Clone optimization generates conditional branches of variables for loops in order to promote other optimizations, such as full unrolling. Clone optimization is applied by specifying the optimization control line. The calculation result is not guaranteed if the optimization control line clone is used incorrectly. See Section ["3.4.1.2 Optimization Control Specifier"](#) and ["3.4.1.3 Notes for Optimization Control Specifiers"](#) for more information about clone.

Example:

```
#pragma loop clone n==10
for (i=0;i<n; ++i) {
    a[i] = i;
}
```

->

```
if (n==10) {
    for (i=0;i<10;++i) {
        a[i] = i;
    }
} else {
    for (i=0;i<n;++i) {
        a[i] = i;
    }
}
```

3.3.8 Unroll-and-Jam

Unroll-and-jam is the optimization to unroll an outer loop of a nested loop N times and jam the unrolled statements into the inner loop as loop fusion. Unroll-and-jam promotes removing common expression and improves the execution performance. The increase of data stream and change of the order of data access may decrease the cache efficiency and the execution performance. This optimization is not applied to innermost loop.

This optimization is not expected to be effective for all the loop in the program, so it is recommended to use the optimization control specifier `unroll_and_jam` or `unroll_and_jam_force` for each loop rather than the `-Kunroll_and_jam` option for the entire program.

For details about `unroll_and_jam` and `unroll_and_jam_force` specifier, see Section "3.4.1 Using Optimization Control Line (OCL)".

To output line numbers and expansion counts of optimized loops, specify `-Koptmsg=2` option.

Example: Unroll-and-jam

```
#pragma loop unroll_and_jam_force 2
for (i=0; i<128; i++) {
    for (j=0; j<128; j++) {
        a[i][j] = b[i][j] + b[i+1][j];
        ...
    }
}
```

->

```
for (i=0; i<128; i+=2) {
    for (j=0; j<128; j++) {
        a[i][j] = b[i][j] + b[i+1][j];
        a[i+1][j] = b[i+1][j] + b[i+2][j];
        ...
    }
}
```

3.3.9 Tree-Height-Reduction Optimization

The tree-height-reduction optimization increases the instruction-level parallelism by changing the order of calculations in a loop to keep the calculation tree as short as possible.

The calculation tree is an expression of arithmetic operations in a form of tree structure in the order of operations. Leaf nodes of the tree structure hold values and other nodes hold operators. Nodes whose operator has higher priority are placed lower in the following example.

This optimization is applied to floating-point operations when the `-Keval_concurrent` option is set. In this case, side effects (calculation errors) may occur in the execution results. See "3.6.1 Side Effect of Optimizations for Floating-Point Operation" for the side effects of optimizations.

An example of optimization is shown below.

Example:

```
for(i = 0; i < n; i++) {
    x[i] = a[i] * b[i] + c[i] * d[i] + e[i] * f[i] + g[i] * h[i];
}
```

Figure 3.3 The calculation tree when the -Keval_noconcurrent option or optimization control specifier eval_noconcurrent is set

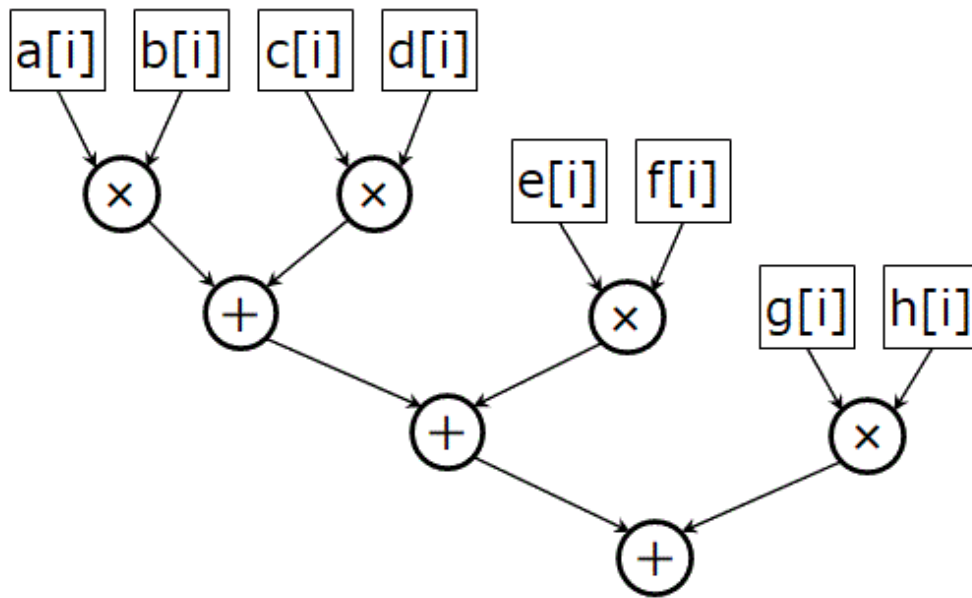
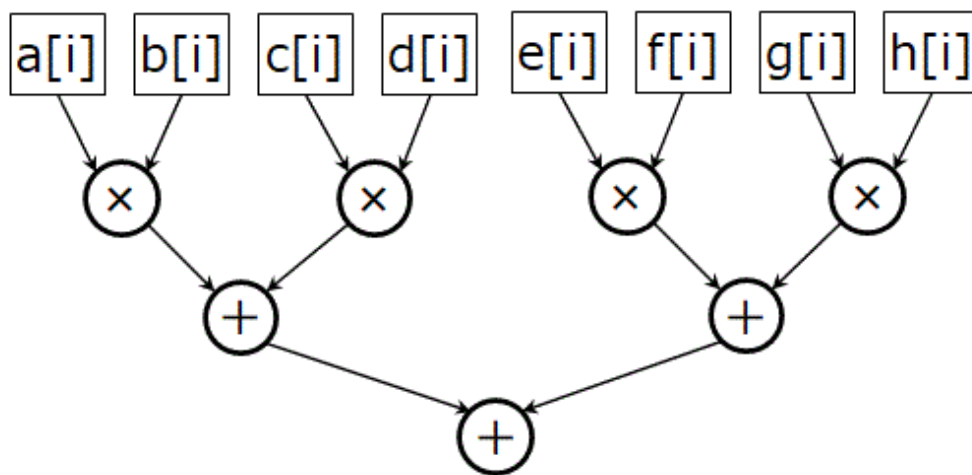


Figure 3.4 The calculation tree when the -Keval_concurrent option or optimization control specifier eval_concurrent is set



3.3.10 Loop Fission

Loop fission is a function to fission a loop into two or more small loops for aiming at the following objectives:

- Promotion of the software pipelining
- Improvement of the cache memory efficiency
- Resolution of register shortage

This optimization is performed on loops which contain no jump from the inside to the outside and no jump from the outside to the inside.

Loop fission is performed on loops to which the loop_fission_target specifier is specified when the -Kloop_fission and -Kocl options are set. It can be suppressed by specifying the -Kloop_nofission option.

The message to show the optimized loop can be output by specifying the -Koptmsg=2 option.

An example of automatic loop fission is given below.

Example: Automatic loop fission

```
#pragma loop loop_fission_target
for(i = 0; i < N; i++) {
    e[i] = a1[i]*b1[i] + a2[i]*b2[i] +
          a3[i]*b3[i] + a4[i]*b4[i] +
          a5[i]*b5[i];
    f[i] = c1[i]*d1[i] + c2[i]*d2[i] +
          c3[i]*d3[i] + c4[i]*d4[i] +
          c5[i]*d5[i];
}
```

->

```
for(i = 0; i < N; i++) {
    e[i] = a1[i]*b1[i] + a2[i]*b2[i] +
          a3[i]*b3[i] + a4[i]*b4[i] +
          a5[i]*b5[i];
}

for(i = 0; i < N; i++) {
    f[i] = c1[i]*d1[i] + c2[i]*d2[i] +
          c3[i]*d3[i] + c4[i]*d4[i] +
          c5[i]*d5[i];
}
```

3.3.10.1 Strip-Mining

Strip-mining is an optimization to fragment the loop by small iteration.

Applied by this optimization for loops after automatic loop fission, improvement of the cache memory efficiency is expected for the data accessed between fissioned loops.

Strip-mining is performed when the -Kloop_fission_stripmining option is specified. The message to show the optimized loop can be output by specifying the -Koptmsg=2 option.

Examples of automatic loop fission and strip-mining with specifying the -Kloop_fission_stripmining=256 are shown as follows.

Example: Automatic loop fission and strip-mining

Source code

```
#pragma loop loop_fission_target
float a[N], b[N], p[N], q;
for(i = 0; i < N; i++) {
    q = a[i] + b[i];
    ...
    p[i] = p[i] + q;
    ...
}
```

Automatic loop fission pseudo-code

```
float a[N], b[N], p[N], q;
float temparray_q[N];
for(i = 0; i < N; i++) {
    temparray_q[i] = a[i] + b[i];
    ...
}
for(i = 0; i < N; i++) {
    p[i] = p[i] + temparray_q[i];
    ...
}
```

The temporary array temparray_q is generated by the compiler with loop fission. The array is required to hand over data between the fissioned loops in the source code. The number of array elements is the same value as the number of loop iteration.

Automatic loop fission and strip-mining pseudo-code

```
#define MIN(x,y) ((x<y)?x:y)
float a[N], b[N], p[N], q;
float temparray_q[256];
for(ii = 0; ii < N; ii = ii+256) {
    for(i = ii; i < MIN(N, ii+256); i++) {
        temparray_q[i-ii] = a[i] + b[i];
        ...
    }
    for(i = ii; i < min(N, ii+256); i++) {
```

```

        p[i] = p[i] + tmparray_q[i-ii];
        ...
    }
}

```

A loop is generated outside of the fissioned loop. The original loop iteration is fragmented by the strip length 256. The number of elements of temporary array tmparray_q is 256.

3.3.11 Strict Aliasing

Strict Aliasing is optimization with considering overlaps of memory regions according to the strict aliasing rules defined by language standard. Some optimizations such as the movement of invariant expressions and elimination of common expressions are promoted.

Strict Aliasing can be controlled with the -Kstrict_aliasing and -Knostrict_aliasing options.

Examples of alias are shown below.

Example 1:

If the -Kstrict_aliasing option is specified, the int type and the float type have no alias.

```

void sub(int *a, float *b) {
    long i;
    for (i=0; i<256; i++) {
        a[i] = *b;
    }
}

```

Example 2:

Even if the -Kstrict_aliasing option is specified, the int type and the unsigned int type have alias.

```

void sub(int *a, unsigned int *b) {
    long i;
    for (i=0; i<256; i++) {
        a[i] = *b;
    }
}

```

Example 3:

Even if the -Kstrict_aliasing option is specified, the char type has alias with all types.

```

void sub(int *a, char *b) {
    long i;
    for (i=0; i<256; i++) {
        a[i] = *b;
    }
}

```

Example 4:

If the -Kstrict_aliasing option is specified to the source code that does not conform to the rules, the compiler may incorrectly determined that there is no alias. The execution result may be incorrect.

```

void sub(int *i_pointer, long *l_pointer) {
    *i_pointer = 10;
    *l_pointer = 15;
}
int main() {
    long double l_double = 0.0;
    int *i_pointer = &l_double;
    long *l_pointer = &l_double;
    sub(i_pointer, l_pointer);
    if (*i_pointer == *l_pointer) {
        // OK
    }
}

```

```

    } else {
        // NG
    }
    return 0;
}

```

3.4 Using Optimization Functions

This section describes techniques for effectively using the optimization functions.

3.4.1 Using Optimization Control Line (OCL)

It may be possible to increase the optimization level using certain `#pragma` directives in the source code. The following `#pragma` directive falls into this category. The `-Kocl` option enables the `#pragma` directive optimization, and the `-Knoocl` option suppresses the effect of the `#pragma` directive optimization.

3.4.1.1 Types of Optimization Control Lines

The types of optimization control lines and their scope are shown in "[Table 3.2 Scope of Optimization Control Lines](#)".

Table 3.2 Scope of Optimization Control Lines

Preprocessing Literal Line	Type of optimization control line	Scope
global	global line	Compilation unit
procedure	procedure line	Function
loop	loop line	Subsequent loop
statement	statement line	Subsequent statement

global line

The global line is used when important information in the compilation unit is given to the compiler.

Description Format

```
#pragma[ fj] global Optimization-control-specifier
```

Insert Position

This is positioned at the top of the corresponding compilation unit.

procedure line

The procedure is used when important information in a function is given to the compiler.

Description Format

```
#pragma[ fj] procedure Optimization-control-specifier
```

Insert Position

This is positioned between a declaration statement of the corresponding function (including the statement in which both the declaration and initialization value are specified) and an initial execution statement.

loop line

The loop line is used when valuable information is given to the compiler.

Description Format

```
#pragma[ fj] loop Optimization-control-specifier
```

Insert Position

This is positioned just before the corresponding loop.

statement line

The statement line is used when the valuable information is instructed to the compiler.

Description Format

```
#pragma[ fjl] statement Optimization-control-specifier
```

Insert Position

This is positioned just before the corresponding statement.



Note

Only one *optimization-control-specifier* can be specified for one optimization control line.

Two or more *optimization-control-specifiers* can be specified with consecutive lines.

3.4.1.2 Optimization Control Specifier

The optimization control specifiers that can be used in the optimization control line are shown in "[Table 3.3 Optimization Control Specifiers that can be specified to optimization control line](#)".

Table 3.3 Optimization Control Specifiers that can be specified to optimization control line

Optimization Control Specifiers	Explanation	Optimization control line that can be Specified ^[a]			
		global line	procedure line	loop line	statement line
array_declaration_opt	Specifies to perform optimization assuming that the array subscript does not exceed the range of the array declaration.	*	*	*	-
noarray_declaration_opt	Specifies to perform the optimization without assuming that the array subscript does not exceed the range of the array declaration.	*	*	*	-
assume {shortloop noshortloop memory_bandwidth nomemory_bandwidth time_saving_compilation notime_saving_compilation}	Directs to control optimization considering features of the program.	*	*	-	-
clone var==n1[,n2]... ^[b]	<p>Directs to generate conditional branches along to the arguments and generate loop copies and put them for the conditional blocks assuming that the <i>var</i> is invariant in the loops.</p> <p>The conditional expressions are equals-to expressions which operands are <i>var</i> and <i>n1[,n2]...</i> of specified arguments.</p> <p>Variable <i>var</i> is a name of integer variable.</p> <p>An integer value from -9223372036854775808 to 9223372036854775807 can be specified as constant values(<i>n1[, n2]...</i>).</p>	-	-	*	-

Optimization Control Specifiers	Explanation	Optimization control line that can be Specified ^[a]			
		global line	procedure line	loop line	statement line
eval	Performs the optimization that changes the expression evaluation sequence.	*	*	*	-
noeval	Suppresses the optimization that changes the expression evaluation sequence.	*	*	*	-
eval_concurrent	Gives the priority to the instruction-level parallelism in the tree-height-reduction optimization.	*	*	*	-
eval_noconcurrent	Suppresses the instruction-level parallelism and give priority to utilizing FMA instructions in the tree-height-reduction optimization.	*	*	*	-
extract_stride_store	Directs to use scalar instructions for stride access store in the target loop.	*	*	*	-
noextract_stride_store	Directs to use SIMD instructions for stride access store in the target loop.	*	*	*	-
fission_point [n]	Performs loop fission optimization. <i>n</i> indicates the count of nested loops to be distributed from innermost loop, which should be an integer value from 1 to 6.	-	-	-	*
fp_contract	Performs optimizations using the Floating-Point Multiply-Add/Subtract instructions.	*	*	*	-
nofp_contract	Suppresses optimizations using the Floating-Point Multiply-Add/Subtract instructions.	*	*	*	-
fp_relaxed	Converts floating point divisions and sqrt functions into reciprocal approximation instructions.	*	*	*	-
nofp_relaxed	Converts floating point divisions and sqrt functions into normal instructions and sqrt instruction.	*	*	*	-
fullunroll_pre_simd [n]	Performs the prioritize promotion of the full unrolling before SIMD. <i>n</i> is the upper limit of the iteration count for a target loop. <i>n</i> is a number from 2 to 100.	-	-	*	-
nofullunroll_pre_simd	Suppresses the full unrolling before SIMD.	-	-	*	-
iterations max= <i>n1</i> iterations avg= <i>n2</i> iterations min= <i>n3</i>	Performs optimization assuming the maximum iteration count is max= <i>n1</i> , the average of the iteration count is avg= <i>n2</i> , and the minimum iteration count is min= <i>n3</i> . An integer value from 0 to 2147483647 can be specified as <i>n1</i> , <i>n2</i> and <i>n3</i> . One or more max, avg, and min specifiers can be specified in random order. When specifying	*	*	*	-

Optimization Control Specifiers	Explanation	Optimization control line that can be Specified ^[a]			
		global line	procedure line	loop line	statement line
	max, avg, and min at the same time, use space to separate them like "iterations max=n1 avg=n2 min=n3".				
loop_blocking <i>n</i>	Performs the loop-blocking optimization. <i>n</i> indicates the block size which should be an integer value from 2 to 10000.	*	*	*	-
loop_noblocking	Suppresses the loop-blocking optimization.	*	*	*	-
loop_fission_stripmining [<i>n</i> " <i>c-level</i> "]	Performs the strip-mining optimization for automatic fissioned loops. <i>n</i> is a number from 2 to 100000000. <i>c-level</i> is L1 or L2.	*	*	*	-
loop_nofission_stripmining	Suppresses the strip-mining optimization for automatic fissioned loops.	*	*	*	-
loop_fission_target [<i>cl</i> <i>ls</i>]	Performs automatic loop fission optimization.	-	-	*	-
loop_fission_threshold <i>n</i>	Specifies the threshold to decide the granularity of loop after automatic loop fission. <i>n</i> is a number from 1 to 100.	*	*	*	-
loop_interchange <i>var1, var2[, var3]...</i> ^[b]	Designates that the sequence of a nested loop is changed according to the specified order.	-	-	*	-
loop_nointerchange	Suppresses the change of the sequence of a nested loop.	-	-	*	-
loop_nofusion	Suppresses the loop fusion.	*	*	*	-
loop_part_simd	Performs optimization which divides loops and partially uses SIMD extensions.	*	*	*	-
loop_nopart_simd	Suppresses optimization which divides loops and partially uses SIMD extensions.	*	*	*	-
loop_perfect_nest	Performs optimization which fissions the imperfectly nested loop into the perfectly nested loops.	*	*	*	-
loop_noperfect_nest	Suppress optimization which fissions the imperfectly nested loop into the perfectly nested loops.	*	*	*	-
loop_versioning	Designates to perform optimization by the loop versioning.	*	*	*	-
loop_noversioning	Cancels the loop_versioning.	*	*	*	-
mfunc [<i>level</i>]	Designates to change the function to a multi-operation function. Decimal 1, 2 or 3 can be specified as <i>level</i> .	*	*	*	-
nomfunc	Designates canceling to change the function to a multi-operation function.	*	*	*	-

Optimization Control Specifiers	Explanation	Optimization control line that can be Specified ^[a]			
		global line	procedure line	loop line	statement line
<code>noalias</code>	Designates that pointer variables and other variable do not share memory area.	*	*	*	-
<code>norecurrence [array1[,array2]...] ^[b]</code>	Declares array for which the loop slice optimization can be performed.	*	*	*	-
<code>novrec [array1[,array2]...] ^[b]</code>	Instructs to the system that there is no recurrent operation with the specified array in the specified loop.	*	*	*	-
<code>preex</code>	Optimizes by evaluating an invariant first.	*	*	*	-
<code>nopreex</code>	Suppresses the optimization of evaluating an invariant first.	*	*	*	-
<code>prefetch</code>	Performs the automatic prefetch function of the compiler.	*	*	*	-
<code>noprefetch</code>	Suppresses the automatic prefetch function of the compiler.	*	*	*	-
<code>prefetch_cache_level c-level</code>	Designates cache-level to prefetch of data. 1, 2 or all can be specified as <i>c-level</i> .	*	*	*	-
<code>prefetch_conditional</code>	Directs the system to generate the prefetch instruction to array data used in blocks in "if" statements or "switch" statements.	*	*	*	-
<code>prefetch_noconditional</code>	Suppresses the prefetch_conditional	*	*	*	-
<code>prefetch_indirect</code>	Directs the system to generate the prefetch instruction to array data accessed indirectly in loops.	*	*	*	-
<code>prefetch_noindirect</code>	Suppresses the prefetch_indirect	*	*	*	-
<code>prefetch_infer</code>	Directs the system to generate the prefetch instruction for sequential access when the distance of prefetch is unknown.	*	*	*	-
<code>prefetch_noinfer</code>	Suppresses the prefetch_infer.	*	*	*	-
<code>prefetch_iteration n</code>	Directs the system to be target of a prefetch instruction would be data that is referred to <i>n</i> iterations of a loop. This targets prefetch instructions only for the first level cache. Decimal from 1 to 10000 can be specified as <i>n</i> . If the optimization using SIMD extensions is applied to a loop, specify the loop iteration count after using SIMD extensions for <i>n</i> .	*	*	*	-
<code>prefetch_iteration_L2 n</code>	Directs the system to be target of a prefetch instruction would be data that is defined or is referred to <i>n</i> iterations of a loop. This targets prefetch instructions only for the second level cache.	*	*	*	-

Optimization Control Specifiers	Explanation	Optimization control line that can be Specified ^[a]			
		global line	procedure line	loop line	statement line
	Decimal from 1 to 10000 can be specified as <i>n</i> . If the optimization using SIMD extensions is applied to a loop, specify the loop iteration count after using SIMD extensions for <i>n</i> .				
<code>prefetch_line n</code>	Directs the system to generate prefetch instructions to prefetch data after <i>n</i> cache line(s). This targets prefetch instructions only for first level cache. Decimal from 1 to 100 can be specified as <i>n</i> .	*	*	*	-
<code>prefetch_line_L2 n</code>	Directs the system to generate prefetch instructions to prefetch data after <i>n</i> cache line(s). This targets prefetch instructions only for second level cache. Decimal from 1 to 100 can be specified as <i>n</i> .	*	*	*	-
<code>prefetch_sequential [auto soft]</code>	Directs that prefetch instructions are created for array data that is accessed sequentially. If neither <code>auto</code> nor <code>soft</code> is specified, <code>auto</code> is set by default.	*	*	*	-
<code>prefetch_nosequential</code>	Directs that prefetch instructions are not created for array data that is accessed sequentially.	*	*	*	-
<code>prefetch_stride [soft hard_auto hard_always]</code>	Directs that prefetch instructions are generated for array data that is accessed with a stride larger than the cache line size used in a loop.	*	*	*	-
<code>prefetch_nostride</code>	Directs that prefetch instructions are not generated for array data that is accessed with a stride larger than the cache line size used in a loop.	*	*	*	-
<code>prefetch_strong</code>	Directs that the prefetch instructions for the first level cache are to be the strong prefetch.	*	*	*	-
<code>prefetch_nostrong</code>	Directs that the prefetch instructions generated for the first level cache will not be strong prefetch.	*	*	*	-
<code>prefetch_strong_L2</code>	Directs that the prefetch instructions generated for the second level cache are to be strong prefetch. The prefetch instructions for the second level cache generated for the target loop will be strong prefetch.	*	*	*	-
<code>prefetch_nostrong_L2</code>	Directs that the prefetch instructions generated for the second level cache are not to be strong prefetch.	*	*	*	-

Optimization Control Specifiers	Explanation	Optimization control line that can be Specified ^[a]			
		global line	procedure line	loop line	statement line
<code>preload</code>	Directs to perform the speculative execution of load instructions	*	*	*	-
<code>nopreload</code>	Directs to suppress the speculative execution of load instructions	*	*	*	-
<code>scache_isolate_way L2=n1 [L1=n2]</code> and <code>end_scache_isolate_way</code>	Directs the maximum number of ways in sector 1 of cache.	-	*	-	*
<code>scache_isolate_assign array1[,array2]...</code> ^[b] and <code>end_scache_isolate_assign</code>	Directs the array to store in sector 1 of cache.	-	*	-	*
<code>simd [aligned unaligned]</code>	Performs the optimization that uses SIMD Extensions.	*	*	*	-
<code>nosimd</code>	Suppresses the optimization that uses SIMD Extensions.	*	*	*	-
<code>simd_listv [all then else]</code>	Performs list vector conversion to the statements in the block of "if" statement.	-	-	-	*
<code>simd_noredundant_vl</code>	This directs that SIMD with redundant executions for the SIMD width is not applied.	*	*	*	-
<code>simd_use_multiple_structures</code>	Directs to use the SVE Load Multiple Structures instructions and SVE Store Multiple Structures instructions.	*	*	*	-
<code>simd_nouse_multiple_structures</code>	Directs not to use the SVE Load Multiple Structures instructions and SVE Store Multiple Structures instructions.	*	*	*	-
<code>striping [n]</code>	Performs loop striping. <i>n</i> indicates the stripe-length (number of expansions) which should an integer value be from 2 to 100.	*	*	*	-
<code>nostriping</code>	Suppresses loop striping.	*	*	*	-
<code>swp</code>	Performs software pipelining.	*	*	*	-
<code>noswp</code>	Suppresses software pipelining.	*	*	*	-
<code>swp_freq_rate n</code> <code>swp_ireg_rate n</code> <code>swp_preg_rate n</code>	Changes the condition of the register number of software pipelining. <i>n</i> indicates the rate (percentage) of the register number that can be used by software pipelining. <i>n</i> is a number from 1 to 1000.	*	*	*	-
<code>swp_policy {auto small large}</code>	Specifies a policy to select an instruction scheduling algorithm used in software pipelining.	*	*	*	-

Optimization Control Specifiers	Explanation	Optimization control line that can be Specified ^[a]			
		global line	procedure line	loop line	statement line
swp_weak	Adjusts software pipelining to make overlapping of the instructions less.	*	*	*	-
unroll [n "full"]	Performs unrolling for the corresponding loop. <i>n</i> is an iteration count for unrolling and is an integer value from 2 to 100.	-	-	*	-
nounroll	Suppresses unrolling for the corresponding loop.	-	-	*	-
unroll_and_jam [n]	Performs unroll-and-jam optimization to the loop when the optimization is expected to be effective. <i>n</i> is an iteration count for unrolling and is an integer value from 2 to 100.	*	*	*	-
unroll_and_jam_force [n]	Performs unroll-and-jam optimization. <i>n</i> is an iteration count for unrolling and is an integer value from 2 to 100.	-	-	*	-
nounroll_and_jam	Suppresses unroll-and-jam optimization.	*	*	*	-
unswitching	Instructs loop unswitching to "if" statement.	-	-	-	*
zfill [N]	Directs that the zfill optimization to be performed. The optional parameter <i>N</i> is an integer between 1 and 100 that specifies the number of blocks the DC ZVA instruction writes.	-	-	*	-
nozfill	Directs that the zfill optimization not to be performed.	-	-	*	-

[a]

* : An optimization control line accepts the optimization control specifier.

- : An optimization control line does not accept the optimization control specifier.

[b] *var*, *var1*, *var2*, *var3*, *array1*, and *array2* must be declared before they are used.

array_declaration_opt specifier

The `array_declaration_opt` specifier instructs to perform the optimization assuming that the array subscript does not exceed the range of the array declaration.

An example is shown in the following.

Example:

```
double a[8];
#pragma loop array_declaration_opt
for(int i = 0; i < n; i++) {
    a[i] = 0;
}
```

The optimization is performed assuming that the operation range of the subscript is equal to or less than the SIMD length.

`noarray_declaration_opt` specifier

The `noarray_declaration_opt` specifier instructs to perform the optimization without assuming that the array subscript does not exceed the range of the array declaration.

An example is shown in the following.

Example:

```
double a[8];
#pragma loop noarray_declaration_opt
for(int i = 0; i < n; i++) {
    a[i] = 0;
}
```

Performs optimization with the operating range of the subscript unknown.

`assume` specifier

The `assume` specifier instructs to control optimization considering features of the program. Multiple `assume` specifiers can be specified at the same time.

Either of specifiers shown below is instructed after `assume` specifier.

`shortloop`

The `assume shortloop` specifier instructs to assume that iteration counts are small when the iteration count of the innermost loop in the program is unknown at compilation. Optimizations such as automatic parallelization, loop unrolling, and software pipelining may be controlled or invalidated.

`noshortloop`

The `assume noshortloop` specifier instructs to assume that iteration counts are not small when the iteration count of the innermost loop in the program is unknown at compilation.

`memory_bandwidth`

The `assume memory_bandwidth` specifier instructs to assume that innermost loops in the program has a memory bandwidth bottleneck. Optimizations such as zfill optimization promotion and software pipelining may be controlled or invalidated.

`nomemory_bandwidth`

The `assume nomemory_bandwidth` specifier instructs to assume that innermost loops in the program does not have a memory bandwidth bottleneck.

`time_saving_compilation`

The `assume time_saving_compilation` specifier instructs to control optimization for decreasing compilation time of the program.

Optimization is controlled to decrease compilation time of the program. General optimizations and inline expansions may be controlled or invalidated.

`notime_saving_compilation`

The `assume notime_saving_compilation` specifier instructs to optimize with giving priority to the speed of the executable program rather than decrease of the compilation time.

An example is shown in the following.

Example:

```
#pragma global assume shortloop
#pragma global assume memory_bandwidth
#pragma global assume time_saving_compilation
void sub() {
    for(int i = 0; i < N; i++) {
        a[i] = b[i];
    }
}
```



```
}
}
```

The optimization is controlled considering the specified features in the target loop.

clone specifier

The `clone` specifier instructs to generate conditional branches along to the arguments and generate loop copies and put them for the conditional blocks assuming that the `var` is invariant in the loops. The order of the generated branches is the order of the corresponding arguments. This specifier promotes other optimizations, such as full unrolling. The execution result is not guaranteed in case that the value of variable `var` changes in the loop. For more details, refer to "[3.4.1.3 Notes for Optimization Control Specifiers](#)". The size of the object program and compile time may increase because the clone makes copies of loops. This specifier is effective only when the `-O3` option is set.

Variable `var` is a name of integer variable.

You cannot specify following integer variable.

- Structure member variable
- Union member variable
- Array elements
- Threadprivate variable

An integer value from -9223372036854775808 to 9223372036854775807 can be specified as constant values (`n1[,n2]...`).

You can use comma to enumerate constant values and colon to specify a range. The following two specifications are equivalent.

```
#pragma loop clone var==1,3:5,7
```

```
#pragma loop clone var==1,3,4,5,7
```

You can specify up to 20 values at once. When more than 20 values are specified, excess values are ignored. The following specification is limited in the number of values because it specifies 30 values.

```
#pragma loop clone var==11:40
```

The above specification is treated as the same as the following specification because excess values are ignored.

```
#pragma loop clone var==11:30
```

Examples of correct usage are following.

Example 1:

Loop is copied with "if" statements in the order of the specification.

```
#pragma loop clone N==10,20
for (i=0;i<N; ++i) {
    a[i] = i;
}
```

->

```
if (N==10) {
    for (i=0;i<10;++i) {
        a[i] = i;
    }
} else if (N==20) {
    for (i=0;i<20;++i) {
        a[i] = i;
    }
} else {
    for (i=0;i<N;++i) {
        a[i] = i;
    }
}
```

Example 2:

Loop is copied with "if" statements in the order of the multiple specifications.

```
#pragma loop clone N==10
#pragma loop clone M==20
for (i=0;i<N;++i) {
    a[i] = M;
}
```

->

```
if (N==10) {
    for (i=0;i<10;++i) {
        a[i] = M;
    }
} else if (M==20) {
    for (i=0;i<N;++i) {
        a[i] = 20;
    }
} else {
    for (i=0;i<N;++i) {
        a[i] = M;
    }
}
```

Example 3:

clone specifiers for nested loops generates nested "if" statements.

```
#pragma loop clone M==10
for (i=0;i<M;++i) {
    #pragma loop clone N==20
    for (j=0;j<N;++j) {
        a[i][j] = 0;
    }
}
```

->

```
if (M==10) {
    for (i=0;i<10;++i) {
        if (N==20) {
            for (j=0;j<20;++j) {
                a[i][j] = 0;
            }
        } else {
            for (j=0;j<N;++j) {
                a[i][j] = 0;
            }
        }
    }
} else {
    for (i=0;i<M;++i) {
        if (N==20) {
            for (j=0;j<20;++j) {
                a[i][j] = 0;
            }
        } else {
            for (j=0;j<N;++j) {
                a[i][j] = 0;
            }
        }
    }
}
```

eval specifier

The eval specifier instructs to perform the optimization that changes the expression evaluation sequence.

An example is shown in the following.

Example:

```
#pragma loop eval
for(i = 0; i < n; i++) {
    a[i] = a[i] + b[i] + c[i] + d[i];
}
```

->

```
for(i = 0; i < n; i++) {
    a[i] = (a[i] + b[i]) + (c[i] + d[i]);
}
```

The optimization that changes the expression evaluation sequence is performed in the target loop.

noeval specifier

The noeval specifier instructs to suppress the optimization that changes the expression evaluation sequence.

An example is shown in the following.

Example:

```
#pragma loop noeval
for(i = 0; i < n; i++) {
    a[i] = a[i] + b[i] + c[i] + d[i];
}
```

->

```
for(i = 0; i < n; i++) {
    a[i] = (a[i] + b[i]) + (c[i] + d[i]);
}
```

The optimization that changes the expression evaluation sequence is suppressed in the target loop.

eval_concurrent specifier

The `eval_concurrent` specifier instructs to give priority to the instruction-level parallelism in the tree-height-reduction optimization.

An example is shown in the following.

Example:

```
#pragma loop eval_concurrent
for(i = 0; i < n; i++) {
    x[i] = a[i] * b[i] + c[i] * d[i] + e[i] * f[i] + g[i] * h[i];
}
```

See section "[3.3.9 Tree-Height-Reduction Optimization](#)" for the tree-height-reduction optimization.

eval_noconcurrent specifier

The `eval_noconcurrent` specifier instructs to suppress the instruction-level parallelism and specifies to give priority to utilizing FMA instructions in the tree-height-reduction optimization.

An example is shown in the following.

Example:

```
#pragma loop eval_noconcurrent
for(i = 0; i < n; i++) {
    x[i] = a[i] * b[i] + c[i] * d[i] + e[i] * f[i] + g[i] * h[i];
}
```

See section "[3.3.9 Tree-Height-Reduction Optimization](#)" for the tree-height-reduction optimization.

extract_stride_store specifier

The `extract_stride_store` specifier instructs to use scalar instructions for stride access store in the target loop.

An example is shown in the following.

Example:

```
#pragma loop extract_stride_store
for(int i = 0; i < n; i+=2) {
    a[i] = b[i] + c[i];
}
```

Performs optimization using scalar instructions for store of array a that can be converted to SIMD instruction.

noextract_stride_store specifier

The `noextract_stride_store` specifier instructs to use SIMD instructions for stride access store in the target loop.

An example is shown in the following.

Example:

```
#pragma loop extract_stride_store
for(int i = 0; i < n; i+=2) {
    a[i] = b[i] + c[i];
}
```

Performs optimization using SIMD instructions for array a.

fission_point specifier

The `fission_point` specifier instructs to perform loop fission optimization.

The number from 1 to 6 following the specifier instructs the count of nested loops to be distributed from innermost loop. If the count is omitted, the compiler distributes innermost loop.

The `fission_point` specifier is effective only when it is placed in the innermost loop.

An example is shown in the following.

Example:

```
for(j = 1; j < n; j++) {
    for(i = 1; i < n; i++) {
        a[i][j] = a[i-1][j-1];
        #pragma statement fission_point 1
        a[i][j] = a[i][j] + a[i-1][j];
    }
}
```

->

```
for(j = 1; j < n; j++) {
    for(i = 1; i < n; i++) {
        a[i][j] = a[i-1][j-1];
    }
    for(i = 1; i < n; i++) {
        a[i][j] = a[i][j] + a[i-1][j];
    }
}
```



Note

If loop fission is performed on a loop that cannot be executed because of the program logic, Some side effects, such as exceptions on runtime, can happen.

For instance, the compiler generates some temporary arrays for passing data between the split loops. In this case, the necessary instructions may be executed speculatively.

In the following example, the load instructions of the array element `IDX[M][M]` is speculatively executed. If `N` is less than or equal to 0 and `M` used in array element `IDX[M][M]` is greater than the declared size, an exception may be thrown at runtime.

Example:

```
for(j = 0; j < N; j++) {
    for(i = 0; i < IDX[M][M]; i++) {
        x = a[i] + b[i];
        #pragma statement fission_point 1
        c[i] = x;
    }
}
```

→

```
double *tmp;
tmp = (double *)malloc(IDX[M]
[M]*sizeof(double)); // speculative execution
for(j = 0; j < N; j++) {
    for(i = 0; i < IDX[M][M]; i++) {
        tmp[i] = a[i] + b[i];
    }
    for(i = 0; i < IDX[M][M]; i++) {
        c[i] = tmp[i];
    }
}
free(tmp);
```

fp_contract specifier

The `fp_contract` specifier instructs to apply optimizations using the Floating-Point Multiply-Add/Subtract instructions. However, using these instructions may cause calculation errors in the result due to rounding errors.

An example is shown in the following.

Example:

```
#pragma loop fp_contract
for(i = 0; i < n; i++) {
    a[i] = a[i] + b[i] * c[i];
}
```

The optimization that uses the Floating-Point Multiply-Add/Subtract instructions is performed in the target loop.

nofp_contract specifier

The `nofp_contract` specifier instructs to suppress optimizations using the Floating-Point Multiply-Add/Subtract instructions.

An example is shown in the following.

Example:

```
#pragma loop nofp_contract
for(i = 0; i < n; i++) {
    a[i] = a[i] + b[i] * c[i];
}
```

The optimization that uses the Floating-Point Multiply-Add/Subtract instructions is not performed in the target loop.

fp_relaxed specifier

The `fp_relaxed` specifier instructs to convert floating point division operations and `sqrt` functions into the reciprocal approximation operation with the reciprocal approximation instructions and the Floating-Point Multiply-Add/Subtract instructions. This conversion is applied to the single-precision and double-precision real type calculation. At the result of this conversion, the program is executed faster. However it may cause calculation errors in the result due to rounding errors. Also, this optimization may result in generating a floating-point exception whether the `-NRtrap` option is valid or not.

When the `-NRtrap` option and either of the `-Knosimd` option or the `-KNOSVE` option are effective, the optimization to convert a `sqrt` function into reciprocal approximation instructions is suppressed. Therefore, the execution performance may decrease as compared with when the `-NRnotrap` option is effective. Note that the reciprocal approximation instructions generated by this specifier may be evaluated in advance even if the `-Knopreex` option is enabled. And a floating-point exception may occur when the `-Knopreex` option, `-NRtrap` option, and this specifier are valid.

An example is shown in the following.

Example:

```
#pragma loop fp_relaxed
for(i = 0; i < n; i++) {
    a[i] = sqrt(b[i] / c[i]);
}
```

The optimization of using the reciprocal approximation instructions is performed in the target loop.

nofp_relaxed specifier

The `nofp_relaxed` specifier instructs to convert floating point division operations and `sqrt` functions into the normal division and `sqrt` instructions without using reciprocal approximation instructions.

An example is shown in the following.

Example:

```
#pragma loop nofp_relaxed
for(i = 0; i < n; i++) {
    a[i] = sqrt(b[i] / c[i]);
}
```

The normal division and `sqrt` instructions are used for the target loop.

fullunroll_pre_simd specifier

The `fullunroll_pre_simd` specifier instructs to perform the prioritize promotion of the full unrolling before SIMD.

The number from 2 to 100 following the specifier instructs the upper limit of the iteration count for a target loop. If the upper limit is not specified, the compiler will automatically determine the value.

Note that the `fullunroll_pre_simd` specifier targets only the loop immediately after the optimization control line.

If the number of the iteration is unknown at compilation, no optimization of the full unrolling before SIMD is performed.

An example is shown in the following.

Example:

```
for(i = 0; i < n; i++) {  
    #pragma loop fullunroll_pre_simd  
    for(j = 0; j < 16; j++) {  
        a[i][j] = b[j][i] + c[j][i];  
    }  
}
```

Full unrolling before SIMD is applied for the inner loop.

nofullunroll_pre_simd specifier

The `nofullunroll_pre_simd` specifier instructs to suppress the full unrolling before SIMD.

Note that the `nofullunroll_pre_simd` specifier targets only the loop immediately after the optimization control line.

An example is shown in the following.

Example:

```
for(i = 0; i < n; i++) {  
    #pragma loop nofullunroll_pre_simd  
    for(j = 0; j < 5; j++) {  
        a[i][j] = b[j][i] + c[j][i];  
    }  
}
```

Full unrolling before SIMD is not applied to the inner loop.

iterations max=n1 specifier

iterations avg=n2 specifier

iterations min=n3 specifier

The iterations specifier instructs to optimize assuming the specified value is the loop iteration count.

`max=n1`, `avg=n2`, and `min=n3` are specified following the specifier. An integer value from 0 to 2147483647 can be specified as *n1*, *n2*, and *n3*. One or more `max`, `avg`, and `min` specifiers can be specified in random order. When specifying `max`, `avg`, and `min` at the same time, use space to separate them like "iterations max=n1 avg=n2 min=n3". Note that values *n1*, *n2* and *n3* specified as `max`, `avg`, and `min` are required to be *n1* >= *n2* >= *n3*.

When `max=n1` is specified following the specifier, the optimization assuming the maximum loop iteration count is *n1* is performed.

When `avg=n2` is specified following the specifier, the optimization assuming the average loop iteration count is *n2* is performed.

When `min=n3` is specified following the specifier, the optimization assuming the minimum loop iteration count is *n3* is performed.

This specifier is effective when the loop iteration count is unknown at compilation.

The execution result is guaranteed even if the actual average iteration loop count is different from *n2* because the specified average loop iteration count *n2* is used as an optimization hint.



Note

The execution result is not guaranteed in the following cases:

- When the actual loop iteration count is greater than *n1*
- When the actual loop iteration count is less than *n3*

For details, see the `iterations max=n1` specifier and `iterations min=n3` specifier in Section "[3.4.1.3 Notes for Optimization Control Specifiers](#)".

This specifier can be specified at global line, procedure line, and loop line, but it is recommended to specify at loop line because the iteration counts of loops in the program may not be the same.

An example is shown in the following.

Example 1: Single iteration count avg

```
#pragma loop iterations avg=32
for(i = 0; i < m; i++) {
    a[i] = b[i] + c[i];
}
```

Optimized assuming average loop iteration count is 32.

Example 2: Multiple iteration count min and avg

```
#pragma loop iterations min=1 avg=8
for(i = 0; i < m; i++) {
    a[i] = b[i] + c[i];
}
```

Optimized assuming minimum loop iteration count is 1, and average loop iteration count is 8.

Example 3: Multiple iteration count max, avg, and min

```
#pragma loop iterations max=128 avg=16 min=16
for(i = 0; i < m; i++) {
    a[i] = b[i] + c[i];
}
```

Optimized assuming maximize loop iteration count is 128, average loop iteration count is 16, and minimum iteration count is 16.

loop_blocking specifier

The `loop_blocking` specifier instructs to apply the loop-blocking optimization. *n* is a decimal value ranging from 2 to 10000, and specifies the block size. If *n* is omitted, the compiler automatically determines a suitable value for *n*.

An example is shown in the following.

Example:

```
#define MIN(x,y) ((x<y)?x:y)
#pragma loop loop_blocking 80
for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
        a[i][j] = a[i][j] + b[i][j];
    }
}
```

->

```
for(ii = 0; ii < n; ii += 80) {
    for(jj = 0; jj < n; jj += 80) {
        for(i = ii; i < MIN(n,(ii+80)); i++) {
            for(j = jj; j < MIN(n,(jj+80)); j++) {
                a[i][j] = a[i][j] + b[i][j];
            }
        }
    }
}
```

The optimization of loop blocking by block size 80 is performed in the target loop.

loop_noblocking specifier

The `loop_noblocking` specifier instructs to suppress the loop-blocking optimization.

An example is shown in the following.

Example:

```
#pragma loop loop_noblocking
for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
        a[i][j] = a[i][j] + b[i][j];
    }
}
```

The optimization of loop blocking by block size 80 is suppressed in the target loop.

loop_fission_stripmining specifier

The `loop_fission_stripmining` specifier instructs to perform the strip-mining optimization for automatic fissioned loops.

The strip length is specified by *n*, "L1", or "L2" following the specifier. If the strip length is omitted, the compiler chooses the strip length automatically.

When *n* is specified, the strip length is as *n*. *n* is a number from 2 to 100000000.

When "L1" is specified, the strip length is adjusted to the size of the first level cache for the cache memory efficiency.

When "L2" is specified, the strip length is adjusted to the size of the second level cache as well.

This specifier is effective when the `loop_fission_target` specifier is specified in the optimization control line, and the `-Kloop_fission` option and the `-O2` option or higher are set.

For details about strip-mining, see Section ["3.3.10.1 Strip-Mining"](#).

An example is shown in the following.

Example:

```
float a[N],b[N],p[N],q;
#pragma loop loop_fission_target
#pragma loop loop_fission_stripmining 256
for(i = 0; i < N; i++) {
    q = a[i] + b[i];
    ...
    p[i] = p[i] + q;
    ...
}
```

->

```
#define MIN(x,y) ((x<y)?x:y)
float a[N],b[N],p[N];
float temparray_q[256];
for(ii = 0; ii < N; ii=ii+256) {
    for(i = ii; i < MIN(N, ii+256); i++) {
        temparray_q[i-ii] = a[i] + b[i];
        ...
    }
    for(i = ii; i < min(N, ii+256); i++) {
        p[i] = p[i] + temparray_q[i-ii];
        ...
    }
}
```

loop_nofission_stripmining specifier

The `loop_nofission_stripmining` specifier instructs to suppress the stripmining optimization for automatic fissioned loops.

An example is shown in the following.

Example:

```
float a[N], b[N], p[N], q;
#pragma loop loop_fission_target
#pragma loop loop_nofission_stripmining
for(i = 0; i < N; i++) {
    q = a[i] + b[i];
    ...
    p[i] = p[i] + q;
    ...
}
```

loop_fission_target specifier

The `loop_fission_target` specifier instructs to perform automatic loop fission for target loop.

The `loop_fission_target c1` specifier instructs to perform loop fission by the clustering algorithm for target loop. This optimization performs the loop fission to prioritize reduction of work array for temporary data transfer by the loop fission. This optimization may increase the translation time.

The `loop_fission_target ls` specifier instructs to perform loop fission by the local search algorithm for target loop. This optimization performs the loop fission to prioritize promotion of the software pipelining. This optimization may increase the translation time longer than the clustering algorithm.

If neither `c1` nor `ls` is specified, `c1` is set by default.

For more details, refer to ["3.3.10 Loop Fission"](#).

Examples are shown in the following.

Example 1: c1 specified

```
#pragma loop loop_fission_target c1
for(i = 0; i < n; i++) {
    s1 = a[i] + b[i];
    s2 = c[i] + d[i];
    ...
    p[i] = s1 + q[i];
    x[i] = s2 + y[i];
    ...
}
```

The loop is fissioned to prioritize reduction of work array for temporary data transfer by the loop fission.

Example 2: ls specified

```
#pragma loop loop_fission_target ls
for(i = 0; i < n; i++) {
    s1 = a[i] + b[i];
    s2 = c[i] + d[i];
    ...
    p[i] = s1 + q[i];
    x[i] = s2 + y[i];
    ...
}
```

The loop is fissioned to prioritize promotion of the software pipelining.

loop_fission_threshold specifier

The `loop_fission_threshold` specifier specifies the threshold to decide the granularity of loop after automatic loop fission. *n* is a number from 1 to 100, and specifies the threshold to decide the granularity of loop after fission.

This specifier is effective when the `loop_fission_target` specifier is specified in the optimization control line, and the `-Kloop_fission` option and the `-O2` option or higher are set.

Examples are shown in the following. In the example 2 which specifying a small value for the `loop_fission_threshold` specifier, the compiler fissions the loop into small granularity compared with example 1 and the number of fissioned loops increase.

Example 1:

```
#pragma loop loop_fission_target
#pragma loop loop_fission_threshold 50
for(i = 0; i < N; i++) {
    a1[i] = a1[i] + b1[i];
    ...
    a2[i] = a2[i] + b2[i];
    ...
    a3[i] = a3[i] + b3[i];
    ...
    a4[i] = a4[i] + b4[i];
    ...
}
```

->

```
for(i = 0; i < N; i++) {
    a1[i] = a1[i] + b1[i];
    ...
    a2[i] = a2[i] + b2[i];
    ...
}
for(i = 0; i < N; i++) {
    a3[i] = a3[i] + b3[i];
    ...
    a4[i] = a4[i] + b4[i];
    ...
}
```

Example 2:

```
#pragma loop loop_fission_target
#pragma loop loop_fission_threshold 20
for(i = 0; i < N; i++) {
    a1[i] = a1[i] + b1[i];
    ...
    a2[i] = a2[i] + b2[i];
    ...
    a3[i] = a3[i] + b3[i];
}
```

->

```
for(i = 0; i < N; i++) {
    a1[i] = a1[i] + b1[i];
    ...
}
for(i = 0; i < N; i++) {
    a2[i] = a2[i] + b2[i];
    ...
}
```

```

...
a4[i] = a4[i] + b4[i];
...
}

```

```

for(i = 0; i < N; i++) {
    a3[i] = a3[i] + b3[i];
    ...
}
for(i = 0; i < N; i++) {
    a4[i] = a4[i] + b4[i];
    ...
}

```

loop_interchange specifier

The `loop_interchange` specifier instructs the compiler to interchange the nested loops from the inner loops in order of the specified loop control variable names. As a result of this interchange, the execution speed may improve. However, when the interchange is impossible, this optimization is not applied.

An example is shown in the following.

Example:

```

#pragma loop loop_interchange j, i
for(j = 0; j < n; j++) {
    for(i = 0; i < m; i++) {
        a[i][j] = b[i][j] * c[j][i];
    }
}

```

->

```

for(i = 0; i < m; i++) {
    for(j = 0; j < n; j++) {
        a[i][j] = b[i][j] * c[j][i];
    }
}

```

In left example, the nested loops are interchanged from the inner loop in order of the loop control variables `i` and `j`. The result of this interchange is equivalent to the loops shown in right.

loop_nointerchange specifier

The `loop_nointerchange` specifier instructs the compiler not to interchange the nested loops.

An example is shown in the following.

Example:

```

#pragma loop loop_nointerchange
for(j = 0; j < n; j++) {
    for(i = 0; i < m; i++) {
        a[i][j] = b[i][j] * c[j][i];
    }
}

```

The nested loops are not interchanged.

loop_nofusion specifier

The `loop_nofusion` specifier instructs the compiler not to fuse the neighboring loop.

An example is shown in the following.

Example:

```

#pragma loop loop_nofusion
for(i = 0; i < n; i++) {
    i++;
}
for(j = 0; j < n; j++) {
    j++;
}
for(k = 0; k < n; k++) {
    k++;
}

```

Fusing of the first and second loops is suppressed. However fusing of the second and third loops are not suppressed.

loop_part_simd specifier

The `loop_part_simd` specifier instructs to perform optimization which divides loops and partially uses SIMD extensions.

An example is shown in the following.

Example:

```
#pragma loop loop_part_simd
for(i = 1; i < n; i++) {
    a[i] = a[i] - b[i] + log(c[i]); /* SIMD extensions can be applied to this statement */
    d[i] = d[i-1] + a[i];           /* SIMD extensions cannot be applied to this statement */
}
```

↓

```
#pragma loop loop_part_simd
for(i = 1; i < n; i++) {
    a[i] = a[i] - b[i] + log(c[i]); /* SIMD execution */
}
for(i = 1; i < n; i++) {
    d[i] = d[i-1] + a[i];           /* no SIMD execution */
}
```

The loop is divided and SIMD extensions are applied to the part of the divided loop.

loop_nopart_simd specifier

The `loop_nopart_simd` specifier instructs to suppress optimization which divides loops and partially uses SIMD extensions.

An example is shown in the following.

Example:

```
#pragma loop loop_nopart_simd
for(i = 1; i < n; i++) {
    a[i] = a[i] - b[i] + log(c[i]);
    d[i] = d[i-1] + a[i];
}
```

Neither loop dividing nor using SIMD extensions is applied to the loop.

loop_perfect_nest specifier

The `loop_perfect_nest` specifier instructs to perform optimization which fissions the imperfectly nested loop into the perfectly nested loops.

An example is shown in the following.

Example:

```
#pragma loop loop_perfect_nest
for(i = 0; i < n; i++) {
    /* imperfectly nested loop */
    a[i] = b[i] + 1;
    for(j = 0; j < n; j++) {
        c[j][i] = d[j][i] + a[i];
    }
}
```

->

```
for(i = 0; i < n; i++) {
    a[i] = b[i] + 1;
}
for(i = 0; i < n; i++) {
    /* perfectly nested loop */
    for(j = 0; j < n; j++) {
        c[j][i] = d[j][i] + a[i];
    }
}
```

The imperfectly nested loops `i` and `j` are fissioned into the perfectly nested loops.

loop_noperfect_nest specifier

The `loop_noperfect_nest` specifier instructs to suppress optimization which fissions the imperfectly nested loop into the perfectly nested loops.

An example is shown in the following.

Example:

```
#pragma loop loop_noperfect_nest
for(i = 0; i < n; i++) {    /* imperfectly nested loop*/
    a[i] = b[i] + 1;
    for(j = 0; j < n; j++) {
        c[j][i] = d[j][i] + a[i];
    }
}
```

The imperfectly nested loops *i* and *j* are not optimized into the perfectly nested loops.

loop_versioning specifier

The `loop_versioning` specifier instructs to perform optimization by the loop versioning.

The loop versioning is applied to only an innermost loop, and also the loop contains only one array with unknown data dependency.

The following shows an example:

Example:

```
    for (i = 0; i < n; i++) {
#pragma loop loop_versioning
        for (j = 0; j < n; j++) {
            a[j] = a[j+m] + b[i][j];
        }
    }
```

The data dependency of array "a" is unknown at compile time because values of variable "n" and "m" are unknown at compile time.

When `loop_versioning` is specified, the compiler generates two loops with "if" statement for judging the values of variable "n" and "m" so that the data dependency of the array "a" can be analyzed at execution time.

If the array "a" judges no data dependency, SIMD may be promoted for the loop.

loop_noversioning specifier

The `loop_noversioning` specifier instructs that loop versioning is suppressed.

The `loop_noversioning` specifier is effective only for the innermost loop.

The following shows an example:

Example:

```
    for (i = 0; i < n; i++) {
#pragma loop loop_noversioning
        for (j = 0; j < n; j++) {
            a[j] = a[j+m] + b[i][j];
        }
    }
```

mfunc specifier

The `mfunc` specifier instructs to convert the function in a loop to the multi-operation function.

The number 1, 2, or 3 following the specifier instructs the level of conversion to multi-operation function. If the level is omitted, 1 is applied. For more details about the levels of conversion, see the `-K mfunc` option in Section "2.2.2.5 -K Option".

An example is shown in the following.

Example:

```
#pragma loop mfunc 1
for(i = 0; i < n; i++) {
    a[i] = log(b[i]);
}
```

The function `log(b[i])` is converted into multi-operation functions in the target loop.

`nomfunc` specifier

The `nomfunc` specifier instructs not to convert the intrinsic function in a loop to the multi-operation function.

An example is shown in the following.

Example:

```
#pragma loop nomfunc
for(i = 0; i < n; i++) {
    a[i] = log(b[i]);
}
```

The function `log(b[i])` is not converted into multi-operation functions in the target loop.

`noalias` specifier

The `noalias` specifier instructs that two or more pointer variables never point to the same memory area.

The compiler can assume from the `noalias` specifier that the pointer variables never point to the same memory area, however the memory area pointed to by the pointer variables is determined at execution.

An example is shown in the following. This promotes optimizations on the pointer variable. Note that this specifier may not promote the optimization when the value of the pointer variable is changed in the loop.

Example:

```
float *a;
float *b;
#pragma loop noalias
for (i=0; i<10; i++) {
    b[i] = a[i] + 1.0;
}
```



Note

If there are two or more pointer variables referring the same memory area in a loop with the `noalias` specifier, the execution result may be incorrect.

`norecurrence` specifier

The `norecurrence` specifier instructs that each element of the arrays which are specified by this optimization control specifier is defined and referenced within the limits of iteration.

This enables some optimizations to be performed for the loop because the order of definitions and references become certain.

For example, the following optimizations are applied:

- Loop Slicing (Automatic Parallelization)
- SIMD
- Software Pipelining

In the code example of "Example code without a `norecurrence` specifier", the system cannot determine whether array `a` can be safely sliced, because the subscript expression of array `a` is an array element `l[i]`, which is unknown at compile-time. Therefore, the system will not slice the outer loop.

Example 1: Example code without a `norecurrence` specifier

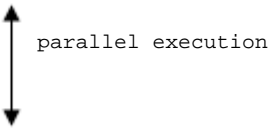
```
for(j = 0; j < 1000; j++) {
    for(i = 0; i < 1000; i++) {
        a[j][l[i]] = a[j][l[i]] + b[j][i];
    }
}
```

If the programmer knows that array a can safely be sliced, the outer loop will be sliced if the `norecurrence` specifier is used as shown in "Usage of the `norecurrence` specifier".

Example 2: Usage of the `norecurrence` specifier

```
#pragma loop norecurrence a

for(j = 0; j < 1000; j++) {
    for(i = 0; i < 1000; i++) {
        a[j][1[i]] = a[j][1[i]] + b[j][i];
    }
}
```





Note

Use of this specifier is the programmer's responsibility. If an array is specified in `norecurrence` specifier, but cannot safely be sliced, the results will be unpredictable.

`novrec` specifier

The `novrec` specifier instructs the compiler that there is no recurrent operation with array in the specified loop.

The SIMD instructions may be generated for the array looping using this specifier. However, the SIMD instructions may not be generated, depending on the type of operation and the loop structure.

When an array name is specified as an operand of the `novrec` specifier, it indicates that the array is excluded from the target of recurrent operation. When no operand is specified as the `novrec` specifier, it indicates that all arrays in the target loop are excluded from the target of recurrent operation.

An example is shown in the following.

Example:

```
double a[20], b[20];
...
#pragma loop novrec a
for(i = 1; i < 10; i++) {
    a[i] = a[i+n] + 1;
    b[i] = b[i] + 2;
}
```

The `novrec` specifier instructs that array a whose data dependency is unknown is not a target for recurrence operation. The operation of arrays a and b uses SIMD instructions.

`preex` specifier

The `preex` specifier instructs to optimize of evaluating the invariant first.

An example is shown in the following.

Example:

```
#pragma loop preex
for(i = 0; i < n; i++) {
    if(m[i] != 0) {
        a[i] = a[i] / b[k];
    }
}
```

->

```
t = 1 / b[k];
for(i = 0; i < n; i++) {
    if(m[i] != 0) {
        a[i] = a[i] * t;
    }
}
```

The optimization of evaluating the invariant first is performed in the target loop.

nopreex specifier

The **nopreex** specifier instructs to suppress the optimization of evaluating the invariant first. Note that a floating-point exception may occur when the **-Kfp_relaxed** option, **-NRtrap** option, and this specifier are valid.

An example is shown in the following.

Example:

```
#pragma loop nopreex
for(i = 0; i < n; i++) {
    if(m[i] != 0) {
        a[i] = a[i] / b[k];
    }
}
```

The optimization of evaluating the invariant first is suppressed in the target loop.

prefetch specifier

The **prefetch** specifier instructs to perform the automatic prefetch function. This function inserts the prefetch instruction in the optimal position to execute, as determined by the compiler.

An example is shown in the following.

Example:

```
#pragma loop prefetch
for(i = 0; i < 10; i++) {
    a[i] = b[i];
}
```

The automatic prefetch function is performed in the loop.

noprefetch specifier

The **noprefetch** specifier suppresses the automatic prefetch function.

An example is shown in the following.

Example:

```
#pragma loop noprefetch
for(i = 0; i < 10; i++) {
    a[i] = b[i];
}
```

The automatic prefetch function is suppressed in the loop.

prefetch_cache_level specifier

The **prefetch_cache_level** specifier instructs the data cache-level of prefetch instructions.

When **prefetch_cache_level 1** specifier is used, prefetch instructions are generated to prefetch the data to the first level cache.

When **prefetch_cache_level 2** specifier is used, prefetch instructions are generated to prefetch the data to the second level cache.

When **prefetch_cache_level all** specifier is used, prefetch instructions are generated to prefetch the data to the first level and second level caches.

Examples are shown in the following.

Example 1:

```
#pragma loop prefetch_cache_level 1
for(i = 0; i < 10; i++) {
    a[i] = b[i];
}
```

Prefetch instructions are generated to prefetch the data to the first level cache in the target loop.

Example 2:

```
#pragma loop prefetch_cache_level 2
for(i = 0; i < 10; i++) {
    a[i] = b[i];
}
```

Prefetch instructions are generated to prefetch the data to the second level cache in the target loop.

Example 3:

```
#pragma loop prefetch_cache_level all
for(i = 0; i < 10; i++) {
    a[i] = b[i];
}
```

Prefetch instructions are generated to prefetch the data to the first level and second level caches in the target loop.

`prefetch_conditional` specifier

The `prefetch_conditional` specifier instructs to generate the prefetch instructions to array data used in blocks in "if" statements or "switch" statements.

An example is shown in the following.

Example:

```
#pragma loop prefetch_conditional
for(i = 0; i < n; i++) {
    if (x[i] = y[i]) {
        a[i] = b[i];
    }
}
```

The `prefetch_conditional` specifier generates the prefetch instructions to array data used in blocks in "if" statements.

`prefetch_noconditional` specifier

The `prefetch_noconditional` specifier instructs not to generate the prefetch instructions to array data used in blocks in "if" statements or "switch" statements.

An example is shown in the following.

Example:

```
#pragma loop prefetch_noconditional
for(i = 0; i < n; i++) {
    if (x[i] = y[i]) {
        a[i] = b[i];
    }
}
```

The `prefetch_noconditional` specifier does not generate the prefetch instructions to array data used in blocks in "if" statements.

`prefetch_indirect` specifier

The `prefetch_indirect` specifier instructs to generate the prefetch instruction to array data accessed indirectly (list access) in loops.

An example is shown in the following.

Example:

```
#pragma loop prefetch_indirect
for(i = 0; i < n; i++) {
```



```

    a[c[i]] = b[c[i]];
}

```

The `prefetch_indirect` specifier generates the prefetch instruction to array data accessed indirectly (list access) in loops.

`prefetch_noindirect` specifier

The `prefetch_noindirect` specifier instructs not to generate the prefetch instruction to array data accessed indirectly (list access) in loops.

An example is shown in the following.

Example:

```

#pragma loop prefetch_noindirect
for(i = 0; i < n; i++) {
    a[c[i]] = b[c[i]];
}

```

The `prefetch_noindirect` specifier does not generate the prefetch instruction to array data indirectly accessed (list access) in loops.

`prefetch_infer` specifier

The `prefetch_infer` specifier directs the system to generate the prefetch instruction for sequential access when the distance of prefetch is unknown.

An example is shown in the following.

Example:

```

#pragma loop prefetch_infer
for(i = 0; i < n; i++) {
    a[c[i]] = b[c[i]];
}

```

Prefetch instructions are generated regarding data access sequential in the target loop.

`prefetch_noinfer` specifier

The `prefetch_noinfer` specifier directs the system to generate the prefetch instruction without assuming sequential access when the distance of prefetch is unknown.

An example is shown in the following.

Example:

```

#pragma loop prefetch_noinfer
for(i = 0; i < n; i++) {
    a[c[i]] = b[c[i]];
}

```

Prefetch instructions are generated without regarding data access sequential in the target loop.

`prefetch_iteration` specifier

The `prefetch_iteration` specifier directs the system to be target of a prefetch instruction would be data that is referred to n iterations of a loop. This targets prefetch instructions only for the first level cache. Decimal from 1 to 10000 can be specified as n .

If the optimization using SIMD extensions is applied to a loop, specify the loop iteration count after using SIMD extensions for n .

An example is shown in the following.

Example:

```

#pragma loop prefetch_iteration 50
for(i = 0; i < n; i++) {
    a[i] = b[i];
}

```

Prefetch instructions of the first level cache generated in the target loop prefetch the data of 50 iterations.

prefetch_iteration_L2 specifier

The `prefetch_iteration_L2` specifier directs the system to be target of a prefetch instruction would be data that is defined or is referred to n iterations of a loop. This targets prefetch instructions only for the second level cache. Decimal from 1 to 10000 can be specified as n .

If the optimization using SIMD extensions is applied to a loop, specify the loop iteration count after using SIMD extensions for n .

An example is shown in the following.

Example:

```
#pragma loop prefetch_iteration_L2 50
for(i = 0; i < n; i++) {
    a[i] = b[i];
}
```

Prefetch instructions of the second level cache generated in the target loop prefetch the data of 50 iterations.

prefetch_line specifier

The `prefetch_line` specifier instructs to target data after n cache line(s) for generating prefetch instruction to the first level cache. Decimal from 1 to 100 can be specified as n .

An example is shown in the following.

Example:

```
#pragma loop prefetch_line 5
for(i = 0; i < n; i++) {
    a[i] = b[i];
}
```

The `prefetch_line` specifier targets data after 5 cache lines for generating prefetch instructions to first level cache.

prefetch_line_L2 specifier

The `prefetch_line_L2` specifier instructs to target data after n cache line(s) for generating prefetch instruction to the second level cache. Decimal from 1 to 100 can be specified as n .

An example is shown in the following.

Example:

```
#pragma loop prefetch_line_L2 20
for(i = 0; i < n; i++) {
    a[i] = b[i];
}
```

The `prefetch_line_L2` specifier targets data after 20 cache lines for generating prefetch instructions to second level cache.

prefetch_sequential specifier

The `prefetch_sequential` specifier directs that prefetch instructions are created for array data that is accessed sequentially.

When `prefetch_sequential auto` specifier is used, the compiler automatically selects whether to use hardware-prefetch or to create prefetch instructions for array data that is accessed sequentially in the loop.

When `prefetch_sequential soft` specifier is used, the compiler does not use hardware-prefetch, but rather creates prefetch instructions for array data that is accessed sequentially in the loop.

If neither `auto` nor `soft` is specified, `auto` is set by default.

Examples are shown in the following.

Example 1: `auto` specified

```
#pragma loop prefetch_sequential auto
for(i = 0; i < n; i++) {
```

```

    a[i] = b[i];
}

```

Hardware prefetch is used for the target loop rather than generating prefetch instructions.

Example 2: `soft` specified

```

#pragma loop prefetch_sequential soft
for(i = 0; i < n; i++) {
    a[i] = b[i];
}

```

Prefetch instructions are generated for the target loop rather than using hardware-prefetch.

`prefetch_nosequential` specifier

The `prefetch_nosequential` specifier instructs that prefetch instructions are not generated for array data that is accessed sequentially.

An example is shown in the following.

Example:

```

#pragma loop prefetch_nosequential
for(i = 0; i < n; i++) {
    a[i] = b[i];
}

```

Prefetch instructions are not generated for the target loop.

`prefetch_stride` specifier

The `prefetch_stride` specifier instructs that prefetch instructions are generated for array data that is accessed with a stride larger than the cache line size used in a loop.

When `prefetch_stride soft` specifier is used, the compiler generates prefetch instructions and perform the prefetch.

When `prefetch_stride hard_auto` specifier is used, the compiler does not generate prefetch instructions, but rather use hardware prefetcher. This specifier sets the stride prefetcher to prefetch only data that is not on cache.

When `prefetch_stride hard_always` specifier is used, the compiler does not generate prefetch instructions, but rather use hardware prefetcher. In contrast with the `prefetch_stride hard_auto` specifier, this specifier sets the stride prefetcher to always prefetch.

If neither `soft`, `hard_auto` nor `hard_always` is specified, `soft` is set by default.

Examples are shown in the following.

Example 1: `soft` specified

```

#pragma loop prefetch_stride soft
for(i = 0; i < n; i=i+k) {
    a[i] = b[i];
}

```

Prefetch instructions are generated for the target loop.

Example 2: `hard_auto` specified

```

#pragma loop prefetch_stride hard_auto
for(i = 0; i < n; i=i+k) {
    a[i] = b[i];
}

```

Hardware prefetch is used for the target loop rather than generating prefetch instructions.

Example 3: `hard_always` specified

```

#pragma loop prefetch_stride hard_always
for(i = 0; i < n; i=i+k) {

```

```
a[i] = b[i];  
}
```

Hardware prefetch is used for the target loop rather than using prefetch instructions. Hardware prefetcher is set to always prefetch.
prefetch_nostride specifier

The `prefetch_nostride` specifier directs that prefetch instructions are not generated for array data that is accessed with a stride larger than the cache line size used in a loop.

An example is shown in the following.

Example:

```
#pragma loop prefetch_nostride  
for(i = 0; i < n; i=i+k) {  
    a[i] = b[i];  
}
```

Prefetch instructions are not generated for the target loop.

prefetch_strong specifier

The `prefetch_strong` specifier directs that prefetch instructions for the first level cache are to be the strong prefetch.

An example is shown in the following.

Example:

```
#pragma loop prefetch_strong  
for(i = 0; i < n; i++) {  
    a[i] = b[i];  
}
```

Prefetch instructions for the first level cache generated for the target loop will be strong prefetch.

prefetch_nostrong specifier

The `prefetch_nostrong` specifier directs that prefetch instructions generated for the first level cache will not be strong prefetch.

An example is shown in the following.

Example:

```
#pragma loop prefetch_nostrong  
for(i = 0; i < n; i++) {  
    a[i] = b[i];  
}
```

Prefetch instructions for the first level cache generated for the target loop will not be strong prefetch.

prefetch_strong_L2 specifier

The `prefetch_strong_L2` specifier directs that prefetch instructions for the second level cache are to be the strong prefetch.

An example is shown in the following.

Example:

```
#pragma loop prefetch_strong_L2  
for(i = 0; i < n; i++) {  
    a[i] = b[i];  
}
```

Prefetch instructions for the second level cache generated for the target loop will be strong prefetch.

prefetch_nostrong_L2 specifier

The `prefetch_nostrong_L2` specifier directs that prefetch instructions generated for the first level cache will not be strong prefetch.

An example is shown in the following.

Example:

```
#pragma loop prefetch_nostrong_L2
for(i = 0; i < n; i++) {
    a[i] = b[i];
}
```

Prefetch instructions for the second level cache generated for the target loop will not be strong prefetch.

preload specifier

The preload specifier instructs to perform the speculative execution of load instructions.

An example is shown in the following.

Example:

```
#pragma loop preload
for(i = 0; i < n; i++) {
    if(m[i] > 0) {
        c[i] = a[i] + b[i];
    }
}
```

The speculative execution of load instructions is performed in the target loop.

nopreload specifier

The nopreload specifier instructs to suppress the speculative execution of load instructions.

An example is shown in the following.

Example:

```
#pragma loop nopreload
for(i = 0; i < n; i++) {
    if(m[i] > 0) {
        c[i] = a[i] + b[i];
    }
}
```

The speculative execution of load instructions is not performed in the target loop.

scache_isolate_way specifier

end_scache_isolate_way specifier

scache_isolate_assign specifier

end_scache_isolate_assign specifier

See Section "[3.5 Software Control of Sector Cache](#)".

simd specifier

The simd specifier instructs to perform the optimization that uses SIMD Extensions. However, SIMD Extensions may not be used depending on the type of operation and the loop structure.

"simd", "simd aligned", or "simd unaligned" can be specified as the simd specifier, but they all have the same effect. The "aligned" and "unaligned" parameters are deprecated for this product and left for backward compatibility with previous products only.

The simd specifier cannot be specified for the "if-goto" loop.

Examples are shown in the following.

Example:

```
double a[10], b[10];
...
#pragma loop simd
```

```
for(i = 0; i < 10; i++) {
    a[i] = a[i] + b[i];
}
```

The `simd` specifier is in effect for the data in the loop.

The loop structure is modified to adjust boundaries for using SIMD Extensions.

`nosimd` specifier

The `nosimd` specifier instructs to suppress the optimization that uses SIMD Extensions.

An example is shown in the following.

Example:

```
#pragma loop nosimd
for(i = 0; i < 10; i++) {
    a[i] = a[i] + b[i];
}
```

The data operation is not the subject of SIMD Extensions in the loop.

`simd_listv` specifier

The `simd_listv` specifier instructs to perform list vector conversion to the statements in the block of "if" statement.

When `then` is specified with `simd_listv`, list vector conversion is performed to the first substatement of the specified "if" statement. When `else` is specified with `simd_listv`, list vector conversion is performed to the second substatement of the specified "if" statement.

When `all` or `nothing` is specified with `simd_listv`, list vector conversion is performed to the first and second substatements of the specified "if" statement.

When the `-O2` option or higher is set, and the `-Ksimd=2` option or the same effect by the `simd` specifier is effective, this specifier is effective.

For details about list vector conversion, see Section "[3.2.7.3 List Vector Conversion](#)".

Examples are shown in the followings.

Example 1:

```
#pragma loop simd
for(i = 0; i < n; i++ ) {
    #pragma statement simd_listv then
    if(a[i] == 0.0) {
        a[i] = a[i] + b[i];
    } else {
        a[i] = a[i] + c[i];
    }
}
```

The list vector conversion is performed to the first substatement of the specified "if" statement.

Example 2:

```
#pragma loop simd
for(i = 0; i < n; i++ ) {
    #pragma statement simd_listv else
    if(a[i] == 0.0) {
        a[i] = a[i] + b[i];
    } else {
        a[i] = a[i] + c[i];
    }
}
```

The list vector conversion is performed to the second substatement of the specified "if" statement.

Example 3:

```
#pragma loop simd
for(i = 0; i < n; i++ ) {
#pragma statement simd_listv all
    if(a[i] == 0.0) {
        a[i] = a[i] + b[i];
    } else {
        a[i] = a[i] + c[i];
    }
}
```

The list vector conversion is performed to the both substatements of the specified "if" statement.

simd_noredundant_vl specifier

The `simd_noredundant_vl` specifier instructs to invalidate SIMD with redundant executions for the SIMD width. For details about SIMD with redundant executions for the SIMD width, see section ["3.2.7.4 SIMD with Redundant Executions for the SIMD Width"](#).

Examples are shown in the followings. This specifier directs only specific loop to suppress SIMD with redundant executions for the SIMD width as follows.

Example:

```
void foo(double *a, double *b, double *c) {
    ...
#pragma loop simd_noredundant_vl
    for(i=0; i<m; i++) {
        a[i] = b[i]+c[i];
    }
    ...
}
```

simd_use_multiple_structures specifier

The `simd_use_multiple_structures` specifier instructs to use the SVE Load Multiple Structures instructions and the Store Multiple Structures instructions when using SIMD extensions.

This specifier is effective when the `-Ksimd[={1|2|auto}]` option or the `simd` specifier is effective, and the `-KSVE` option is set.

An example is shown in the following.

Example:

```
#include <complex.h>
#define N 10000
float _Complex a[N];
float _Complex b[N];
float _Complex c[N];
double y[N];
double x[N][4];
...
#pragma loop simd_use_multiple_structures
for(int i=0; i<N; ++i) {
    a[i] = b[i] * c[i];
}
...
#pragma loop simd_use_multiple_structures
for(int i=0; i<N; ++i) {
    y[i] = x[i][0] + x[i][1] + x[i][2] + x[i][3];
}
```

The SVE Load Multiple Structures instruction and the Store Multiple Structures instruction are used in this loop.

simd_nouse_multiple_structures specifier

The `simd_nouse_multiple_structures` specifier instructs not to use the SVE Load Multiple Structures instructions and the Store Multiple Structures instructions when using SIMD extensions.

An example is shown in the following.

Example:

```
#include <complex.h>
#define N 10000
float _Complex a[N];
float _Complex b[N];
float _Complex c[N];
double y[N];
double x[N][4];
...
#pragma loop simd_nouse_multiple_structures
for(int i=0; i<N; ++i) {
    a[i] = b[i] * c[i];
}
...
#pragma loop simd_nouse_multiple_structures
for(int i=0; i<N; ++i) {
    y[i] = x[i][0] + x[i][1] + x[i][2] + x[i][3];
}
```

The SVE Load Multiple Structures instruction and the Store Multiple Structures instruction are not used in this loop.

striping specifier

The `striping` specifier directs that the loop striping optimization is performed. The number from 2 to 100 following the specifier specifies stripe length (number of expansions).

If the stripe length is omitted, 2 is applied.

When the iteration count of a loop in the source program is known, the expansion number determined by the compiler will be used if a number that exceeds the iteration count is specified for stripe length.

Examples are shown in the following.

Example 1:

```
#pragma loop striping
for(i = 0; i < n; i++) {
    statement;
}
```

Loop striping is to be performed on the *statement* using the striping length determined by the compiler.

Example 2:

```
#pragma loop striping 4
for(i = 0; i < n; i++) {
    statement;
}
```

Loop striping is to be performed on the *statement* using the striping length 4.

Example 3:

```
#pragma loop striping 8
for(i = 0; i < 4; i++) {
    statement;
}
```

The specified stripe length (8) exceeds the loop iteration count (4), so the stripe length automatically determined by the compiler will be used.

Example 4:

```
void func() {  
#pragma procedure striping 8  
...  
    for(i = 0; i < n; i++) {  
        statement;  
    }  
...  
    for(j = 0; j < m; j++) {  
        statement;  
    }  
}
```

All the "for" loops in the program are targeted of loop striping.

nostriping specifier

The nostriping specifier directs that the loop striping optimization is suppressed.

An example is shown in the following.

Example:

```
#pragma loop nostriping  
for(i = 0; i < n; i++) {  
    statement;  
}
```

Loop striping is suppressed.

swp specifier

The swp specifier instructs the compiler to perform software pipelining.

Note that the swp specifier is effective only if the -O2 option or higher is set.

An example is shown in the following.

Example:

```
#pragma loop swp  
for(i = 0; i < n; i++) {  
    ...  
}
```

Software pipelining is performed to the specified loop.

noswp specifier

The noswp specifier instructs the compiler to suppress software pipelining.

An example is shown in the following.

Example:

```
#pragma loop noswp  
for(i = 0; i < n; i++) {  
    ...  
}
```

Software pipelining is suppressed in the target loop.

swp_freq_rate specifier

swp_ireg_rate specifier

swp_preg_rate specifier

Specifies the rate (percentage) about the following registers that can be used by software pipelining.

- Floating-point register and SVE vector register

- Integer register
- SVE predicate register

When specifying to `swp_freg_rate` specifier, the rate about floating-point register and SVE vector register is adjusted.

When specifying to `swp_ireg_rate` specifier, the rate about integer register is adjusted.

When specifying to `swp_preg_rate` specifier, the rate about SVE predicate register is adjusted.

The number from 1 to 1000 following specifiers instructs the rate (percentage) of the number of registers which is assumed to be usable by the software pipelining. This specifier is effective only if the `-O2` option or higher is set.

An example is shown in the following.

Example:

```
#pragma loop swp_freg_rate 120
#pragma loop swp_ireg_rate 150
#pragma loop swp_preg_rate 80
for(i = 0; i < n; i++) {
    ...
}
```

Perform the optimization of the software pipelining which adjusts the condition about the number of each kind of register for the target loop.

`swp_policy` specifier

Specifies a policy to select an instruction scheduling algorithm used in software pipelining.

Either of specifiers shown below is instructed after `swp_policy` specifier.

`auto`

An algorithm fit for a small loop, such as a loop with low register pressure, is used.

`small`

An algorithm fit for a small loop, such as a loop with low register pressure, is used.

`large`

An algorithm fit for a large loop, such as a loop with high register pressure, is used.

`swp_weak` specifier

The `swp_weak` specifier instructs the compiler to adjust software pipelining to make smaller overlapping of the instructions. This specifier is effective only if the `-O2` option or higher is set.

An example is shown in the following.

Example:

```
#pragma loop swp_weak
for(i = 0; i < n; i++) {
    ...
}
```

Perform the optimization of the software pipelining which adjust software pipelining for the target loop and make smaller overlapping of the instructions.

`unroll` specifier

The `unroll` specifier instructs the compiler to perform the optimization of loop unrolling for the corresponding loop.

The number from 2 to 100 following the specifier instructs the upper bound on the number of loops to be unrolled. If the upper bound of the number is omitted, the compiler automatically determines a suitable value.

When the `unroll "full"` specifier is used, the statement is unrolled up to the number of the iteration in the specified `"for"` statement. If the number of the iteration is unknown at compilation, no optimization of loop unrolling is performed.

Note that the `unroll` specifier targets only the loop specified immediately after.

An example is shown in the following.

Example:

```
#pragma loop unroll 8
for(i = 0; i < n; i++) {
    statement;
}
```

The "*statement*" is unrolled 8 times in the target loop.

nounroll specifier

The `nounroll` specifier instructs the compiler to suppress unrolling for the corresponding loop.

Note that the `nounroll` specifier targets only the loop specified immediately after.

An example is shown in the following.

Example:

```
#pragma loop nounroll
for(i = 0; i < n; i++) {
    ...
}
```

Loop unrolling is suppressed in the target loop.

unroll_and_jam specifier

The `unroll_and_jam` specifier instructs the compiler to apply unroll-and-jam. However, the optimization is suppressed in the following case:

- Assuming that the optimization is not effective.
- Assuming that there is a data dependency over iterations of the loops.

The number from 2 to 100 following the specifier instructs the upper limit of the unrolling expansion number. If the upper limit is omitted, the compiler automatically determines a value.

This optimization is not applied to innermost loop.

This specifier is effective only if the `-O2` option or higher is set.

The following shows examples:

Example 1:

```
#pragma loop unroll_and_jam 2
for (i=0; i<128; i++) {
    for (j=0; j<128; j++) {
        a[i][j] = b[i][j] + b[i+1][j];
        ...
    }
}
```

->

```
for (i=0; i<128; i+=2) {
    for (j=0; j<128; j++) {
        a[i][j] = b[i][j] + b[i+1][j];
        a[i+1][j] = b[i+1][j] + b[i+2][j];
    }
    ...
}
```

Unroll-and-jam is applied to the loop whose control variable is *i*.

Example 2:

```
for (i=0; i<128; i++) {
    #pragma loop unroll_and_jam 2
    for (j=0; j<128; j++) {
        a[i][j] = b[i][j] + b[i+1][j];
        ...
    }
}
```

Unroll-and-jam is not applied because the specified loop is the innermost.

Example 3:

```
#pragma loop unroll_and_jam 2
for (i=0; i<128; i++) {
    for (j=0; j<127; j++) {
        a[i][j] = a[i-1][j+1] + b[i][j];
        ...
    }
}
```

Unroll-and-jam is not applied because an array "a" has a data dependency over iterations of the loops.

unroll_and_jam_force specifier

The `unroll_and_jam_force` specifier instructs the compiler to perform the unroll-and-jam optimization for the corresponding loop assuming that there is no data dependency over iterations of the loops.

The result of executions is not guaranteed if the specifier is used incorrectly. For details, see the `unroll_and_jam_force` specifier in Section "3.4.1.3 Notes for Optimization Control Specifiers".

The number from 2 to 100 following the specifier instructs the upper limit of the unrolling expansion number. If the upper limit is omitted, the compiler automatically determines a value.

Note that the `unroll_and_jam_force` specifier targets only the loop immediately after the optimization control line. If the loop is the innermost, the unroll-and-jam is not applied.

This specifier is effective only if the `-O2` option or higher is set.

The following shows examples:

Example 1:

```
#pragma loop unroll_and_jam_force 2
for (i=0; i<128; i++) {
    for (j=0; j<128; j++) {
        a[i][j] = b[i][j] + b[i+1][j];
        ...
    }
}
```

->

```
for (i=0; i<128; i+=2) {
    for (j=0; j<128; j++) {
        a[i][j] = b[i][j] + b[i+1][j];
        a[i+1][j] = b[i+1][j] + b[i+2][j];
        ...
    }
}
```

Unroll-and-jam is applied to the loop whose control variable is `i`.

Example 2:

```
for (i=0; i<128; i++) {
    #pragma loop unroll_and_jam_force 2
    for (j=0; j<128; j++) {
        a[i][j] = b[i][j] + b[i+1][j];
        ...
    }
}
```

Unroll-and-jam is not applied because the specified loop is the innermost.

This specifier instructs that unroll-and-jam is performed assuming that there is no data dependency over iterations of the loops. The execution result is not guaranteed if the specifier is used incorrectly. See Section "3.4.1.3 Notes for Optimization Control Specifiers".

unswitching specifier

The `unswitching` specifier instructs loop unswitching to "if" statement.

This specifier must be described immediately before "if" statement which is invariant in a loop. This specifier is not effective in the case of specified other than the above.

The following shows an example:

Example 1:

```
void foo(double *a, double *b, double *c, int x, int n) {
    int i;
    for (i=0; i<n; i++) {
#pragma statement unswitching
        if (x == 0) {
            a[i] = b[i];
        } else {
            a[i] = c[i];
        }
    }
}
```

Loop unswitching to "if" statement is performed.

Note that "if" statement without this specifier is not the target of loop unswitching.

The following shows an example:

Example 2:

```
void foo(double *a, double *b, double *c, double *d, int x, int n) {
    int i;
    for (i=0; i<n; i++) {
        if (x == 0) {
            a[i] = b[i];
#pragma statement unswitching
        } else if (x == 1) {
            a[i] = c[i];
        } else {
            a[i] = d[i];
        }
    }
}
```

Only "if" statement with this specifier is the target of loop unswitching.

The memory consumption and the compilation time may increase drastically when loops to which loop unswitching is applied include a lot of execution statements.

zfill specifier

The `zfill` specifier directs to apply the `zfill` optimization. The `zfill` optimization speeds up write operations for array data that is only written in a loop, by using an instruction that allocates space on the cache for writing without loading data from the memory. The `zfill` optimization works on the data N blocks ahead of the address pointed to by the target store instruction where one block is 256 byte-long and N is an integer value between 1 and 100. If a value is not specified for N , the compiler will automatically determine a value.

For details about `zfill`, see "[3.3.5 zfill](#)".

Examples are shown in the following.

Example 1:

```
#pragma loop zfill
for(i = 0; i < n; i++) {
    ...
}
```

This specifies to improve the cache utilization for the data on certain blocks ahead, and the number is determined by the compiler.

Example 2:

```
#pragma loop zfill 1
for(i = 0; i < n; i++) {
    ...
}
```

This specifies to improve the cache utilization for the data on 1 block ahead.

nozfill specifier

The **nozfill** specifier directs not to apply the **zfill** optimization.

An example is shown in the following.

Example:

```
#pragma loop nozfill
for(i = 0; i < n; i++) {
    ...
}
```

This specifies that the **zfill** optimization will not be performed.

3.4.1.3 Notes for Optimization Control Specifiers

clone specifier

In the copied loops under the generated branches, the specified variables are treated as invariants in the loops. Thus, the execution result is not guaranteed in case that the value of the variables changes in the loop.

Example 1:

Do not specify a variable which is changed in the loop because the execution result is not guaranteed.

```
int *M = &N;
#pragma loop clone N==10
for(i=0;i<32;++i) {
    *M = 5;
    a[i] = N;
}
```

Example 2:

Do not specify a variable which may be changed in the loop because the execution result is not guaranteed.

```
#pragma loop clone N==10
for(i=0;i<32;++i) {
    sub(&N);
}
```

novrec specifier

If the **novrec** specifier is specified, the compiler generates the SIMD instruction given that there is no array for recurrent operation in the loop.

However, if there are some arrays for recurrent operation in the loop, the compiler uses SIMD instructions, but the execution result will be unpredictable.

An example is shown in the following.

Example:

```
#pragma loop novrec
for(i = 0; i < 100; i++) {
    a[i] = a[i + m] + ...;    <- When array 'a' is recurrence data and novrec specifier is
                             effective, the compiler uses SIMD instructions.
}
```

This is illegal usage of **novrec** specifier.

unroll_and_jam_force specifier

The **unroll_and_jam_force** specifier instructs that unroll-and-jam is performed assuming that there is no data dependency over iterations of the loops. The result of executions is not guaranteed if the specifier is used incorrectly.

Example:

```
#pragma loop unroll_and_jam_force 2
for (i=1; i<128; i++) {
    for (j=0; j<127; j++) {
        a[i][j] = a[i-1][j+1] + b[i][j];
        ...
    }
}
```

Do not use the `unroll_and_jam_force` specifier because there exists data dependency of array `a` over iterations of the loops.

`iterations max=n1` specifier

`iterations min=n3` specifier

The `iterations max=n1` and `iterations min=n3` specifiers are effective when the loop iteration count is unknown at compilation, and optimize assuming the maximum loop iteration count is $n1$ and the minimum loop iteration count is $n3$.

The result of executions is not guaranteed if the actual loop iteration count is greater than iteration count $n1$, or less than iteration count $n3$.

Example:

```
#pragma iterations max=100 min=1
for (i=1; i<m; i++) { // The actual maximum count is 1000 or minimum count is 0.
    a[i] = a[i] + b[i];
}
```

When the actual maximum iteration loop count is greater than 100, or minimum iteration loop count is 0, the execution result is not guaranteed.

3.5 Software Control of Sector Cache

3.5.1 Use of Sector Cache

Sector cache is a mechanism to prevent reusable data on the cache from being driven out by non-reusable data. In particular, Sector cache is effective for parallel execution when the shared L2 cache is accessed by multiple cores. By using Sector cache, reusable data is separated from non-reusable data on the cache. There are two methods to control Sector caches with software; environment variables or optimization control lines (OCL). The environment variables can specify the maximum number of ways for each sector, while the OCL can specify that of sector 1.

Performance will not be improved if the maximum number of ways is specified without consideration for the size of the array that is to be protected from being driven out. The cache utilization will decline, and may cause a reduction in speed. Further, even if software control of the Sector cache is not used, cache control with LRU (Least Recently Used) still works, so speed may not improve even if this is specified.

One effective way to improve the performance of Sector caches is to determine the reusable array size and specify the number of ways for sector 1 to store the array, after ensuring L2 cache miss has occurred by confirming the Hardware Monitor Information. Note that in order to use Sector caches, the compile-time option `-Khpctag` must be set.



Information

The maximum number of cache way for each sector

In the A64FX processor, the maximum number of the first level cache is 4, and that of the second level cache is 16.

Note that the assistant core always uses two ways for the second level cache.

3.5.2 Controlling Sector Cache via Software

The following two methods are provided to control Sector cache via software.

- Optimization Control Line (OCL).

- Environment variables and Optimization Control Line.

3.5.2.1 Software Control with Optimization Control Lines

The following OCLs exist to control Sector cache.

```
scache_isolate_way L2=n1 [L1=n2]
and
end_scache_isolate_way
```

The `scache_isolate_way` specifier directs the maximum number of ways in sector 1 of the caches.

The argument of the `scache_isolate_way` specifier directs the maximum number of ways of sector 1 of the cache. The `L2=n1` directs the maximum number of ways of sector 1 in the second level cache and the `L1=n2` directs the maximum number of ways of sector 1 in the first level cache. Specifying the `L1=n2` is optional. In this case, this specifier controls the maximum number of ways of sector 1 of the second level cache.

Note that the assistant core always uses two ways for the second level cache. With that in mind, the arguments `n1` and `n2` must be as follows;

- `0 <= n1 <= "the maximum number of ways in the second level cache - 2"`
- `0 <= n2 <= "the maximum number of ways in the first level cache"`

To direct in function, the `scache_isolate_way` specifier must be used in procedure line. In this case, the `scache_isolate_way` specifier is effective in the scope of the function.

To direct a part of function, the range must be indicated by the `scache_isolate_way` and `end_scache_isolate_way` specifiers in statement lines.

Note that the range must not be nested. However, combined use of procedure line and statement line is accepted.

```
scache_isolate_assign array1[,array2]...
and
end_scache_isolate_assign
```

The `scache_isolate_assign` specifier directs the array and pointer to the array to store in sector 1 of cache. However, it becomes effective only when the array of arithmetic type or pointer is specified.

To direct in function, the `scache_isolate_assign` specifier must be used in procedure line. In this case, the `scache_isolate_assign` specifier is effective in the scope of the function.

To direct a part of function, the range must be indicated by the `scache_isolate_assign` and `end_scache_isolate_assign` specifiers in statement lines.

Note that the range must not be nested. However, combined use of procedure line and statement line is accepted.



Example

Example of software method to control Sector cache

```
/* Reuse the array a that can fit in 10 ways in the L2 cache */
#pragma statement scache_isolate_way L2=10
#pragma statement scache_isolate_assign a
for(int j = 0; j < n; j++) {
#pragma omp parallel for
    for(int i = 0; i < m; i++) {
        a[i] = a[i] + b[j][i];
    }
}
#pragma statement end_scache_isolate_assign
#pragma statement end_scache_isolate_way
```


3.5.2.2 Software Control with Environment Variables and Optimization Control Line

The initial value for the maximum number of ways for each Sector can also be specified with the following environment variables. The maximum number of ways for the object program at startup can be specified with environment variables.

FLIB_SCCR_CNTL

This environment variable specifies to use Sector cache. Allowed values for the variable are shown below.

Value	Explanation
TRUE	Use Sector cache. The default is TRUE.
FALSE	Do not use Sector cache.

FLIB_L1_SCCR_CNTL

The Sector cache for the second level cache is not available when multiple processes run on a NUMA node. This environment variable specifies whether to use the Sector cache for the first level cache when the Sector cache for the second level cache is unavailable. FLIB_L1_SCCR_CNRTL is effective only when FLIB_SCCR_CNTL is assigned to the value TRUE.

The values that can be set in the environment variable are shown below, along with their meanings.

Value	Explanation
TRUE	If the Sector cache for the second level cache is not available, the Sector cache for the first level cache is used. The value is set by default.
FALSE	If the Sector cache for the second level cache is not available, the Sector cache for the first level cache is not used. And the following error message is output, the Sector cache control function is disabled, and program execution continues. <div>jwe1047i-w A sector cache couldn't be used.</div>

FLIB_L2_SECTOR_NWAYS_INIT

This environment variable specifies the initial maximum number of cache ways for each sector of the second level cache. FLIB_L2_SECTOR_NWAYS_INIT is effective only when the value of FLIB_SCCR_CNTL is TRUE.

The values for the first sector (sector 0) *n0* and for the second sector (sector 1) *n1* should be specified as follows:

n0 , *n1*

The possible values are as follows:

- $0 \leq n0 \leq$ "the maximum number of ways in the second level cache - 2"
- $0 \leq n1 \leq$ "the maximum number of ways in the second level cache - 2"

The two ways are reserved for the assistant cores.

It is recommended to satisfy the following condition to avoid conflicts.

- $n0 + n1 =$ "the maximum number of ways in the second level cache - 2"



Example

Example of software method to control Sector cache with environment variables and optimization control line

As shown in the following example of software control of Sector cache, by setting the value "2,10" to the environment variable FLIB_L2_SECTOR_NWAYS_INIT before execution starts, the maximum number of ways is set to 10 and the array data is cached on the sector 1 cache.

1. Set environment variables (bash)

```
FLIB_SCCR_CNTL=TRUE
export FLIB_SCCR_CNTL
FLIB_L2_SECTOR_NWAYS_INIT=2,10
export FLIB_L2_SECTOR_NWAYS_INIT
```

2. Specify an array data stored in sector 1 by optimization control lines

```
/* Assign sector 1 to the array a */
#pragma statement scache_isolate_assign a
for(int j = 0; j < n; j++) {
#pragma omp parallel for
    for(int i = 0; i < m; i++) {
        a[i] = a[i] + b[j][i];
    }
}
#pragma statement end_scache_isolate_assign
```

3.5.2.3 Behavior when an Exceptional Value Is Specified

If a value specified for the `scache_isolate_way` specifier or the environment variable `FLIB_L2_SECTOR_NWAYS_INIT` exceeds the upper limit, it is assumed that the upper limit is specified. If a value less than 0 is specified, it has no effect. If a value outside the range of two-byte signed integer is specified, the behavior is undefined.

3.6 Notes


3.6.1 Side Effect of Optimizations for Floating-Point Operation


Optimizations for floating-point operation might cause the side effect. This section explains the side effect (mainly, computation error).

This system basically creates objects which comply with IEEE 754 arithmetic. Note that the numerical operations may not comply with IEEE 754 arithmetic due to the optimizations with calculation errors in "Table 3.4 Side effect of optimization for floating-point operation".

See Section "2.2.2.5 -K Option" for compiler options. See section "3.4.1.2 Optimization Control Specifier" and "4.2.6.3 Optimization Control Specifiers for Automatic Parallelization" for optimization control specifiers.

Table 3.4 Side effect of optimization for floating-point operation

Compiler Option	Optimization Control Specifier	Side Effect
-Keval	eval	<p>Calculation errors may occur when optimization that changes the method of operator evaluation.</p> <p> Example</p> <hr/> <p>$x*y + x*z \rightarrow x*(y+z)$</p> <hr/> <p>When the -Keval option is set, the following options are valid. See the description about the side effect of each option.</p> <ul style="list-style-type: none"> -Kfsimple option -Kreduction option (When -Kparallel option is valid) -Ksimd_reduction_product option (When -Ksimd[={ 1 2 auto}] option is valid)

Compiler Option	Optimization Control Specifier	Side Effect
-Kfsimple	-	<p>Simplification of floating point operation on source programs is performed. Therefore, the numerical operations do not comply with IEEE 754 arithmetic, as in the example below.</p> <p> Example</p> <hr/> <p><code>x*0.0 -> 0.0</code></p> <p>Operations such as "x*0.0" will be simplified to "0.0".</p>
-Kfp_contract	fp_contract	Rounding errors may occur when optimization using Floating-Point Multiply-Add/Subtract instructions is performed on source programs.
-Kfp_relaxed	fp_relaxed	<p>Side effects may occur because reciprocal approximation operation instructions are used on single-precision or double-precision floating point division or <code>sqr t</code> functions.</p> <p>The side effects that may occur are:</p> <ul style="list-style-type: none"> - Rounding errors. - Replacing denormalized numbers found in the arguments or the return value with zero regardless of set of -Kfz option. - Replacing negative zeroes found in the arguments or the return value with positive zeroes. - Behaviors not conforming to IEEE 754 when NaN, positive or negative Inf, numbers which are close to maximal normalized number or numbers which are close to minimal normalized number are found in the arguments or the return value.
-Kfz	-	Flush-to-zero mode is used. If a result or a source operand is a denormalized number, flush-to-zero mode replaces it with zero with the same sign.
-Kfast_matmul	-	Calculation errors may occur when using high speed library call for the loop of matrix multiplication.
-Kilfunc	-	Side effects similar to the ones caused by -Kfp_relaxed may occur when using inline expansion for the math functions, because reciprocal approximation instructions and trigonometric instructions, etc. are used. Plus, use of reciprocal approximation instructions or Floating-Point Multiply-Add/Subtract instructions regardless of set of -Knofp_contract and/or -Knofp_relaxed options or their setting order.
-Kmfunc[={ 1 2 3}]	mfunc [<i>level</i>]	<p>Side effects similar to the ones caused by -Kfp_relaxed and/or -Kilfunc may occur because different algorithms, reciprocal approximation instructions, trigonometric instructions, etc. are used internally when the function is converted to the multi-operation function.</p> <p>See Section "3.3.3 Multi-Operation Function".</p>
-Kreduction	reduction	The automatic parallelization loop reduction is performed. Therefore, calculation errors may occur when the optimization that changes the method of operator evaluation.
-Ksimd_reduction_product	-	SIMD extensions are used to the reduction operation of product. Therefore, calculation errors may occur when the optimization that changes the method of operator evaluation.
-Kfast	-	-Kfast option induces the -Keval option, -Kilfunc option, and so on.
-Kvisimpact	-	-Kvisimpact option induces the -Kfast option.

-: None

If the -O1 option or higher is set, this system performs optimizations that execute floating-point arithmetic instructions that may not to be executed based on the logic of the program. This may result in generating extra floating-point exceptions at execution time. In this case, the following phenomena may occur.

- If a program uses the `fegetexcept` function included in standard library functions and accesses a floating-point status flag by it, the flag which should not be set based on the logic of the program is set.
- If a program uses the `feenableexcept` function included in the GNU C Library and enables a trap of a floating-point exception by it, the SIGFPE signal which should not be raised based on the logic of the program is raised.

This phenomenon may be avoided by setting the option -NRtrap.



Information

To avoid side effect for floating-point operations error at parallelization (automatic parallelization/OpenMP)

Specify the following options to avoid calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads. When these options are set, the execution performance may decrease.

- -Kparallel_fp_precision option (When the -Kparallel option or -Kopenmp option is valid.)
- -Kopenmp_ordered_reduction option (When the -Kopenmp option is valid.)

See Section "2.2.2.5 -K Option" for compiler options.

3.6.2 Notes on Specified SVE Vector Register Size

-Ksimd_reg_size={ 128|256|512 } specifies the bitwise SVE vector register size. When this option is specified, the optimization is performed considering the vector register size as a fixed value at compilation. Therefore, the generated executable program works normally on the CPU architecture which has the same size of the SVE vector register as the specified size.

When the program is executed on the CPU whose implemented vector register size is different from the vector register size specified by -Ksimd_reg_size={ 128|256|512 } option, an abnormal end occurs. And, the result of executions is not guaranteed.

It is required to change the effective size of the vector register by the system call `prctl (2)` or in other ways when the vector register size specified by -Ksimd_reg_size={ 128|256|512 } option is smaller than that of the CPU.

When -Ksimd_reg_size=agnostic is effective, the executable program does not depend on the SVE vector register size.

See also Section "C.2.8 Changing SVE Vector Register Size" for notes on changing the SVE vector register size.

3.6.3 Notes of Wrong Erroneous Program

When optimization is performed on the following erroneous programs, execution results may differ with the optimization level. If results differ, check the following and correct the programs.

- Variables containing undefined values are referenced
- Variables containing values with unacceptable formats are referenced
- Array elements are referenced using index values that exceed array declarations
- There are violations of C language syntax

Chapter 4 Multiprocessing

This chapter describes the method of processing a C program in parallel or "multiprocessing" using LLVM OpenMP Library.

See "[Appendix J Fujitsu OpenMP Library](#)" when you use Fujitsu OpenMP Library.

Information

The table below shows a connectable combination of library for multiprocessing (LLVM OpenMP Library and Fujitsu OpenMP Library) and connectable object file.

	Fortran object file	C/C++ object file	
		Trad Mode (-Nnoclang option)	Clang Mode (-Nclang option)
LLVM OpenMP Library (-Nlibomp option)	Available	Available	Available
Fujitsu OpenMP Library (-Nfjompilib option)	Available	Available	Not available

4.1 Overview of Multiprocessing

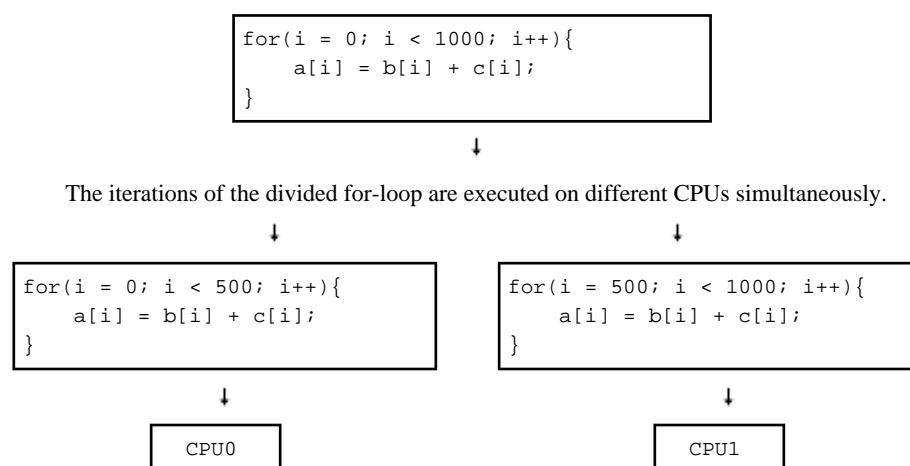
This section describes multiprocessing of a C program and parallelization using LLVM OpenMP Library.

4.1.1 What is Multiprocessing?

In this document, multiprocessing means that one program is executed on two or more CPUs that can work independently at the same time. As used here, it does not mean executing two or more programs simultaneously.

"[Figure 4.1 Multiprocessing](#)" illustrates multiprocessing by using two CPUs simultaneously.

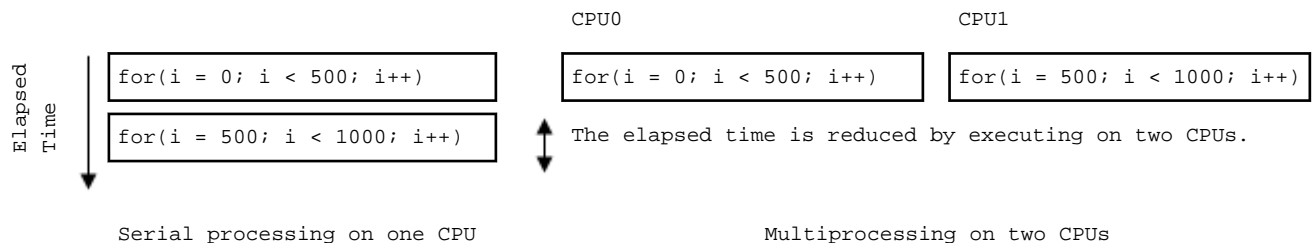
Figure 4.1 Multiprocessing



4.1.2 Effect of Multiprocessing

The effect of multiprocessing is to reduce elapsed execution time by using two or more CPUs simultaneously. For instance, if a `for` loop can be executed in parallel by dividing it as shown in "Figure 4.1 Multiprocessing", the execution time of this `for` loop may be cut in half (See "Figure 4.2 Reducing elapsed time using multiprocessing").

Figure 4.2 Reducing elapsed time using multiprocessing



Although the elapsed time will usually decrease with multiprocessing, the total CPU time required to execute the program may increase. This is because the total CPU time is at least as large as the CPU time when the program is executed on a single processor, and the overhead time for multiprocessing may increase the total CPU time.

The performance of a multiprocessing program is usually evaluated by the reduction of the elapsed time. The total CPU time of a program using multiprocessing is larger than that required by serial processing. This is because of the overhead time required by multiprocessing.

4.1.3 Requirements for Effective Multiprocessing

A computer environment that can use two or more CPUs simultaneously is necessary to reduce elapsed time by multiprocessing. A multiprocessing program can be executed on hardware with only a single CPU; however, the elapsed time will not be less than the execution time for a comparable program written without multiprocessing features. Moreover, even if the program is executed on hardware with two or more CPUs, shortening the elapsed time is difficult when other jobs are executing on the same CPUs. The reason for this is that the probability that two or more CPUs can be allocated at the same time decreases.

To achieve effective multiprocessing, it is necessary to run a multiprocessing program on hardware with multiple CPUs and within a system environment that has room for multiprocessing.

To reduce the relative overhead of multiprocessing, it is necessary that a loop have many iterations and many statements in its body.

4.1.4 Parallelization

This system offers automatic parallelization. "`for`" loops, "`while`" loops, "`do-while`" loops and "`if-goto`" loops are parallelized without any modifications made to the program. Therefore, it is easy to migrate source programs to other processing systems as long as the program conforms to the C standard.

However, the automatic parallelization is not applied to the loops that the control jumps outside of the loop or jumps in from outside of the loop.

Moreover, this system offers the optimization control line (OCL), which helps automatic parallelization. The optimization control line is used by the programmer to identify constructs that may be executed in parallel.

For more information about automatic parallelization, see Section "4.2 Automatic Parallelization".

This system also offers parallelization by the OpenMP specification, see Section "4.3 Parallelization by OpenMP Specification" for details.

4.2 Automatic Parallelization

This section describes automatic parallelization.

4.2.1 Compilation

To activate automatic parallelization, specify the following options:

```
-Kparallel [-Nlibomp]
```

4.2.1.1 Compiler Option for Automatic Parallelization

The compiler option for automatic parallelization is:

```
-K{parallel|parallel_strong|visimpact}  
[,array_private,dynamic_iteration,instance=N,loop_part_parallel,ocl,optmsg=2,parallel_fp_precision,p  
arallel_iteration=N,reduction,region_extension] [-Nlibomp]
```

See Section ["2.2 Compiler Options"](#) for details on how to use each option.

4.2.2 Execution Process

When a multiprocessing program is executed, the number of threads to be used may be specified by the environment variable OMP_NUM_THREADS. The size of the thread stack may be specified by the environment variable OMP_STACKSIZE. The method of synchronizing threads can be changed using the environment variable OMP_WAIT_POLICY.

Except these environment variable settings, the procedure for execution is same as for a serial-processing program.

4.2.2.1 Number of Threads

The number of threads is determined with the following priority.

1. The value of the environment variable OMP_NUM_THREADS
2. The number of CPUs that can be used in the system

The following environment variable for OpenMP specifications can be used in automatic parallelization. See OpenMP specifications for the detail.

OMP_NUM_THREADS

Sets the number of threads to use during execution.

4.2.2.2 Stack Size on Execution

The stack area size for each thread can be specified using the following environment variable.

OMP_STACKSIZE

The stack area size for each thread can be specified using the environment variable OMP_STACKSIZE in byte, Kbytes, Mbytes, Gbytes, or Tbytes. The default size is 8M bytes.

4.2.2.3 Synchronization Process

The synchronization process can be controlled with the environment variable OMP_WAIT_POLICY.

OMP_WAIT_POLICY

ACTIVE	Uses spin wait until all threads are synchronized.
PASSIVE	Uses no spin wait but suspending wait.

The default value is "PASSIVE".

Select ACTIVE to give priority to the elapsed time. Select PASSIVE to give priority to the CPU time.

4.2.2.4 CPU Binding for Thread

Environmental variable GOMP_CPU_AFFINITY or OMP_PROC_BIND can control the CPU binding of threads. GOMP_CPU_AFFINITY takes precedence over OMP_PROC_BIND.

The environment variable GOMP_CPU_AFFINITY

Threads are bound to CPUs in order of the specified cpuid list.

When the number of specified CPUs is exceeded, it is repeatedly used from the beginning of the list.

The cpuid list shall be separated by comma (',') or space (' ').

The cpuid list can have the next form that has range with increment.

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: cpuid of the beginning of the range. ($0 \leq \text{cpuid1} < \text{CPU_SETSIZE}$)

cpuid2: cpuid of the end of the range. ($0 \leq \text{cpuid2} < \text{CPU_SETSIZE}$)

inc: increment ($1 \leq \text{inc} < \text{CPU_SETSIZE}$)

In addition, it is necessary to be the following.

```
cpuid1 <= cpuid2
```

It becomes equivalent to the case where all CPUs for every increment value *inc* in the range from *cpuid1* to *cpuid2* are specified.

The cpuid can be used the above-mentioned value.

However, cpuid which can actually be assigned becomes only within the limits of CPU affinity of the process at the start of execution.

See CPU_SET(3) about details of CPU_SETSIZE.



Note

If cpuid is outside the CPU affinity of the process at the start of execution, an error will be output and the program will terminate. Correct the setting value.



Example

Examples of using environment variable GOMP_CPU_AFFINITY

- Example 1:

```
$ export GOMP_CPU_AFFINITY="12,14,13,15"
```

The thread is bound to CPU in order of 12, 14, 13, and 15.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

- Example 2:

```
$ export GOMP_CPU_AFFINITY="12-19"
```

The thread is bound to CPU in order of 12, 13, 14, 15, 16, 17, 18, and 19.

When the number of threads is nine or more, it is repeatedly used from the beginning of the list.

- Example 3:

```
$ export GOMP_CPU_AFFINITY="12-19:2"
```

The thread is bound to CPU in order of 12, 14, 16, and 18.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

- Example 4:

```
$ export GOMP_CPU_AFFINITY="12-16:2,13,19"
```

The thread is bound to CPU in order of 12, 14, 16, 13, and 19.

When the number of threads is six or more, it is repeatedly used from the beginning of the list.

.....

The environment variable OMP_PROC_BIND

The environment variable OMP_PROC_BIND can control the thread affinity. Either `true`, `false` or comma separated list of `master`, `close`, or `spread` can be specified to this environment variable. The values of the list sets the thread affinity policy used by a `parallel` region corresponding to the nest level.

This environment variable is set to `close` by default.

If this environment variable is set to `false`, thread affinity is disabled, and the `proc_bind` clause on `parallel` construct is ignored.

Otherwise, thread affinity is enabled and the initial thread is bound to first place in place list.

The `master` thread affinity policy indicates that all threads are bound to same place as master thread.

The `close` thread affinity policy indicates that threads are bound to next to the place where master thread is bound.

The `spread` thread affinity policy indicates that the threads are bound to the one of sub-partition places divided from master thread's place partition.

The effect of `true` is same as `spread`.

See environment variable OMP_PLACES for place.

The environment variable OMP_PLACES

The environment variable OMP_PLACES defines place list.

The explicit place list is defined by ordered set of comma separated non-negative numbers enclosed by braces. The numbers represents the smallest unit of CPU resource in the system.

The value of OMP_PLACES is either abstract name or an explicit place list. The value can be specified using following format.

```
{ abstract-name[(num-places)] | place-list }
```

abstract-name

The following abstract names can be set to this environment variable. The default value is `cores`.

Abstract Name	Meaning
<code>threads</code>	Each place corresponds to a single hardware thread, which represents smallest unit of CPU resource in the system. The effect of <code>threads</code> is equivalent to <code>cores</code> .
<code>cores</code>	Each place corresponds to a single hardware core, which represents smallest unit of CPU resource in the system. The effect of <code>cores</code> is equivalent to <code>threads</code> .
<code>sockets</code>	Each place corresponds to a single socket, which represents NUMA node.

num-places

The length of the place list. Positive integer.

place-list

The explicit place list can be specified using following format.

```
{ place:length[:stride] | [!]place[,place-list] } (*1)
```

place

"{"*placeI*" }" (*2)

place1

{ *cpuid.length[:stride]* | [!]*cpuid*[,*placeI*] }

length

The length of elements. Positive integer.

stride

The value of increment or decrement. Positive integer. Default value is 1.

cpuid

Smallest unit of CPU resource in the system.

*1) An exclusion operator "!" exclude the number or place immediately following the operator.

*2) "{" and "}" are braces.



Example

Example of environmental variable OMP_PLACES

- Example 1:

```
$ export OMP_PLACES="cores"
```

This example defines place list of cores abstract name.

- Example 2:

```
$ export OMP_PLACES="cores(4)"
```

This example defines place list of cores abstract name of 4 places.

- Example 3:

```
$ export OMP_PLACES="{12,13,14},{15,16,17},{18,19,20},{21,22,23}"
$ export OMP_PLACES="{12:3},{15:3},{18:3},{21:3}"
$ export OMP_PLACES="{12:3}:4:3"
```

All above examples define same place of 12 to 14, 15 to 17, 18 to 20, and 21 to 23.

4.2.3 Example of Compilation and Execution

Example 1:

```
$ fccpx -Kparallel,reduction,ocl -Nlibomp test1.c
$ ./a.out
```

In this example, reduction parallelization optimization and the optimization control lines (OCL) are in effect during compilation.

The number of threads using at the runtime is defined by the execution environment. For details about the number of threads, see Section ["4.2.2.1 Number of Threads"](#).

Example 2:

```
$ fccpx -Kparallel -Nlibomp test2.c
$ OMP_NUM_THREADS=2
$ export OMP_NUM_THREADS
$ ./a.out
$ OMP_NUM_THREADS=4
```

```
$ export OMP_NUM_THREADS
$ ./a.out
```

The environment variable OMP_NUM_THREADS is set to 2 and the program executes with two threads.

Next, the environment variable OMP_NUM_THREADS is set to 4 and the program executes with four threads.

4.2.4 Performance Tuning

Tuning information for application programs is gathered by the execution time sampling function provided by the Profiler. A programmer can use this information to identify high cost statements in the program. To execute a program at high speed, the programmer needs to tune the program so that the system is able to parallelize the high cost statements. Refer to the "Profiler User's Guide" for further details about the Profiler.

4.2.5 Feature Details on Automatic Parallelization

This section describes automatic parallelization.

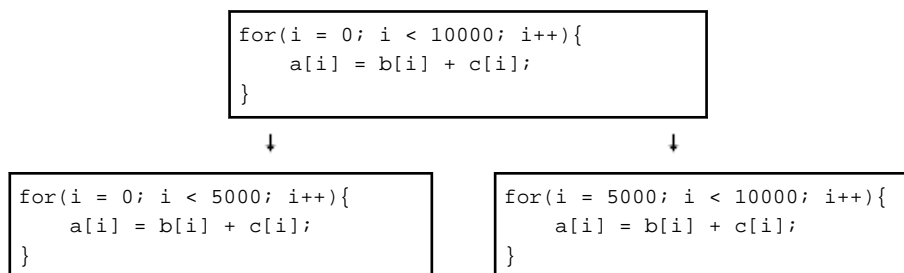
4.2.5.1 Targets for Automatic Parallelization

Target statements of automatic parallelization are loops (including nested loops) and pointer operations.

4.2.5.2 Loop Slicing

Automatic parallelization may slice a loop into several pieces. The elapsed execution time is reduced by executing the loop slices in parallel. "Figure 4.3 Loop slicing" illustrates loop slicing.

Figure 4.3 Loop slicing

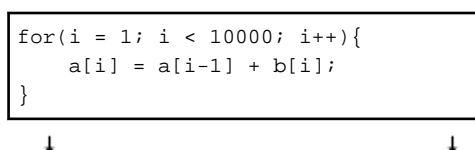


4.2.5.3 Automatic Loop Slicing by Compiler

The system parallelizes a loop if the order of data reference will be the same as with serial execution, and if the system is certain that the result of the parallel loop will be the same as if the loop were processed serially.

"Figure 4.4 A loop that is not a candidate for loop slicing" is an example of a loop that is not amenable to loop slicing. In this `for` loop, when the loop variable `i` is 5000, the value of array element `a[4999]` should be available on the corresponding thread. However, this value is not available and the loop will not be sliced.

Figure 4.4 A loop that is not a candidate for loop slicing



```
for(i = 1; i < 5000; i++){
    a[i] = a[i-1] + b[i];
}
```

```
for(i = 5000; i < 10000; i++){
    a[i] = a[i-1] + b[i];
}
```

4.2.5.4 Loop Interchange and Automatic Loop Slicing

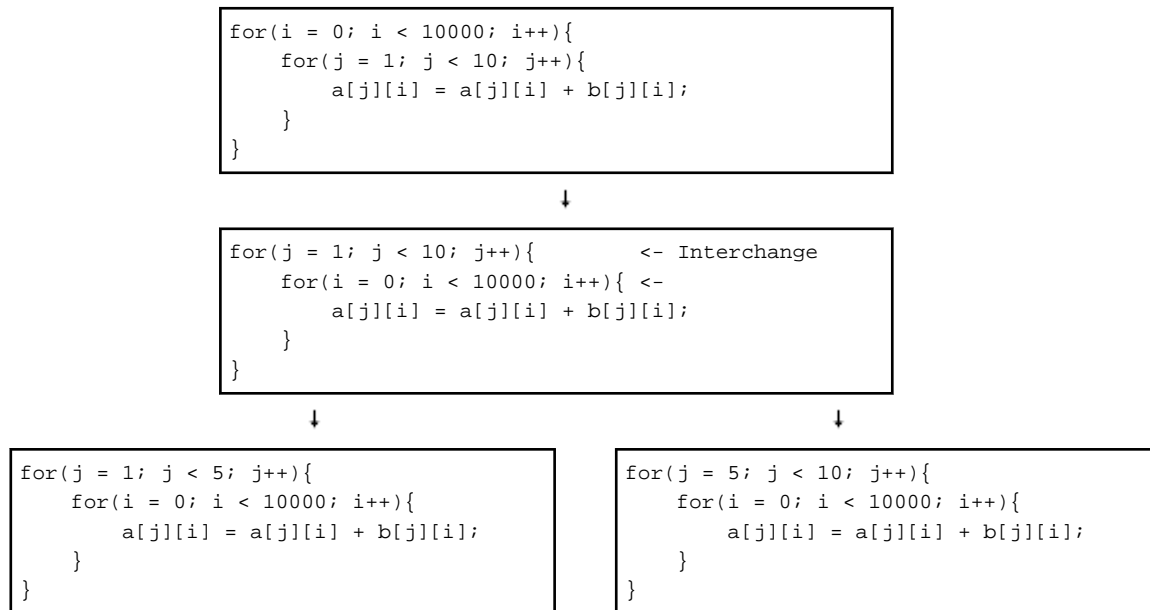
When nested loops are sliced, the system attempts to parallelize the outermost loop if it can. Therefore, the system selects the loop that can be sliced most cheaply and interchanges it with the outermost loop. The purpose of this is to reduce the overhead of multiprocessing and improve the execution performance.

Loop interchange optimization is performed if both `-Kparallel` and `-Keval` are set.

"Figure 4.5 Loop interchange on a nested "for" loop" illustrates the loop slicing after performing loop interchange on a nested "for" loop.

Before performing parallelization for the "for" loop variable `j`, loop interchange is performed so that the parallelization overhead is reduced.

Figure 4.5 Loop interchange on a nested "for" loop



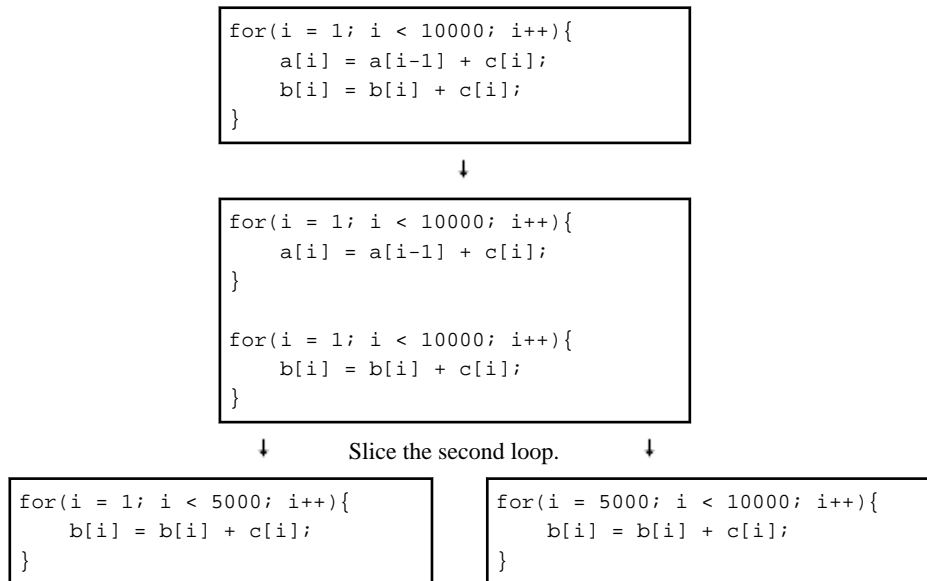
4.2.5.5 Loop Distribution and Automatic Loop Slicing

In "Figure 4.6 Distribution of loop and loop slicing", the references to array `a` cannot be sliced, because the order of data references in parallel execution would be different from the order of data references in serial execution. The references to array `b` can be sliced, because the order of parallel data references is the same as the order of serial data references. In this example, the statement where array `a` is defined and the statement where array `b` is defined are separated into two loops, and the loop where array `b` is defined is parallelized.

The partial automatic parallelization by the distribution of loop is performed when the `-Kloop_part_parallel` option is effective and the compiler assumes that the distribution of loop is effective.

"Figure 4.6 Distribution of loop and loop slicing" shows the example of loop separation and automatic loop slicing.

Figure 4.6 Distribution of loop and loop slicing



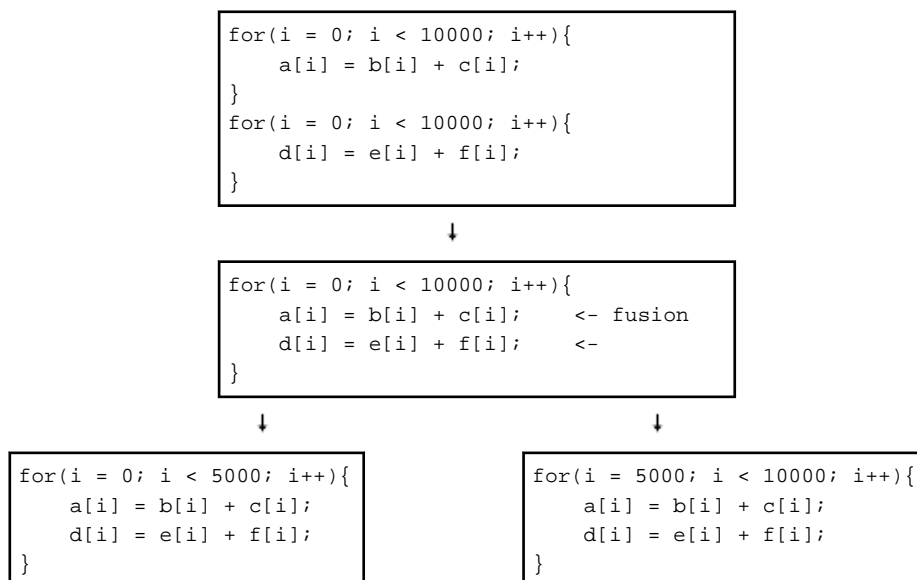
4.2.5.6 Loop Fusion and Automatic Loop Slicing

In "Figure 4.7 Fusion of loops and loop slicing", there are loops in sequence having the same loop control.

In this case, the overhead of the loop control and the overhead of multiprocessing can be reduced by merging those two loops into a single loop.

"Figure 4.7 Fusion of loops and loop slicing" shows the example of loop fusion and automatic loop slicing.

Figure 4.7 Fusion of loops and loop slicing



4.2.5.7 Loop Reduction

If both the -Kparallel and -Kreduction options are specified, loop reduction takes place. This optimization slices the loop, and changes the order of the reduction operation (addition, multiplication). Loop reduction may cause calculation errors in the execution results.

The loop reduction optimization is applied if there is one of the following operations in the loop:

- Sum

Example: `s = s + a[i];`

- Product

Example: `p = p * a[i];`

- Dot product

Example: `p = p + a[i] * b[i];`

- Min

Example: `x = min(x, a[i]);`

- Max

Example: `y = max(y, a[i]);`

- Bit or

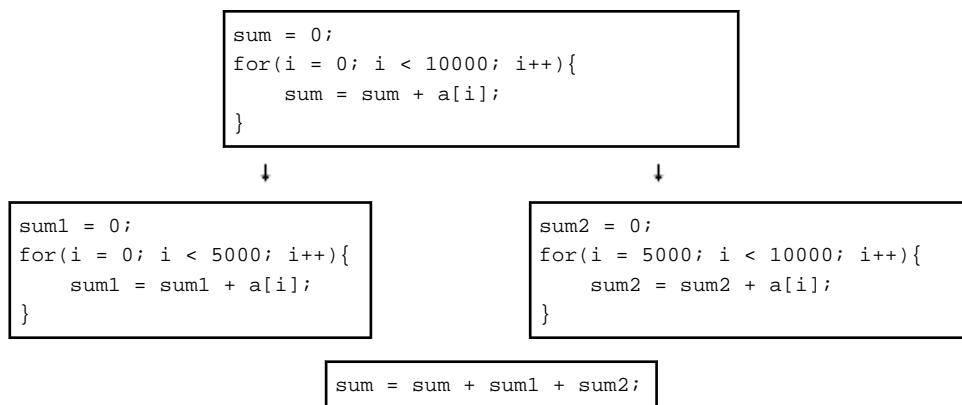
Example: `n = n | a[i];`

- Bit and

Example: `m = m & a[i];`

"Figure 4.8 Loop reduction" shows an example of loop reduction and automatic loop slicing.

Figure 4.8 Loop reduction



4.2.5.8 Restrictions on Loop Slicing

Loops cannot be sliced under the following circumstances.

- Loop slicing would not reduce the elapsed time.
- The loop contains a function reference.
- The loop is complicated.
- The loop contains a standard library function.
- The order of data references in parallel would be different from that of the data references in serial execution.

Loop slicing would not reduce the elapsed time.

If the loop iteration count is small or the number of operations in the loop is small, execution performance of the multiprocessing program may decrease due to the overhead of parallel execution. Therefore, the system does not slice loops when it is clear that the performance will not improve.

"[Figure 4.9 Small loop iteration count and small number of operations](#)" shows an example of a loop for which the loop iteration count and the number of operations is small.

Figure 4.9 Small loop iteration count and small number of operations

```
for(i = 0; i < 10; i++){      <- Not parallelized because of small loop iteration count and
    a[i] = a[i] + b[i];      small number of operations
}
```

The loop contains a function reference.

A loop containing a function reference cannot be sliced. However, parallelization can be invoked by an optimization control line (OCL). See Section "[4.2.6 Optimization Control Line](#)" for details.

"[Figure 4.10 Loop contains a function call](#)" shows an example of a loop that contains a function call.

Figure 4.10 Loop contains a function call

```
for(j = 0; j < 10; j++){
    for(i = 0; i < 10000; i++){      <- Not parallelized because of the function call
        a[j][i] = a[j][i] + b[j][i];
        func(a);
    }
}
```

The loop is complicated

The following loops cannot be sliced because they are too complicated:

- There is a branch from inside the loop to outside the loop.

"[Figure 4.11 Jumping out from inside the loop](#)" shows an example of jumping from the inside to the outside of a loop.

Figure 4.11 Jumping out from inside the loop

```
for(j = 0; j < 10; j++){
    for(i = 0; i < 10000; i++){      <- Not parallelized because of jumping of the loop
        a[j][i] = a[j][i] + b[j][i];
        if(a[j][i]<0) goto out;
    }
}
out;
```

The loop contains a standard library function.

Some standard library functions do not prevent loop slicing. The compiler produces a diagnostic message during compilation if a loop is not parallelized because of a standard library function reference.

The order of data references in parallel would be different from that of the data references in serial execution.

If the order of data references would be different from that of serial execution, the loop is not suitable for loop slicing as shown in the example of "[Figure 4.4 A loop that is not a candidate for loop slicing](#)".

4.2.5.9 Displaying the State of Automatic Parallelization

Specify both the `-Kparallel` and the `-Koptmsg=2` options to check whether automatic parallelization has been performed, and also to confirm what has been parallelized or learn what has prevented parallelization if it has not been performed.

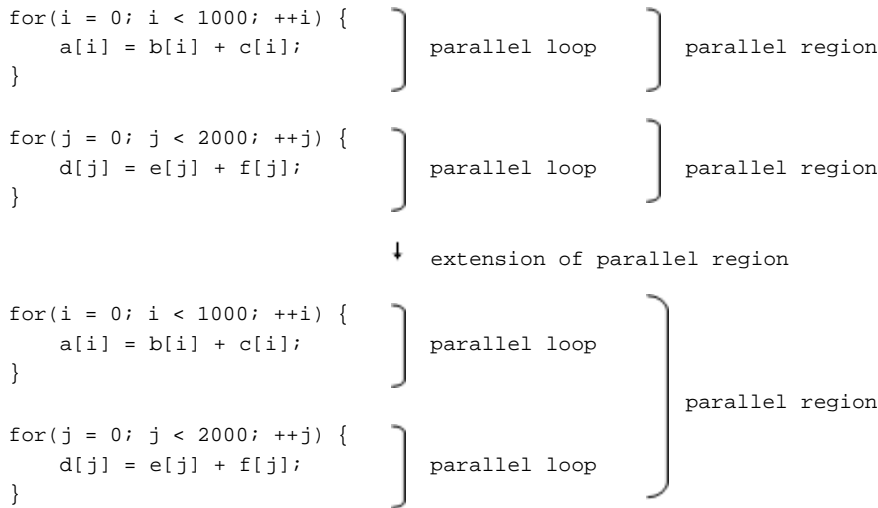
4.2.5.10 Parallel Region Extension

When the `-Kregion_extension` and `-Kparallel` options are effective, parallelization overhead may be reduced by extending the parallel region.

The parallel region is one executed by a team consisting of more than one thread. Usually, one parallelization loop is generated for one parallel region as in "[Figure 4.12 Example of parallel region Extension](#)". The cost of generating the parallel region is the overhead of parallel execution.

By applying a parallel region extension, these parallelization loops are generated in one parallel region. As a result, the parallel region is generated only once, and the cost of the overhead is reduced.

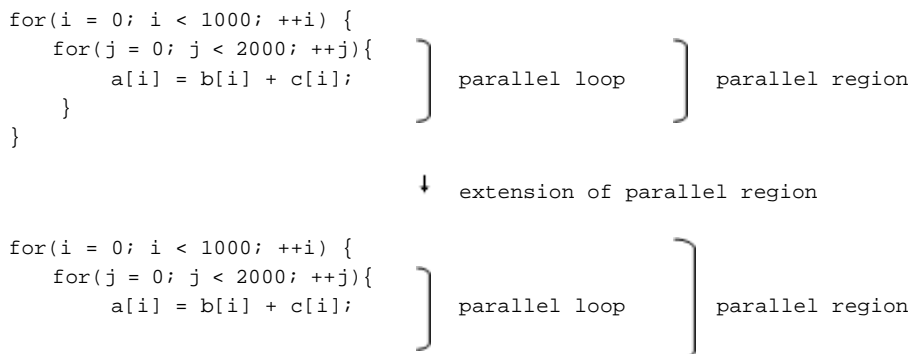
Figure 4.12 Example of parallel region Extension



The parallel region extension is also effective in reducing the overhead of the parallel region at the nested loops as in "[Figure 4.13 Example of extension of parallel region \(nested loop\)](#)".

In this case, the number of times the parallel region is generated is reduced from 1000 to 1 because the generating point is moved from the top of the inner loop to the top of the outer loop.

Figure 4.13 Example of extension of parallel region (nested loop)




```

    }
}

```

4.2.5.11 Block Distribution and Cyclic Distribution

The methods of loop slicing in the automatic parallelization are block distribution and cyclic distribution.

The block distribution is to allocate the block that the loop iteration count is distributed by the number of threads to each thread.

The cyclic distribution is to sequentially allocate the block that the loop iteration count is distributed by the arbitrary size to each thread.

The default method of loop slicing in the automatic parallelization is block distribution.

When `parallel_cyclic` optimization control specifier is specified, the loop is cyclically distributed.

Figure 4.14 Block distribution and cyclic distribution

```

for(i = 0; i < 20; i++) {
    for(j = i; j < 20; j++){
        a[i][j] = b[i][j];
    }
}

```

} parallel loop

When block distribution or cyclic distribution is applied to the program of "Figure 4.14 Block distribution and cyclic distribution", the loop iteration count is allocated to the each thread as follows:

- Block distribution

The blocks that the loop iteration count is divided by the number of threads (2) are allocated to each thread as follows:

Thread 0:	{0,1,2,3,4,5,6,7,8,9}
Thread 1:	{10,11,12,13,14,15,16,17,18,19}

- Cyclic distribution (block size: 2)

The blocks that the loop iteration count is divided by two iteration (block size: 2) are sequentially allocated to each thread as follows:

Thread 0:	{0,1}, {4,5}, {8,9}, {12,13}, {16,17}
Thread 1:	{2,3}, {6,7}, {10,11}, {14,15}, {18,19}

4.2.6 Optimization Control Line

This system provides the optimization control line (OCL) to guide automatic parallelization. The optimization control line (OCL) takes effect when both the `-Kparallel` and `-Kocl` options are set.

4.2.6.1 Optimization Control Specifier

The optimization control line (OCL) can have several functions depending on the optimization control specifier. The following table shows a list of the optimization control specifiers for automatic parallelization.

The specifier gives the compiler information on parallelization to the compiler.

Table 4.1 Optimization Identifiers that can be specified for parallelization

Optimization Control Specifiers	Explanation	Optimization control line that can be Specified[a]		
		global line	procedure line	loop line
<code>array_private</code>	Designates privatized arrays.	*	*	*
<code>noarray_private</code>	Designates no privatized arrays.	*	*	*
<code>independent [ext[,ext]...]^[b]</code>	Performs the loop slice that calls function (<i>ext</i>).	-	*	*
<code>loop_part_parallel</code>	Designates automatic parallelization that requires dividing loops.	*	*	*
<code>loop_nopart_parallel</code>	Suppresses automatic parallelization that requires dividing loops.	*	*	*
<code>serial</code>	Suppresses automatic parallelization.	*	*	*
<code>parallel</code>	Performs automatic parallelization.	*	*	*
<code>parallel_cyclic [n]</code>	Designates cyclic distribution in block size (<i>n</i>)	-	-	*
<code>parallel_strong</code>	Designates to parallelize a loop with a small number of loop iterations or a loop that has a small number of operations.	*	*	*
<code>reduction</code>	Designates to apply reduction operation.	*	*	*
<code>noreduction</code>	Designates not to apply reduction operation.	*	*	*
<code>temp [var[,var]...]^[d]</code>	Specifies variable <i>var</i> that is used temporarily in loop.	*	*	*
<code>temp_private var [,var]...^[d]</code>	Designates to assign variable (<i>var</i>) to local area in thread. As a result, the data dependency between threads is solved, and the automatic parallelization is promoted.	-	-	*
<code>first_private var [,var]...^[d]</code>	Designates to assign variable (<i>var</i>) to local area in thread. And, designates to have a possibility of reference of the initial value of the variable. As a result, the data dependency between threads is solved, and the automatic parallelization is promoted.	-	-	*
<code>last_private var [,var]...^[d]</code>	Designates to assign variable (<i>var</i>) to local area in thread. And, designates to have a possibility of reference of the variable after executing the loop. As a result, the data dependency between threads is solved, and the automatic parallelization is promoted.	-	-	*
<code>var1 op var2</code> or <code>var1 op const</code>	Designates a relationship between variable <i>var1</i> and variable <i>var2</i> or between variable <i>var1</i> and constant <i>const</i> .	*	*	*

[a]

* : Specifies an optimization control specifier to an optimization control line.

- : Cannot specify an optimization control specifier to an optimization control line.

[b] *ext* should be declared before use.

[c] `ary` should be declared before use.

[d] `var` should be declared before use.

4.2.6.2 Automatic Parallelization and Optimization Control Specifiers

An optimization control specifier becomes ineffective for a loop that cannot be sliced, even if the optimization control specifier for automatic parallelization is specified. See Section ["4.2.5.8 Restrictions on Loop Slicing"](#) for the loops that cannot be sliced.

4.2.6.3 Optimization Control Specifiers for Automatic Parallelization

Optimization control specifiers for automatic parallelization are described as follows.

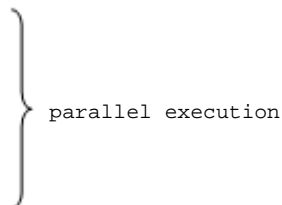
`array_private` specifier

The `array_private` specifier instructs that arrays in a loop are privatized.

The `array_private` specifier is in effect for the code shown in ["Figure 4.15 Usage of the `array_private` specifier"](#), and array `a` in the loop is privatized and the loop is parallelized.

Figure 4.15 Usage of the `array_private` specifier

```
#pragma loop array_private
for(i = 0; i < 100; i++) {
    for(j = 0; j < 1000; j++) {
        a[j] = i + d[j];
        b[i][j] = a[1] + c[j];
    }
}
```



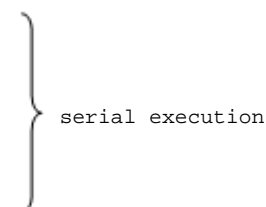
`noarray_private` specifier

The `noarray_private` specifier instructs that arrays in a loop are not privatized.

The `noarray_private` specifier is in effect for the code shown in ["Figure 4.16 Usage of the `noarray_private` specifier"](#), and array `a` in the loop is not privatized and the loop is not parallelized.

Figure 4.16 Usage of the `noarray_private` specifier

```
#pragma loop noarray_private
for(i = 0; i < 100; i++) {
    for(j = 0; j < 1000; j++) {
        a[j] = i + d[j];
        b[i][j] = a[1] + c[j];
    }
}
```



`independent` specifier

The `independent` specifier instructs that parallel execution will give the same results as serial execution even if a function is called in the loop. As a result, the loop that contains the function is suitable for loop slicing.

A function name which does not inhibit loop slicing can be specified to the `independent` specifier. If a function name is omitted, the `independent` specifier is set for all functions within the range of the loop. A function that is a target of the `independent` specifier should be compiled with the `-Kparallel` option.

In the program of "[Figure 4.17 Example program without independent specifier](#)", because the `func` called in the loop may have side effects, the system cannot determine whether the loop can be parallelized.

Figure 4.17 Example program without independent specifier

```
#include <math.h>
main()
{
    int i;
    double a[10000],j;
    double func(double);
    for(i = 0; i < 10000; i++){
        j = 1;
        a[i] = func(j);
    }
    ...
}
double func(double j)
{
    return sqrt(pow(j, 2) + 3 * j + 6);
}
```

If it is known that the function `func` has no side effects, slicing of the loop can be promoted using the `independent` specifier, as shown in "[Figure 4.18 Usage of the independent specifier](#)".

Figure 4.18 Usage of the independent specifier

```
#include <math.h>
main()
{
    int i;
    double a[10000],j;
    double func(double);

    #pragma loop independent func
    for(i = 0; i < 10000; i++){
        j = 1;
        a[i] = func(j);
    }
    ...
}
double func(double j)
{
    return sqrt(pow(j, 2) + 3 * j + 6);
}
```

} parallel execution



This system may produce incorrect results if a procedure that cannot be a target of loop-slicing is specified for the independent specifier.

An example of such a procedure is:

- a procedure that has data dependency with other procedures

loop_part_parallel specifier

The `loop_part_parallel` specifier instructs to perform automatic parallelization that requires dividing loops.

The `loop_part_parallel` specifier is in effect for the code shown in "[Figure 4.19 Usage of the loop_part_parallel specifier](#)", then the loop is divided and automatic parallelization is applied to the part of the divided loop without the `-Kloop_part_parallel` option.

Figure 4.19 Usage of the `loop_part_parallel` specifier

```
#pragma loop loop_part_parallel
for(i = 1; i < n; i++){
    a[i] = a[i] - b[i] + log(c[i]);      ] parallel execution
    d[i] = d[i-1] + a[i];                ] serial execution
}
```

loop_nopart_parallel specifier

The `loop_nopart_parallel` specifier instructs to suppress automatic parallelization that requires dividing loops.

The `loop_nopart_parallel` specifier is in effect for the code shown in "[Figure 4.20 Usage of the loop_nopart_parallel specifier](#)", then the loop is not divided and automatic parallelization is not performed to the loop.

Figure 4.20 Usage of the `loop_nopart_parallel` specifier

```
#pragma loop loop_nopart_parallel
for(i = 1; i < n; i++){
    a[i] = a[i] - b[i] + log(c[i]);
    d[i] = d[i-1] + a[i];
}
```

} serial execution

serial specifier

The `serial` specifier instructs to inhibit loop slicing.

For instance, if the programmer knows that serial execution of a loop is faster than parallel execution, perhaps because the iteration count will always be small, the `serial` specifier may be specified for the loop.

In the program of "[Figure 4.21 Example program without the serial specifier](#)", if loop2 should not be sliced, the slicing can be inhibited by specifying `serial` specifier as shown in "[Figure 4.22 Usage of the serial specifier](#)".

Figure 4.21 Example program without the `serial` specifier

```

for(j = 0; j < 10; j++){
    for(i = 0; i < 1; i++){      <- loop1
        a1[j][i] = a1[j][i] + b1[j][i];
    }
}
...
for(j = 0; j < 10; j++){
    for(i = 0; i < m; i++){      <- loop2
        a2[j][i] = a2[j][i] + b2[j][i];
    }
}
...
for(j = 0; j < 10; j++){
    for(i = 0; i < n; i++){      <- loop3
        a3[j][i] = a3[j][i] + b3[j][i];
    }
}

```

parallel execution

parallel execution

parallel execution

Figure 4.22 Usage of the serial specifier

```

for(j = 0; j < 10; j++){
    for(i = 0; i < 1; i++){      <- loop1
        a1[j][i] = a1[j][i] + b1[j][i];
    }
}
...
#pragma loop serial
for(j = 0; j < 10; j++){
    for(i = 0; i < m; i++){      <- loop2
        a2[j][i] = a2[j][i] + b2[j][i];
    }
}
...
for(j = 0; j < 10; j++){
    for(i = 0; i < n; i++){      <- loop3
        a3[j][i] = a3[j][i] + b3[j][i];
    }
}

```

parallel execution

serial execution

parallel execution

parallel specifier

The parallel specifier instructs to reverse the effect of the serial specifier and to allow loop slicing for specific loops.

For example, it may be necessary to slice only loop2 in the program of "[Figure 4.23 Example program without the parallel specifier](#)". This can be achieved by specifying a combination of the parallel and serial specifiers as shown in "[Figure 4.24 Example program with combination of the parallel and serial specifiers](#)".

Figure 4.23 Example program without the parallel specifier

```

void foo(){
    ...
    for(j = 0; j < m1; j++){          <- loop1
        for(i = 0; i < 100; i++){
            a1[j][i] = a1[j][i] + b1[j][i];
        }
    }
    ...

    for(j = 0; j < m2; j++){          <- loop2
        for(i = 0; i < 100; i++){
            a2[j][i] = a2[j][i] + b2[j][i];
        }
    }
    ...

    for(j = 0; j < m3; j++){          <- loop3
        for(i = 0; i < 100; i++){
            a3[j][i] = a3[j][i] + b3[j][i];
        }
    }
    ...
}

```

parallel execution

parallel execution

parallel execution

Figure 4.24 Example program with combination of the `parallel` and `serial` specifiers

```

#pragma global serial
void foo(){
    ...

    for(j = 0; j < m1; j++){          <- loop1
        for(i = 0; i < 100; i++){
            a1[j][i] = a1[j][i] + b1[j][i];
        }
    }
    ...

    #pragma loop parallel
    for(j = 0; j < m2; j++){          <- loop2
        for(i = 0; i < 100; i++){
            a2[j][i] = a2[j][i] + b2[j][i];
        }
    }
    ...

    for(j = 0; j < m3; j++){          <- loop3
        for(i = 0; i < 100; i++){
            a3[j][i] = a3[j][i] + b3[j][i];
        }
    }
    ...
}

```

serial execution

parallel execution

serial execution

parallel_cyclic specifier

The `parallel_cyclic` specifier instructs to perform cyclic distribution for a loop that automatic parallelization can be analyzed by compiler. n following the `parallel_cyclic` specifier is a decimal value ranging from 1 to 10000, and specifies the block size. If n is omitted, the default block size is 1.

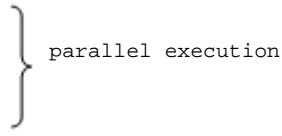
When this specifier is applied, the loop is parallelized without estimating the effects of the parallelization.

For details about cyclic distribution, see Section "[4.2.5.11 Block Distribution and Cyclic Distribution](#)".

In "[Figure 4.25 Usage of the parallel_cyclic specifier](#)", cyclic distribution in block size 2 is applied to the loop.

Figure 4.25 Usage of the `parallel_cyclic` specifier

```
#pragma loop parallel_cyclic 2
for (i = 0; i < 1000; i++) {
    for (j = i; j < 1000; j++) {
        a[i][j] = b[i][j];
    }
}
```



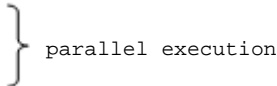
parallel_strong specifier

The `parallel_strong` specifier instructs to indicate compelling parallelization for loops that might not usually be parallelized, for example those with minimal iteration or with few operations.

Usually, for the code shown in "[Figure 4.26 Usage of the parallel_strong specifier](#)", parallelization is suppressed to the loop because of the minimal iteration, but the `parallel_strong` specifier forces parallelization to the loop.

Figure 4.26 Usage of the `parallel_strong` specifier

```
#pragma loop parallel_strong
for (i = 0; i < 10; i++) {
    a[i] = a[i] + b[i] + c[i];
}
```



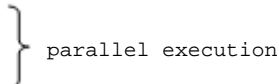
reduction specifier

The `reduction` specifier instructs that the reduction operation in a loop is parallelized.

The `reduction` specifier is in effect for the code shown in "[Figure 4.27 Usage of the reduction specifier](#)", and the reduction operation in the loop is parallelized without the `-Kreduction` option.

Figure 4.27 Usage of the `reduction` specifier

```
#pragma loop reduction
for(i = 0; i < 5000; i++){
    s = s + a[i];
}
```



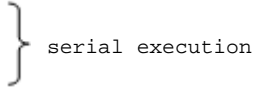
noreduction specifier

The `noreduction` specifier instructs that the reduction operation in a loop is not parallelized.

The `noreduction` specifier is in effect in "[Figure 4.28 Usage of the noreduction specifier](#)", and the reduction operation in the loop is not parallelized.

Figure 4.28 Usage of the `noreduction` specifier

```
#pragma loop noreduction
for(i = 0; i < 5000; i++){
    s = s + a[i];
}
```



`temp` specifier

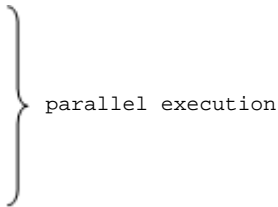
The `temp` specifier instructs that the variables listed are used only within the loop. As a result, the execution performance of the parallelized loop can be improved.

Figure 4.29 Example program without a `temp` specifier

```
int t;
main()
{
    ...

    for(j = 0; j < 50; j++){
        for(i = 0; i < 1000; i++){
            t = a[j][i] + b[j][i];
            c[j][i] = t + d[j][i];
        }
    }
    ...

    func();
}
```

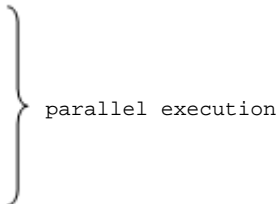


In this case, if it is known that the value of `t` at the end of the loop is not needed in the function `func`, the programmer may assign the `temp` specifier with `t` as shown in "[Figure 4.30 Usage of the temp specifier](#)". As a result, the execution performance improves, because the instruction which corrects the value of `t` becomes unnecessary at the end of the loop.

Figure 4.30 Usage of the `temp` specifier

```
int t;
main()
{
    ...

    #pragma loop temp t
    for(j = 0; j < 50; j++){
        for(i = 0; i < 1000; i++){
            t = a[j][i] + b[j][i];
            c[j][i] = t + d[j][i];
        }
    }
    ...
}
```



```
func();
}
```



Note

Usage of this specifier is the programmer's responsibility. If a variable is specified in a `temp` specifier, and is referred to later in the code, the results will be unpredictable.

`temp_private` specifier

In the target loop of automatic parallelization, the `temp_private` specifier instructs to treat a specified variable `var` as a local variable in each thread.

When the optimization message like the following is output, specify this specifier to assign the target data to local area of each thread. As a result, the data dependency is solved, and the automatic parallelization is promoted.

```
jwd5208p-i "file", line n: This loop is not parallelized because the uncertain order of the
definition and reference to variable 'var' may cause different results from serial execution.
```

This specifier is only effective for the subsequent loop of this specifier.

The variable that can be specified for this specifier is as follows.

- The variable of integer type or float type.
- The variable of array type with integer type or float type in element, and array size has been fixed at compilation.

When this specifier is applied, the following message is output.

```
jwd5013p-i "file", line n: The temp_private specifier in Optimization Control Line (OCL) is applied
to variable 'var'.
```

In "[Figure 4.31 Example program without `temp_private` specifier](#)", this loop is not parallelized, because the order of definition and the reference of array '`a`' is uncertain in an outside loop.

Figure 4.31 Example program without `temp_private` specifier

```
for (i = 0; i < 100; i++) {          <- Target loop for automatic parallelization.
    for (j = 0; j < m; j++) {        <- Because the value of variable 'm' is uncertain, the range
        a[j] = b[j];                where array 'a' is defined is uncertain.
    }
    for (j = 0; j < n; j++) {        <- Because the value of variable 'n' is uncertain, the range
        c[i][j] = a[j];              where array 'a' is referred is uncertain.
    }
}
```

In "[Figure 4.32 Usage of `temp_private` specifier](#)", data dependency of array '`a`' is solved by the `temp_private` specifier. As a result, the outside loop is parallelized.

Figure 4.32 Usage of `temp_private` specifier

```
#pragma loop temp_private a
for(i = 0; i < 100; i++){
    for(j = 0; j < m; j++){
        a[j] = b[j];
    }
    for(j = 0; j < n; j++){
        c[i][j] = a[j];
    }
}
```

} parallel execution

Note

When `temp_private` specifier is specified incorrectly, errors in execution results may occur.

first_private specifier

In the target loop of automatic parallelization, the `first_private` specifier instructs to treat a specified variable `var` as a local variable in each thread, and instructs that the initial value of the variable is referred at the loop entry of each thread.

When the optimization message like the following is output, specify this specifier to assign the target data to local area of each thread. As a result, the data dependency is solved, and the automatic parallelization is promoted.

```
jwd5208p-i "file", line n: This loop is not parallelized because the uncertain order of the
definition and reference to variable 'var' may cause different results from serial execution.
```

This specifier is only effective for the subsequent loop of this specifier.

The variable that can be specified for this specifier is as follows.

- The variable of integer type or float type.
- The variable of array type with integer type or float type in element, and array size has been fixed at compilation.

When this specifier is applied, the following message is output

```
jwd5014p-i "file", line n: The first_private specifier in Optimization Control Line (OCL) is
applied to variable 'var'.
```

In "Figure 4.33 Example program without `first_private` specifier", this loop is not parallelized, because the order of definition and the reference of array 'a' is uncertain in an outside loop.

Figure 4.33 Example program without `first_private` specifier

```
for (i = 0; i < 100; i++) {          <- Target loop for automatic parallelization.
    for (j = 0; j < m; j++) {        <- Because the value of variable 'm' is uncertain, the range
        b[i][j] = a[j];              where array 'a' is defined is uncertain.
    }
    for (j = n; j < 100; j++) {      <- Because the value of variable 'n' is uncertain, the range
        a[j] = b[i][j] + d[j];        where array 'a' is referred is uncertain.
        c[i][j] = a[j];
    }
}
```

In "Figure 4.34 Usage of `first_private` specifier", data dependency of array 'a' is solved by the `first_private` specifier. As a result, the outside loop is parallelized.

Figure 4.34 Usage of `first_private` specifier

```
#pragma loop first_private a
for(i = 0; i < 100; i++){
    for(j = 0; j < m; j++){
        b[i][j] = a[j];
    }
    for(j = n; j < 100; j++){
        a[j] = b[i][j] + d[j];
        c[i][j] = a[j];
    }
}
```

} parallel execution



When `first_private` specifier is specified incorrectly, errors in execution results may occur.

`last_private` specifier

In the target loop of automatic parallelization, the `last_private` specifier instructs to treat a specified variable `var` as a local variable in each thread, and instructs that the value of the variable is referred after executing the loop.

When the optimization message like the following is output, specify this specifier to assign the target data to local area of each thread. As a result, the data dependency is solved, and the automatic parallelization is promoted.

```
jwd5208p-i "file", line n: This loop is not parallelized because the uncertain order of the
definition and reference to variable 'var' may cause different results from serial execution.
```

This specifier is only effective for the subsequent loop of this specifier.

The variable that can be specified for this specifier is as follows.

- The variable of integer type or float type.
- The variable of array type with integer type or float type in element, and array size has been fixed at compilation.

When this specifier is applied, the following message is output.

```
jwd5015p-i "file", line n: The last_private specifier in Optimization Control Line (OCL) is applied
to variable 'var'.
```

In "Figure 4.35 Example program without `last_private` specifier", this loop is not parallelized, because the order of definition and the reference of array 'a' is uncertain in an outside loop.

Figure 4.35 Example program without `last_private` specifier

```
for (i = 0; i < 100; i++) {      <- Target loop for automatic parallelization.
    for (j = 0; j < m; j++) {
        b[i][j] = i;
    }
    for (j = n; j < 100; j++) { <- Because the value of variable 'n' is uncertain, the range
        a[j] = b[i][j] + d[j];    where array 'a' is referred is uncertain.
        c[i][j] = a[j];
    }
}
foo(a, c);
```

In "Figure 4.36 Usage of `last_private` specifier", data dependency of array 'a' is solved by the `last_private` specifier. As a result, the outside loop is parallelized.

Figure 4.36 Usage of `last_private` specifier

```
#pragma loop last_private a
for(i = 0; i < 100; i++){
    for(j = 0; j < m; j++){
        b[i][j] = i;
    }
    for(j = n; j < 100; j++){
        a[j] = b[i][j] + d[j];
        c[i][j] = a[j];
    }
}
foo(a, c);
```

} parallel execution



Note

When `last_private` specifier is specified incorrectly, errors in execution results may occur.

var1 op var2 specifier

var1 op const specifier

The "*var1 op var2*" and "*var1 op const*" specifiers indicate the relationship between variables or between a variable and a constant.

var1 and *var2* should be variables. *const* should be an integer constant. *op* should be the relational operators (<, <=, >, >=) or the equality operators (==, !=).

In "Figure 4.37 Usage of the *var1 op const* specifier", since the relationship between values of the subscripts of array elements is not known unless the optimization control line is described, data dependency of array a in parallel execution may vary with that in serial execution, and the loop is not parallelized. In this case, by specifying the "*var1 op const*" specifier, it turns out that data dependency in parallel execution is equivalent to that in serial execution, and the loop is parallelized.

Figure 4.37 Usage of the *var1 op const* specifier

```
int i, m;
float a[10000], b[10000];
...

#pragma loop m > 2000
for (i = 0; i < 2000; i++){
    a[i] = a[i + m] + b[i];
}
```

} parallel execution

4.2.7 Notes on Automatic Parallelization

This section provides notes on the automatic parallelization feature.

4.2.7.1 Multiprocessing of Nested Loops

This system can only parallelize one level of nesting in nested loops. If there is a call in a parallelized loop to a function which contains itself another parallelized loop, a nest of parallelized loops through function boundaries is generated. A program that contains these types of nested loops will give unpredictable results when compiled with the `-Kparallel,instance=N` option. It should be compiled only with the `-Kparallel` compiler option.

"[Figure 4.38 Multiprocessing of nested loops](#)" shows an example in which one of the parallelized loops should be executed in serial. If a source program that contains such a nest of loops is compiled with the `-Kparallel,instance=N` option, the result may be incorrect.

Figure 4.38 Multiprocessing of nested loops

```
#include <stdio.h>
#define M 4000
double a[M][M], b[M][M];
void f1(int);
void f2(int);

main()
{
    f1(M);
    printf("%e %e\n", a[0][0], b[0][0]);
}

void f1(int n)
{
    int i;
#pragma loop independent
    for(i = 0; i < n; i++){      <- Executed in parallel
        f2(i);
    }
}

void f2(int i)
{
    int j;
    for(j = 0; j < i + 1; j++){  <- Should be executed in serial
        a[i][j] = 3.8;
    }
    for(j = 0; j < i + 1; j++){  <- Should be executed in serial
        b[i][j] = 4.8;
    }
}
```

The result may be incorrect if the source program "a.c" is compiled as follows.

```
$ fccpx -Kparallel,instance=4 -Nlibomp a.c (invalid use)
```

To prevent this type of incorrect execution, serialize the loop in the function by specifying the `serial` specifier as follows.

```
void f2(int i)
{
    int j;
#pragma procedure serial
    for(j = 0; j < i + 1; j++){  <- Serialization
        a[i][j] = 3.8;
    }
    for(j = 0; j < i + 1; j++){  <- Serialization
        b[i][j] = 4.8;
    }
}
```

```
}
}
```

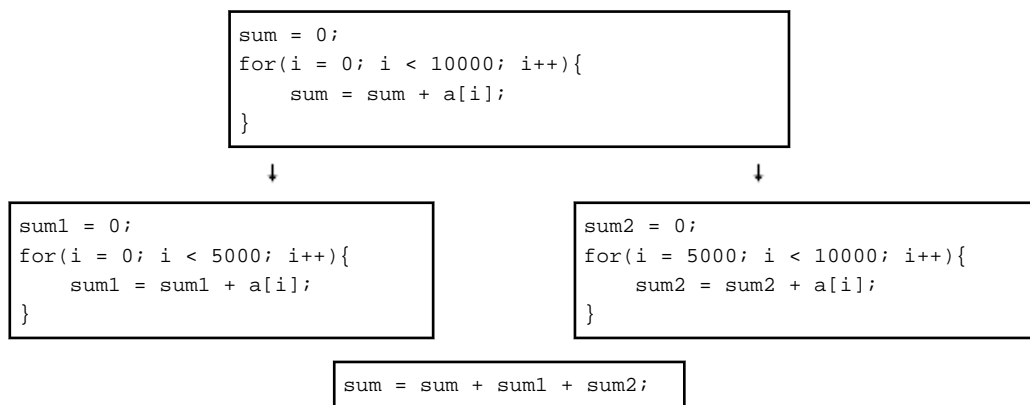
4.2.7.2 Side Effect of Using -Kparallel,reduction

When -Kparallel,reduction is specified as a compiler option, the result of parallel execution may be different from the result of serial execution. The reason for this is that the order of operations in the parallel execution may be different from the order of operations in the serial execution.

"Figure 4.39 Calculation error with the reduction optimization" is an example illustrating the loop reduction optimization in Section "4.2.5.7 Loop Reduction". The variable sum accumulates the values a[0] to a[9999] in order with serial execution. In parallel execution, sum1 accumulates the values a[0] to a[4999], and sum2 accumulates the values a[5000] to a[9999] simultaneously. After that, the sum of sum1 and sum2 is assigned to sum.

Loop reduction optimization may cause a calculation error in the execution result, because the order of adding the array elements is different between parallel and serial execution.

Figure 4.39 Calculation error with the reduction optimization



4.2.7.3 Examples of Invalid Usage of an Optimization Control Line

This section shows examples of invalid usage of the optimization control lines norecurrence, temp, independent, temp_private, and last_private. The system will perform incorrect loop slicing because of the incorrect optimization control lines.

In "Figure 4.40 Invalid usage of the norecurrence specifier", the norecurrence specifier is specified incorrectly for array a. The execution result will be unpredictable when array a is sliced, because the order of data references for array a in parallel execution is different from the order of data references in serial execution.

Figure 4.40 Invalid usage of the norecurrence specifier

```
#pragma loop norecurrence a
for(i = 1; i < 10000; i++){
    a[i] = a[i-1] + b[i];
}
```

In "Figure 4.41 Invalid usage of the temp specifier", the temp specifier is specified incorrectly for variable t. There will not be a correct value assigned to variable last, because the system cannot guarantee a correct value of variable t at the end of the loop.

Figure 4.41 Invalid usage of the temp specifier

```
#pragma loop temp t
for(i = 0; i < 1000; i++){
    t = a[i] + b[i];
    c[i] = t + d[i];
}
last = t;
```

In "Figure 4.42 Invalid usage of the independent specifier", the independent specifier is specified incorrectly for function func. The execution result is unpredictable when array a is sliced, because the order of the data references for array a in parallel execution is different from the order of data references in serial execution.

Figure 4.42 Invalid usage of the independent specifier

```
int a[1000], b[1000];

main()
{
    int i;
    void func(int);
#pragma procedure independent func
    for(i = 1; i < 1000; i++){
        a[i] = b[i] + 1;
        func(i-1);
    }
}

void func(int j)
{
    a[j] = a[j] + 1;
}
```

In "Figure 4.43 Invalid usage of the temp_private specifier", the temp_private specifier is specified incorrectly for variable a. Because an uncertain value is referred in each thread, the execution result is incorrect.

Figure 4.43 Invalid usage of the temp_private specifier

```
#pragma loop temp_private a
for(i = 0; i < 1000; i++){
    b[i] = a;
}
```

In "Figure 4.44 Invalid usage of the last_private specifier", the last_private specifier is specified incorrectly for array a. Because the array a may not be defined by the last thread, the execution result may be incorrect.

Figure 4.44 Invalid usage of the last_private specifier

```
#pragma loop last_private a
for(j = 0; j < 1000; j++){
    for(i = j; i < 500; i++){
        a[i] = b[i];
    }
}
```



```
}
foo(a);
```

4.2.7.4 Standard Library Function References

If there is a standard library function reference that is not suitable for loop slicing in a function called in a parallelized loop, execution of the program will produce incorrect results. The execution performance of the multiprocessing program may decrease due to the overhead of parallel execution.

4.3 Parallelization by OpenMP Specification

This section explains parallelization as described in the OpenMP Specification. See the "OpenMP Architecture Review Board" website for details about OpenMP specification.

This system supports following specifications except the directives, the environmental variables and the runtime routines for the device.

Compiler Mode	Supported Specifications
Trad Mode	OpenMP 3.1
	Part of the OpenMP 4.0 ^[a]
	Part of the OpenMP 4.5 ^[a]
Clang Mode	OpenMP 4.5 ^[b]
	Part of the OpenMP 5.0 ^[c]

[a] The following functions can be used:

- `simd` construct
- `declare simd` construct
- `proc_bind` clause of `parallel` construct
- `depend` clause of `task` construct
- `taskgroup` construct

[b] The following functions cannot be used:

- `declare simd` construct
- `linear` clause of `taskloop simd` construct

[c] The following functions can be used:

- `in_reduction` clause of `task` construct
- `task_reduction` clause of `taskgroup` construct
- `reduction` and `in_reduction` clauses of `taskgroup` construct

4.3.1 Compilation

For compiling source programs and linking, specify the following options to the compile command.

The specified option is different by whether to perform parallelization by the OpenMP specification.

Parallelization by the OpenMP specification	Specified option	
	Compile time	Link time
When using the parallelization and the SIMD Extensions	-Kopenmp	-Kopenmp -Nlibomp

Parallelization by the OpenMP specification	Specified option	
	Compile time	Link time
When using the SIMD Extensions only, not using the parallelization	-Kopenmp_simd	-

-: There is no specified option.

4.3.1.1 Compiler Option for OpenMP C Program

This section describes the compilation options used to compile OpenMP programs.

-Kopenmp

This option enables OpenMP directives and compiles the source programs.

The -Kopenmp option should also be specified at the linkage phase of an object file, when the object file has previously been compiled with this option.



Example

```
$ fccpx a.c -Kopenmp -c
$ fccpx a.o -Kopenmp -Nlibomp
```

-Kopenmp_simd

-Kopenmp_simd option enables only the `simd` construct and the `declare simd` construct of OpenMP and compiles the source programs.

-Nlibomp

This option specifies to use LLVM OpenMP Library.

-Nlibomp option is required at linking.

4.3.1.2 Obtaining of Optimization Information

This section describes Trad Mode compiler options obtaining optimization Information.

-Nsrc, -Nlst

When the -Kopenmp option is specified with -Nsrc or -Nlst simultaneously for the compilation of an OpenMP C program, the following letters are considered optimization information and are noted as such in the statements in the compilation information:

- 'p' is marked on each statement that can be executed in parallel. This is not marked on a statement executed redundantly. Statements directly enclosed by a construct of "for" directive or "sections" directive meet this condition.
- 's' is marked on each statement that will be executed on a single thread simultaneously. Statements directly enclosed by a construct of a "master" directive, a "single" directive, a "critical" directive, or "ordered" directive meet this condition. A statement specified with an "atomic" directive is marked with 's' when the whole statement is executed with a single thread exclusively.
- 'm' is marked on each statement that can be executed partially in parallel and partially in serial.

For nested parallelism, the mark is related to the innermost `parallel` region enclosing the statement. The output is not affected whether or not the `parallel` region being processed is executed in parallel or serially.

-Koptmsg=2

The diagnostic message related to the execution performance such as SIMD Extensions and collapse of the OpenMP specification is output.

4.3.1.3 Restriction of OpenMP programs

In Trad Mode, user defined functions that include OpenMP directives are not expanded inline.

4.3.2 Execution Process

The execution process of an OpenMP C program is the same as the procedure for a serial-processing program.

4.3.2.1 Environment Variable at Execution

Environment Variable `OMP_WAIT_POLICY`

ACTIVE	Uses spin wait until all threads are synchronized.
PASSIVE	Uses no spin wait but suspending wait.

The default value is "PASSIVE".

Select ACTIVE to give priority to the elapsed time. Select PASSIVE to give priority to the CPU time.

Environment Variable `OMP_PROC_BIND`

The environment variable `OMP_PROC_BIND` can control the thread affinity. Either `true`, `false` or comma separated list of `master`, `close`, or `spread` can be specified to this environment variable. The values of the list sets the thread affinity policy used by a `parallel` region corresponding to the nest level.

This environment variable is set to `close` by default.

If this environment variable is set to `false`, thread affinity is disabled, and the `proc_bind` clause on `parallel` construct is ignored.

Otherwise, thread affinity is enabled and the initial thread is bound to first place in place list.

The `master` thread affinity policy indicates that all threads are bound to same place as master thread.

The `close` thread affinity policy indicates that threads are bound to next to the place where master thread is bound.

The `spread` thread affinity policy indicates that the threads are bound to the one of sub-partition places divided from master thread's place partition.

The effect of `true` is same as `spread`.

See environment variable `OMP_PLACES` for place.

Environment Variable `OMP_PLACES`

The environment variable `OMP_PLACES` defines place list.

The explicit place list is defined by ordered set of comma separated non-negative numbers enclosed by braces. The numbers represents the smallest unit of CPU resource in the system.

The value of `OMP_PLACES` is either abstract name or an explicit place list. The value can be specified using following format.

<code>{ abstract-name[(num-places)] place-list }</code>

abstract-name

The following abstract names can be set to this environment variable. The default value is `cores`.

Abstract Name	Meaning
<code>threads</code>	Each place corresponds to a single hardware thread, which represents smallest unit of CPU resource in the system. The effect of <code>threads</code> is equivalent to <code>cores</code> .
<code>cores</code>	Each place corresponds to a single hardware core, which represents smallest unit of CPU resource in the system. The effect of <code>cores</code> is equivalent to <code>threads</code> .
<code>sockets</code>	Each place corresponds to a single socket, which represents NUMA node.

num-places

The length of the place list. Positive integer.

place-list

The explicit place list can be specified using following format.

```
{ place:length[:stride] | [!]place[,place-list] } (*1)
```

place

"{"*placeI*" }" (*2)

place1

{ *cpuid:length[:stride] | [!]cpuid[,placeI] }*

length

The length of elements. Positive integer.

stride

The value of increment or decrement. Positive integer. Default value is 1.

cpuid

Smallest unit of CPU resource in the system.

*1) An exclusion operator "!" exclude the number or place immediately following the operator.

*2) "{" and "}" are braces.



Example

Example of environmental variable OMP_PLACES

- Example 1:

```
$ export OMP_PLACES="cores"
```

This example defines place list of *cores* abstract name.

- Example 2:

```
$ export OMP_PLACES="cores(4)"
```

This example defines place list of *cores* abstract name of 4 places.

- Example 3:

```
$ export OMP_PLACES="{12,13,14},{15,16,17},{18,19,20},{21,22,23}"
$ export OMP_PLACES="{12:3},{15:3},{18:3},{21:3}"
$ export OMP_PLACES="{12:3}:4:3"
```

All above examples define same place of 0 to 12 to 14, 15 to 17, 18 to 20, and 21 to 23.

Environment Variable OMP_STACKSIZE

A user can specify the stack area size for each thread using the environment variable OMP_STACKSIZE in Bytes, Kbytes, Mbytes, Gbytes, or Tbytes. The default size is 8M bytes.

Environment Variable GOMP_CPU_AFFINITY

Threads are bound to CPUs in order of the specified cpuid list.

When the number of specified CPUs is exceeded, it is repeatedly used from the beginning of the list.

The cpuid list shall be separated by comma (',') or space (' ').

The cpuid list can have the next form that has range with increment.

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: cpuid of the beginning of the range. ($0 \leq \text{cpuid1} < \text{CPU_SETSIZE}$)

cpuid2: cpuid of the end of the range. ($0 \leq \text{cpuid2} < \text{CPU_SETSIZE}$)

inc: increment ($1 \leq \text{inc} < \text{CPU_SETSIZE}$)

In addition, it is necessary to be the following.

```
cpuid1 <= cpuid2
```

It becomes equivalent to the case where all CPUs for every increment value *inc* in the range from *cpuid1* to *cpuid2* are specified.

The cpuid can be used the above-mentioned value. However, cpuid which can actually be assigned becomes only within the limits of CPU affinity of the process at the start of execution.

See CPU_SET(3) about details of CPU_SETSIZE.



Note

If cpuid is outside the CPU affinity of the process at the start of execution, an error will be output and the program will terminate. Correct the setting value.



Example

Examples of using environment variable GOMP_CPU_AFFINITY

- Example 1:

```
$ export GOMP_CPU_AFFINITY="12,14,13,15"
```

The thread is bound to CPU in order of 12, 14, 13, and 15.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

- Example 2:

```
$ export GOMP_CPU_AFFINITY="12-19"
```

The thread is bound to CPU in order of 12, 13, 14, 15, 16, 17, 18, and 19.

When the number of threads is nine or more, it is repeatedly used from the beginning of the list.

- Example 3:

```
$ export GOMP_CPU_AFFINITY="12-19:2"
```

The thread is bound to CPU in order of 12, 14, 16, and 18.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

- Example 4:

```
$ export GOMP_CPU_AFFINITY="12-16:2,13,19"
```

The thread is bound to CPU in order of 12, 14, 16, 13, and 19.

When the number of threads is six or more, it is repeatedly used from the beginning of the list.

4.3.2.2 Environment Variable for OpenMP Specifications

The user can set the following environment variables, which are in accordance with the OpenMP Specifications.

For environment variables available on OpenMP 4.0 and later, the specifications they are supported on are enclosed in a pair of parentheses after each environment variable.

For the details, refer to the OpenMP specifications.

Environment Variable OMP_SCHEDULE

Sets the schedule type and chunk size for directives that have the schedule type at runtime.

Environment Variable OMP_NUM_THREADS

Sets the number of threads to use during execution.

Environment Variable OMP_DYNAMIC

Enables or disables dynamic thread adjustment.

Environment Variable OMP_PROC_BIND

Specifies whether threads are bound to CPUs.

Environment Variable OMP_NESTED

Enables or disables nested parallelism.

Environment Variable OMP_STACKSIZE

Specifies the stack size for each thread.

Environment Variable OMP_WAIT_POLICY

Specifies the behavior of the standby thread.

Environment Variable OMP_MAX_ACTIVE_LEVELS

Specifies the maximum number of nested active parallel regions.

Environment Variable OMP_THREAD_LIMIT

Specifies the maximum number of threads performed on an Open MP program.

Environment Variable OMP_PLACES (OpenMP 4.0 and later)

This specifies the list of places for thread affinity.

Environment Variable OMP_CANCELLATION (OpenMP 4.0 and later)

This enables or disables the cancellation.

Environment Variable OMP_MAX_TASK_PRIORITY (OpenMP 4.5 and later)

This specifies maximum task priority.

Environment Variable OMP_DISPLAY_ENV (OpenMP 4.0 and later)

This enables or disables to display the OpenMP version number and the initial value of the internal control variables.

The OpenMP version number is 201611.

4.3.2.3 Notes during Execution

When executing programs created on this system, note the following.

Variable allocation at execution

If the size of the stack area for every thread is defined as a specific size, the environment variable OMP_STACKSIZE can be specified.

See Section "[4.3.2.1 Environment Variable at Execution](#)" for information about environment variables OMP_STACKSIZE.

Limits on the Number of CPUs

The number of CPUs that can be used in the system.

Number of Threads

The number of threads used for OpenMP code is determined with the following priority (while the one for automatic parallelization is described in Section "[4.2 Automatic Parallelization](#)"):

- The value specified with the num_threads clause of the "parallel" directive
- The value specified with the omp_set_num_threads function

- The value of the environment variable `OMP_NUM_THREADS`
- The number of CPUs that can be used with jobs

When the dynamic thread adjustment feature is enabled, the number of threads is adjusted to be less or equal to the number of CPUs.

When the dynamic thread adjustment feature is disabled, the thread number determined by the above priority will be used and multiple threads can be executed on the same processor via time-sharing. Such parallel execution is not effective because the overhead of synchronization among threads is very high. Therefore, it is recommended to set the number of threads to be less than the limit on the number of CPUs.

CPU binding for thread

Environmental variable `OMP_PROC_BIND` or `GOMP_CPU_AFFINITY` can control thread affinity. `GOMP_CPU_AFFINITY` takes precedence over `OMP_PROC_BIND`.

4.3.3 Implementation-Dependent Specifications

The OpenMP specification includes the following processor-dependent specification. For details on each item, refer to the OpenMP Specifications.

Memory Model

If a variable is longer than 4 bytes or crosses a 4-byte boundary:

- The value of the variable written from two threads without synchronization may be undefined.
- The value read by a thread from a variable that is written from another thread without synchronization may be undefined.

Internal Control Variables

The initial values for each of the internal control variables are as follows:

- `nthreads-var`: Maximum number of CPUs.
- `dyn-var`: `FALSE`.
- `run-sched-var`: `static` without chunk size.
- `def-sched-var`: `static` without chunk size.
- `bind-var`: `FALSE`.
- `stacksize-var`: the stack area size for threads. Default size is 8M bytes.
- `wait-policy-var`: `PASSIVE`.
- `thread-limit-var`: 2147483647.
- `max-active-levels-var`: 2147483647.
- `place-partition-var`: Default value is `cores`, which is equivalent to threads in the system.

Dynamic Thread Adjustment Features

This system supports the dynamic thread adjustment described in Section "[4.3.2.3 Notes during Execution](#)".

By default, the dynamic adjustment features are off.

Loop Statement

The type used to calculate the iteration count of a loop that is not nested is `long int`.

When `auto` is set for the internal control variable `run-sched-var`, the effect of the `schedule(runtime)` construct is set to `schedule(static)`.

sections Construct

Assignment to a thread of a structured block in a `sections` construct is performed in the same way as a static schedule.

single Construct

A single region is executed by the thread that encounters the region first.

taskloop Construct

When grainsize and num_tasks clause are not specified, these values will be set appropriately considering the loop iteration.

An eight byte integer type is used to calculate the iteration count of a collapsed loop.

critical Construct

This implementation do not support lock hint. Therefore there is no the effect of using a hint clause.

atomic Construct

Two atomic regions will be executed independently (not exclusively), when a variable type to be updated is different. When the types match, it may be executed exclusively even if the address is different.

- When updating a logical type, complex type, 1 byte integer type, 2 byte integer type, or quadruple precision (16 byte) real type variable
- When updating array element and index expression is not the same
- The target expression has explicit or implicit type conversion

omp_set_num_threads Function

If the argument is not a positive integer, the effect of omp_set_num_threads function is same as the argument is 1. A value that exceeds the number of threads supported by the system must not be specified.

omp_set_schedule Function

There are no schedule types that are implementation-dependent.

omp_set_max_active_levels Function

A call to the omp_set_max_active_levels function will be ignored when it is performed from an explicit parallel region. It will also be ignored when the argument is an integer that less than 0.

omp_get_max_active_levels Function

The omp_get_max_active_levels function can be called from anywhere in the program and it returns the value of the internal control variable max-active-levels-var.

omp_get_place_proc_ids Function

The omp_get_place_proc_ids function returns identifiers of the available processors in the specified place. The definition of processor is the CPU controlled by operating system.

omp_init_lock_with_hint and omp_init_nest_lock_with_hint Function

This implementation do not support lock hint. Therefore there is no the effect of using these functions.

Environment Variable OMP_SCHEDULE

The behavior is different by compiler mode.

When the schedule type specified for OMP_SCHEDULE is invalid, the schedule type is ignored, and the following default schedule is used.

Compiler mode	Default value
Trad Mode	static without chunk size.
Clang Mode	static with chunk size 1.

When the schedule type specified for OMP_SCHEDULE is static, dynamic, or guided, and the chunk size is not a positive number, the chunk size will be as follows:

Compiler mode	Schedule type	Chunk size
Trad Mode	static	no chunk size
	dynamic	1

Compiler mode	Schedule type	Chunk size
	guided	
Clang Mode	static	1
	dynamic	
	guided	

Environment Variable OMP_NUM_THREADS

When a value equal to 0 is specified for the list of OMP_NUM_THREADS, it works in the same ways as when 1 is specified. When a value less than 0 is specified for the list of OMP_NUM_THREADS, the number of threads is determined by "[4.3.2.3 Notes during Execution](#)" procedure. A value that exceeds the number of threads supported by the system must not be specified.

Environment Variable OMP_PROC_BIND

If the value does not conform to the specified format for OMP_PROC_BIND, the value is ignored, and the default value (close) is used.

If all of the following conditions are met, the excess threads are assigned to places evenly as possible.

- spread or close is specified to OMP_PROC_BIND.
- T (number of threads) is greater than P (number of places).
- P does not divide T evenly.

Environment Variable OMP_PLACES

If the value does not conform to the specified format, the value is ignored, and the default value (cores) is used.

Environment Variable OMP_DYNAMIC

When a value other than TRUE or FALSE is specified for OMP_DYNAMIC, the value is ignored and the default value (FALSE) is used.

Environment Variable OMP_NESTED

When a value other than TRUE or FALSE is specified for OMP_NESTED, it is ignored and the default value (FALSE) is used.

Environment Variable OMP_STACKSIZE

When the value specified for OMP_STACKSIZE does not meet the defined format, it is ignored and the default value (8M bytes) is used.

Environment Variable OMP_WAIT_POLICY

ACTIVE performs spin wait. PASSIVE performs suspend wait.

Environment Variable OMP_MAX_ACTIVE_LEVELS

When the value specified for OMP_MAX_ACTIVE_LEVELS is an integer that is less than 0, it is ignored and the default value (2147483647) is used.

Environment Variable OMP_THREAD_LIMIT

When the value specified for OMP_THREAD_LIMIT is not a positive integer, it is ignored and the default value (2147483647) is used.

4.3.4 Notes on OpenMP Programming

This section provides notes on OpenMP programming.

4.3.4.1 Implementation of parallel Region and Explicit task Region

The structured block within a parallel construct or a task construct is compiled as an internal function.

The name of the internal function generated from the parallel construct or the task construct is different depending on the compiler mode. The name of an internal function is shown below.

Compiler mode	Type of internal function	Internal function name
Trad Mode	Internal function generated from parallel construct	The name " <code>._OMP_id-numberA</code> " is added
	Internal function generated from task construct	The name " <code>._TSK_id-numberB</code> " is added
Clang Mode	Internal function generated from parallel construct	<code>.omp_outlined.[_debug__][.id-numberC] (*1) (*2)</code>
	Internal function generated from task construct	<code>.omp_task_entry.[.id-numberC] (*3)</code>

id-numberA: Unique number that identifies parallel construct in a source file.

id-numberB: Unique number that identifies task construct in a source file.

id-numberC: Unique number in a source file. When plurals of parallel constructs and task constructs exist in the source file, a unique number to identify the parallel construct and the task construct collectively is assigned to "*id-numberC*".

*1) When -g option is specified, "`_debug__`" is added.

*2) When two or more parallel constructs are specified in the source file, "*id-numberC*" is added.

*3) When two or more task constructs are specified in the source file, "*id-numberC*" is added.

4.3.4.2 Implementation of threadprivate Variable

The `threadprivate` variable is implemented by using the Thread-Local Storage. When the size of the Thread-Local Storage exceeds the range of the offset, an error occurs at link time.

Example of error message:

```
relocation truncated to fit: R_AARCH64_TLSLE_ADD_TPRL_HI12 against symbol 'varname'
```

When the error occurs, specify the size of an appropriate offset by `-Ktls_size={ 12|24|32|48}` option for Trad Mode or `-mfj-tls-size={ 12|24|32|48}` option for Clang Mode. For details about the `-Ktls_size` option, see Section "2.2.2.5 -K Option". For details about the `-mfj-tls-size` option, see Section "9.1.2.2.6 Options for Code Generation".

4.3.4.3 Automatic Parallelization for OpenMP Programs

The `-Kopenmp` and `-Kparallel` options can be specified at the same time.

If both options are specified, automatic parallelization is not applied to the following loops:

- A loop within the OpenMP directives.
- A loop that statically includes OpenMP directives.
- In the case that a loop is parallelized by OpenMP directives,
 - If the loop itself is parallelized by OpenMP directives.
 - If the loop is lexically contained in a loop that is parallelized by OpenMP directives.

4.3.4.4 Creating Shared Libraries in Clang Mode

To create the following shared libraries in Clang Mode, specify the compiler option `-fopenmp`. If created without the compiler option `-fopenmp`, the program linking the shared library may terminate abnormally.

- A shared library that links a shared library created with the compiler option `-fopenmp`
- A shared library that is dynamically opened with a function such as the function `dlopen` and used from a program created with the compiler option `-fopenmp`



Example

- Example 1

```
$ fccpx shraed1.c -Nclang -fopenmp -shared -o libsample_linked.so
$ fccpx shared2.c -Nclang -fopenmp -shared -o libsample.so -L. -lsample_linked
```

The linked libsample_linked.so was created with the compiler option -fopenmp, so specify the compiler option -fopenmp when creating libsample.so as well.

- Example 2

```
$ fccpx main.c -Nclang -fopenmp -o main.exe
$ fccpx shared.c -Nclang -fopenmp -shared -o libsample.so
```

The program main.exe that dynamically opens libsample.so was created with the compiler option -fopenmp, so specify the compiler option -fopenmp when creating libsample.so as well.

4.4 Runtime Messages

LLVM OpenMP Library outputs runtime messages to standard error as following format.

```
OMP: type message
```

The meaning of runtime messages of LLVM OpenMP Library is explained as follows.

OMP:

Message prefix of LLVM OpenMP library.

type

Message type. The one of the following values is output.

Value	Meanings
Hint	Hint about OpenMP.
Info	Information about OpenMP.
Warning	Warning about OpenMP. Not fatal error. Even if this message is displayed, execution continues.
Error	Fatal error about OpenMP. If this message is displayed, execution is aborted.
System error	Fatal error about the system. If this message is displayed, execution is aborted.

message

Content of message



Example

```
$ export OMP_STACKSIZE=1Z
$ ./a.out
OMP: Warning #80: OMP_STACKSIZE="1Z": value too large.
OMP: Info #107: OMP_STACKSIZE value "9223372036854775807" will be used.
OMP: Error #34: System unable to allocate necessary resources for OMP thread:
OMP: System error #11: Resource temporarily unavailable
OMP: Hint Try decreasing the value of OMP_NUM_THREADS.
```

The above example is series of messages when unacceptable value in FX system is set to environmental value OMP_STACKSIZE as stack size.



Note

The following environment variable is invalidated for the control of the runtime message of LLVM OpenMP Library. For details about the environment variable, see Section "[2.5.1 Environment Variable for Execution](#)".

- Environment variable `FLIB_C_MESSAGE`

Chapter 5 Emitting Information

This chapter provides information on C language program compilation and execution.

5.1 Emitting Information at Compilation

This section explains the information emitted by this system at compilation.

5.1.1 Header

When either the -Nlst, -Nlst_out=*file*, -Nsrc or -Nsta compilation option is specified, a header is emitted for each type of information.

5.1.1.1 Output Format

The header is emitted as shown in the following format.

Header Format

```
Fujitsu C/C++ Version version    date
```

<i>version</i>	Language processing system version.
<i>date</i>	Compilation date and time in the format of the <code>asctime</code> function.

5.1.2 Source List

When the program is compiled, the source list is put out according to the following options.

- -Nlst[={p|t}]

Specifies to output source list and statistics list to file(s).

- -Nlst_out=*file*

Specifies the filename to output source list and statistics list.

When this option is specified, the -Nlst=p option is also effective.

- -Nsrc

Specifies to output source list to the standard output.

Note that when the -Nlst or -Nlst_out=*file* option is specified, the source list is output to the *file*.

The source list includes symbols indicating optimization and parallelization if they are performed.

5.1.2.1 Output Format

The source list is emitted as shown in the following format.

Source List Format

```
Compilation information
Current directory : directory-name
Source file      : source-file-name
(line-no.)(optimize)
nnnnnnnn  pi nnnnnn  source
...
                                <<< Loop-information Start >>>
                                <<<  details optimization information
                                <<< Loop-information  End >>>
nnnnnnnn  pi nnnnnn  source
...
```

<i>directory-name</i>	The name of directory where source file is stored
<i>source-file-name</i>	Source file name
<i>nnnnnnnn</i>	Line number of source (variable length)
<i>p</i>	Symbols for parallelization
<i>i</i>	Symbols for inline expansion
<i>mmmm</i>	The number of loop unrolling (variable length)
<i>v</i>	Symbols for using SIMD Extensions
<i>source</i>	Source line
<<< Loop-information Start >>>(a)	Details of optimization and parallelization information header
<i>details optimization information(a)</i>	Details of optimization and parallelization information which has been performed to the next statement.
<<< Loop-information End >>>(a)	Details of optimization and parallelization information footer

a) Emitted only if the -Nlst=t option is effective.

5.1.2.2 Information Included in Source List

5.1.2.2.1 Line Number of Source

Line number of source is emitted.

5.1.2.2.2 Symbols for Parallelization

The symbols for parallelization are emitted by the following conditions:

- Symbols for loop control statement are emitted at the line including loop control statements.
The loop control statement is "for" statement, "while" statement, "do-while" statement, and "if-goto" statement.
- If the parallelization contains more than one loop control statement, the symbols for the left most loop control line are emitted.

Symbols are classified in two different types; one for loop control statements and one for anything other than loop control statements.

Symbols for loop control statements

p	Indicates all statements in this loop have been parallelized. It becomes "pp" for -Nlst=t in the first line within the parallelization is performed.
m	Indicates that only some of the statements in this loop have been parallelized.
s	Indicates none of the statements in this loop have been parallelized.
Blank	Indicates none of the statements in this loop are targeted for parallelization.

Symbols for anything other than loop control statements

p	Indicates all statements in this loop have been parallelized. ^[a]
m	Indicates that only some of the statements in this loop have been parallelized. ^[b]
s	Indicates none of the statements in this loop have been parallelized.
Blank	Indicates none of the statements in this line are targeted for parallelization.

[a] When the mark "s" is displayed for a loop control statement, the parallelization is not applied because parallelization is available for statements in the loop but worthless.

[b] When the mark "s" is displayed for a loop control statement, the parallelization is not applied because parallelization is available for some of statements in the loop but worthless.

5.1.2.2.3 Symbols for Inline Expansion

If inline expansion is performed, "i" is emitted.

Otherwise, blank is emitted.

5.1.2.2.4 Number of Loop Unrolling

If loop unrolling is performed, multiplicity of loop unrolling is emitted.

If the loop full unrolled, "f" is emitted instead of multiplicity of loop unrolling.

Otherwise, blank is emitted.

5.1.2.2.5 Symbols for Using SIMD Extensions

The symbols for the optimization that uses SIMD Extensions are emitted by the following conditions:

- Symbols for the optimization that uses SIMD Extensions are emitted at the line including loop control statements. The loop control statement is "for" statement, "while" statement, "do-while" statement, and "if-goto" statement.
- If the optimization that uses SIMD Extensions contains more than one loop control statement, the symbols for the left most loop control line are emitted.

Symbols are classified in two different types; one for loop control statements and one for anything other than loop control statements.

Symbols for loop control statements

v	Indicates all statements in this loop have been optimized using SIMD Extensions.
m	Indicates that only some of the statements in this loop have been optimized using SIMD Extensions.
s	Indicates none of the statements in this loop have been optimized using SIMD Extensions.
Blank	Indicates none of the statements in this loop are targeted for optimization using SIMD Extensions.

Symbols for anything other than loop control statements

v	Indicates all statements in this loop have been optimized using SIMD Extensions. ^{[a][c]}
m	Indicates that only some of the statements in this loop have been optimized using SIMD Extensions. ^[b] ^[c]
s	Indicates none of the statements in this loop have been optimized using SIMD Extensions.
Blank	Indicates none of the statements in this line are targeted for optimization using SIMD Extensions.

[a] When the mark "s" is displayed for a loop control statement, SIMD optimization is not applied because SIMD optimization is available for statements in the loop but worthless.

[b] When the mark "s" is displayed for a loop control statement, SIMD optimization is not applied because SIMD optimization is available for some of statements in the loop but worthless.

[c] When SIMD optimization is applied to a loop control statement and the mark "v" or "m" is displayed for the statement, the mark "v" could be displayed for statements in the loop though SIMD extensions are not used for them. The mark "s" is displayed for a statement preventing SIMD optimization, which is a clue to tune programs.

5.1.2.2.6 Details Optimization Information

When the -Nlst=t option is specified, the source list containing details of optimization and parallelization information which has been performed are written to file(s).

The following optimization information are put just before each loop.

About Automatic Parallelization

- Standard iteration count: *N*

When the iteration count of a loop is *N* or more, the loop is brought to parallel execution. When less than *N*, it is brought to serial execution.

Optimization Information In Loop Units

- INTERCHANGED(*nest: nest-no*)

It means that loop was interchanged.

- *nest-no* is the nesting level of the loop that was moved by loop interchange optimization.
- (*nest:*) may not be indicated if the other optimization like loop collapsing was applied.

- FUSED(*lines: num1, num2[, num3] . . .*)

It means that loops were fused.

Loops at line *num2* and subsequent lines were fused into a loop at line *num1*.

- (*lines:*) is not indicated at line *num2* and subsequent lines, that were fused to line *num1*.

- FISSION(*num: N*)

It means that loop was split.

- *N* is the number of loops after fission.

- COLLAPSED

It means that nested loops were collapsed into a single loop.

- SOFTWARE PIPELINING(*IPC: ipc, ITR: itr, MVE: mve, POL: pol*)

It means that software pipelining was applied to the loop.

- *ipc* is the expectation value of the number of executed instructions per cycle (Instructions Per Cycle) when the software pipelining is applied to the loop.
- *itr* is the minimum iteration count required to choose the software pipelined loop. It is the same as the value output by the optimization message jwd8205o-i.
- *mve* is the loop expansion number by the software pipelining.
- *pol* is the applied instruction scheduling algorithm.
S means that the algorithm fit for a small loop is applied.
L means that the algorithm fit for a large loop is applied.
These algorithms correspond to the ones applied by -Kswp_policy=small option and -Kswp_policy=large option, respectively.

- SIMD

It means that SIMD instructions were generated for the loop. It is displayed by one of the following.

- SIMD(*VL: length[, length] . . .*)
- It means that SIMD instructions were generated for the loop. A SIMD instruction treats *length* elements of array. When loop fission is applied to the loop and the values of *length* for each loop are different, two or more *lengths* are displayed.
- SIMD(*VL: AGNOSTIC; VL: length[, length] . . . in 128-bit*)
- It means that SIMD instructions were generated for the loop without regarding the SVE vector register size as a specific size. A SIMD instruction treats *length* elements of array if the size of the SVE vector register is 128-bit. When loop fission is applied to the loop and the values of *length* for each loop are different, two or more *lengths* are displayed.

- STRIPING

It means that loop was performed loop striping.

- MULTI-OPERATION FUNCTION

It means that loop includes multi-operation function.

- PATTERN MATCHING (matmul)

It means that loop was changed to library function call (matmul).

- UNSWITCHING

It means that loop was performed loop unswitching.

- FULL UNROLLING

It means that loop was fully unrolled.

- CLONE

It means that clone optimization was applied to the loop.

- LOOP VERSIONING

It means that loop versioning was applied to the loop.

Information That Relates To The Prefetch

Hardware-prefetch

- PREFETCH(HARD) Expected by compiler:

It indicates that the compiler expects hardware-prefetch and does not generate the prefetch instruction for array data accessed sequentially within a loop.

- array name,...

It indicates an array name expecting to use hardware-prefetch. The array which was generated by compiler is shown as "(unknown)".

Prefetch instruction

- PREFETCH(SOFT): N

It indicates the number of all prefetch instructions that exist in the loop.

In addition, the number and the array name of the prefetch instruction of each access type of the prefetch are shown in the form of the following.

access type: N
array name: N,...

- access type

- SEQUENTIAL: N

It indicates the number of prefetch instructions for array data accessed sequentially within a loop.

- STRIDE: N

It indicates the number of prefetch instructions for array data that is accessed with a stride larger than the cache line size used in the loop.

- INDIRECT: N

It indicates the number of prefetch instructions for array data that is accessed indirectly (list access) within a loop.

- SPECIFIED: N

It indicates the number of prefetch instructions generated by Built-in Functions __builtin_prefetch.

- array name

- array name: N,...

It indicates the number of prefetch instructions for the array name. The array which was generated by compiler is shown as "(unknown)".

Information That Relates To The zfill Optimization

- ZFILL

Indicates the zfill optimization is applied.

- *array name*: ...

Indicates arrays names to which the zfill optimization is applied. An array generated by compiler is shown as "(unknown)".

Information That Relates To The Register

- SPILLS :

Indicates the number of saving and restoring instructions for registers that exist in the innermost loop shown for each register type.

- GENERAL : SPILL *N* FILL *N*

Indicates the number of saving and restoring instructions for general registers to and from the memory.

- SIMD&FP : SPILL *N* FILL *N*

Indicates the number of saving and restoring instructions for SIMD and floating-point registers to and from the memory.

- SCALABLE : SPILL *N* FILL *N*

Indicates the number of saving and restoring instructions for scalable vector registers to and from the memory.

- PREDICATE : SPILL *N* FILL *N*

Indicates the number of saving and restoring instructions for predicate registers to and from the memory.

5.1.2.3 Example of Source Listing

Example of source listing is shown in following list.

Example of Source Listing 1

```
Compilation information
Current directory : directory-name
Source file      : source-file-name
(line-no.)(optimize)
 1          #include <stdio.h>
 2
 3          float sub(int i);
 4
 5          int main() {
 6              int i;
 7              float a[1000];
 8
 9  p      2v      for(i = 0; i < 1000; i++) {
10  p      2v          a[i] = i;
11  p      2v      }
12
13          printf("%lf\n", a[0]);
14
15          i = 0;
16          s      _loop:
17          s      if(i < 1000) {
18              goto _next;
19          }
20          s      a[i] = i;
21          s      i++;
22          s      goto _loop;
23          s      _next:
24
25          printf("%lf\n", a[0]);
26
27  s      s      for(i = 0; i < 1000; i++) {
```

```

28  s      s      a[i] = sub(i);
29  s      s      }
30
31      f      for(i = 0; i < 2; i++) {
32      f      a[i] = i;
33      f      }
34
35      printf("%lf\n", a[0]);
36      return 0;
37  }
38
39      float sub(int j) {
40      return (float)j;
41  }

```

In this sample, following information can be read.

- The statements included in the "for"-statement at line-9 are parallelized.
- The statements included in the "for"-statement at line-9 are optimized using SIMD Extensions.
- The statements included in the "for"-statement at line-9 are performed with loop unrolling.
- SIMD Extensions is not used at the loop constructed with "if-goto" statement from line-16 to line-23.
- The statements included in the "for"-statement at line-27 are not parallelized.
- SIMD Extensions is not used at the statements included in the "for"-statement at line-27.
- The statements included in the "for"-statement at line-31 are full unrolled.

Example of Source Listing 2

```

Compilation information
Current directory : directory-name
Source file      : source-file-name
(line-no.)(optimize)
1      #include <stdio.h>
2
3      float sub(int i);
4
5      int main() {
6      int i;
7      float a[10000];
8
9      <<< Loop-information Start >>>
10     <<< [PARALLELIZATION]
11     <<< Standard iteration count: 1000
12     <<< [OPTIMIZATION]
13     <<< SIMD(VL: 16)
14     <<< SOFTWARE PIPELINING(IPC: 3.00, ITR: 80, MVE: 5, POL: S)
15     <<< Loop-information End >>>
16  9  pp    2v    for(i = 0; i < 10000; i++) {
17  10  p    2v    a[i] = i;
18  11  p    2v    }
19
20      printf("%lf\n", a[0]);
21
22      i = 0;
23      _loop:
24      s      if(i < 10000) {
25      s      goto _next;
26      }
27      s      a[i] = i;
28      s      i++;
29      s      goto _loop;

```

```

23      s      _next:
24
25          printf("%lf\n", a[0]);
26
                <<< Loop-information Start >>>
                <<< [PARALLELIZATION]
                <<< Standard iteration count: 1000
                <<< [OPTIMIZATION]
                <<< SIMD(VL: 16)
                <<< SOFTWARE PIPELINING(IPC: 3.00, ITR: 80, MVE: 5, POL: S)
                <<< Loop-information End >>>
27  pp      v      for(i = 0; i < 10000; i++) {
28  pi      v      a[i] = sub(i);
29          }
30
                <<< Loop-information Start >>>
                <<< [PARALLELIZATION]
                <<< Standard iteration count: 1000
                <<< [OPTIMIZATION]
                <<< SIMD(VL: 8)
                <<< Loop-information End >>>
31  s      v      for(i = 0; i < 8; i++) {
32  p      v      a[i] = i;
33  p      v      }
34
35          printf("%lf\n", a[0]);
36          return 0;
37      }
38
39      float sub(int j) {
40          return (float)j;
41      }
Total prefetch num: 0

```

In this sample, following information can be read.

- The loop by "for"-statement at line-9 is parallelized if its iteration count is greater than or equal to 1000.
- The loop by "for"-statement at line-9 is optimized with software pipelining.
- The loop by "for"-statement at line-9 is parallelized.
- The statements included in the "for"-statement at line-9 is optimized using SIMD Extensions.
- The statements included in the "for"-statement at line-9 is optimized with loop unrolling.
- The statements at line-16, line-17, line-20, line-21, line-22, and line-23 are not optimized using SIMD Extensions.
- The loop by "for"-statement at line-27 is parallelized if its iteration count is greater than or equal to 1000.
- The loop by "for"-statement at line-27 is optimized with software pipelining.
- The loop by "for"-statement at line-27 is parallelized.
- The statements included in the "for"-statement at line-27 is optimized using SIMD Extensions.
- The function called at line-28 is optimized with inline expansion.
- The loop by "for"-statement at line-31 is parallelized if its iteration count is greater than or equal to 1000.
- The loop by "for"-statement at line-31 is optimized using SIMD Extensions.
- The statement at line-32 is not parallelized because the parallelization is not effective.
- The statement at line-33 is not parallelized because the parallelization is not effective.

5.1.2.4 Notes on Information at Compilation (Source List)

The `-Nlst=p` option, `-Nlst=t` option and `-Nsrc` option could output wrong compilation information (optimization information) in the following cases.

- When a function is inline expanded, different optimizations could be applied to each line. In this case, the compiler could output information incorrectly as follows.
 - Output messages redundantly.
 - Output optimization messages inconsistent with compilation information (optimization information).
 - Output no compilation information (optimization information).

In addition, `PREFETCH: N`, which is the number of prefetch instructions in the inlined function, includes all prefetch instructions expanded in multiple lines.

- The compiler applies multiple optimizations such as SIMDization, automatic parallelization, loop fusion, and loop distribution to one loop. In this case, the compiler could output information incorrectly as follows.
 - Output optimization information on line number to an incorrect line.
 - Output compilation information (optimization information) inconsistent with optimization messages.
- Loop unswitching optimization creates a loop in each "TRUE" and "FALSE" blocks of the "if" statement, but the compiler outputs optimization message for only one of the loops. In addition, optimization information on line number may not be output.
- When multiple loops are written in the same line in a program, the compiler outputs optimization information for only one of the loops. Please make sure to write one loop in one line in order to output optimization information.
- Detailed optimization information for a loop which consists of "if-goto" statements is not output. Optimization information on line number could be output to an incorrect line.
- The compiler may generate loops under the following condition. The optimization messages may be output for the generated loops, and compilation information (optimization information) may be output for the generated loops.
 - There is a loop parallelized automatically by `-Karray_private` option or optimization control specifier `array_private`, `first_private` or `last_private`.

5.1.3 Parallelization Messages

This parallelization messages indicate why the parallelization failed, or how the parallelization and optimization are performed.

Using the `-Koptmsg=2` option with the `-Kparallel` option, the parallelization messages can be emitted to the standard error. An example of the parallelization message is shown in following.

Example of Parallelization Messages

```
Parallelization messages
jwd5001p-i  "source-filename", line n1: This loop with loop variable 'i' is parallelized.
jwd6001s-i  "source-filename", line n1: SIMD conversion is applied to this loop with the loop
variable 'i'.
jwd8202o-i  "source-filename", line n1: Loop unrolling expanding 8 times is applied to this loop.
jwd5122p-i  "source-filename", line n2: This loop is not parallelized because a function call
that prevents parallelization exists in the loop.
```

5.1.4 Statistics List

When the program is compiled, the statistics list is put out according to the following options.

- `-Nlst={p|t}`
Specifies to output source list and statistics list to file(s).
- `-Nlst_out=file`
Specifies the filename to output source list and statistics list.

When this option is specified, the -Nlst=p option is also effective.

- -Nsta

Specifies to output statistics list to the standard output.

Note that when the -Nlst or -Nlst_out=*file* option is specified, the statistics list is output to the *file*.

Only effective optimization options are emitted in the statistics list. The options which are not emitted are invalid.

5.1.4.1 Output Format

The statistics list is emitted as shown in the following format.

Source List Format

```
Statistics information
Option information
  Profile file      : Options that specified in compilation profile file[a]
  Environment variable : Options that specified in environment variables to set compiler options[a]
  Command line options : Options that specified in compile command
  Effective options   : Effective optimization options
```

[a] The output is on one line in the order of "<mode common> <mode specific>".

Note that the "Profile file" and "Environment variable" lines are not emitted in the following cases:

- The compilation profile file does not exist, or the compilation profile file size is zero
- Environment variables to set compiler options do not exist

5.1.4.2 Options Which Are Not Output

The statistics list outputs only effective options for optimization.

Options for ld command are not output.

- Options which are not output as follows:

```
-# / -### / -A- / -Aname[(tokens)] / -B{dynamic|static} / -C / -Dname[=tokens] / -E / -H / -Idir /
-Ldir / -Nlst_out=file / -P / -S / -Uname / -Wtool,arg1[,arg2]... / -Yitem,dir / -V / -c / {-help|--
help} / -lname / -o pathname / -shared
```

5.1.4.3 Options Which Are Interpreted by Compiler

The following options are interpreted by the compiler as follows.

For more the details about interpretation, see Section "[2.2.2.5 -K Option](#)".

- -Kfast
- -Knoprefetch
- -Kvisimpact

5.2 Information at Execution

This section describes the information emitted during execution.

5.2.1 Runtime Messages

For details about runtime messages displayed by this system:

- When the message starts with "jwe", refer to "Fortran/C/C++ Runtime Messages".
- For all other messages refer to the reference manual provided with this system.

5.2.2 Trace Back Information

When an error or trap occurs in execution, a trace back map is emitted to standard error if either the environment variable `FLIB_C_MESSAGE` is not set to "NO_MESSAGE" or the `-NRtrap` option is specified.

Using this feature, it is possible to find the path from the main program to the program unit where the error occurred.

This information is obtained from the debug information stored in the heap memory allocated at startup. If the information is not output, the information may be output by setting the size of the heap memory larger than the default value using the environment variable `FLIB_TRACEBACK_MEM_SIZE`. If the information cannot be obtained due to insufficient heap memory, a diagnostic message `jwe1531i` will be output. This diagnostic message is output during error processing, so the order in which the diagnostic messages are output may be incorrect.

For details about the environment variables `FLIB_C_MESSAGE` and `FLIB_TRACEBACK_MEM_SIZE`, see Section "[2.5.1 Environment Variable for Execution](#)".

For details about the `-NRtrap` option, see Section "[2.2.2.6 -N Option](#)".

For details about Trace Back Information, refer to "Fortran User's Guide".

Chapter 6 Language Specifications

This chapter describes the language specifications based on the C language standard and extended under this system, and the support status of the language specifications.

Note that the following syntax description symbols are used in this chapter.

- The nonterminal symbols (syntactical elements) are indicated by italic text. Terminal symbols (literals and elements of character sets) are represented by roman text, including special symbols.
- A colon (:) appearing after a nonterminal symbol indicates that the text after the colon is a definition of that nonterminal symbol.
- Except when prefaced by "one of the following", selectable definitions appear on separate lines.
- Optional symbols that may be omitted are indicated with the subscript *opt*. Therefore, in {*expression*<*opt*>}, the curly brackets indicate the range that may be omitted.

6.1 Language Specifications Extended in This System

6.1.1 long long Type

In addition to the four signed integer types of `signed char`, `short int`, `int` and `long int`, the `long long int` type is added.

Suffixes

integer-suffix:

unsigned-suffix long-word-suffix<*opt*>

long-word-suffix unsigned-suffix<*opt*>

unsigned-suffix long-long-word-suffix<*opt*>

long-long-word-suffix unsigned-suffix<*opt*>

unsigned-suffix:

One of the following

- `u`
- `U`

long-long-word-suffix:

long-word-suffix long-word-suffix

long-word-suffix:

One of the following

- `l`
- `L`

Meaning Rules

The compiler uses the first data type in the following list in which the value can be represented, depending on the size of the constant.

Decimal number with no suffix:

`int`, `long int`, `unsigned long int`, `long long int`, `unsigned long long int`

Octal or hexadecimal number with no suffix:

`int`, `unsigned int`, `long int`, `unsigned long int`, `long long int`, `unsigned long long int`

If the letter `u` or `U` is added as a suffix:

`unsigned int`, `unsigned long int`, `unsigned long long int`

If the letter `l` or `L` is added as a suffix:

`long int`, `unsigned long int`, `long long int`, `unsigned long long int`

If both the letter `u` or `U` and the letter `l` or `L` are added as suffixes:

`unsigned long int`, `unsigned long long int`

If both the letter `l` or `L` and the letter `l` or `L` are added as suffixes:

`long long int`, `unsigned long long int`

Limitations of Type Specifiers

The various lists of type specifiers must belong to one of the following groups. If there are two or more groups in one item, they are delimited with commas. The type specifiers may appear in any order, and may be mixed with other declaration specifiers.

- `long long`, `signed long long`, `long long int`, `signed long long int`
- `unsigned long long`, `unsigned long long int`

Code Samples:

```
long long int lli=10LL;
```

```
unsigned long long int ulli=10ULL;
```

6.1.2 The pragma Directive

The pragma directive directs the operation of the processing system.

Meaning

#pragma ident *string-literal new-line*

Directs the processing system to add the string in *string-literal* to the object file as a comment. This is a #pragma directive that has the same meaning as the #ident directive. Typically, this is used to add version control information to the object file.

#pragma weak *identifier new-line*

Directs the processing system to define *identifier* in the object file as a weak symbol.

#pragma weak *identifier1=identifier2 new-line*

Directs the processing system to define *identifier1* in the object file as a weak symbol having the same value and type as *identifier2*.

#pragma unknown_control_flow (*list-of-identifiers*) new-line

Informs the processing system that the function specified in *list-of-identifiers* has control flow unknown to the processing system.

#pragma redefine_extname *<old function name> <new function name> new-line*

Directs the processing system to replace the old external function name with the new external function name in the object code.

6.1.3 The ident Directive

The ident directive specifies that comments are to be added to the C program.

Meaning

#ident *string-literal new-line*

Directs the processing system to add the specified *string-literal* to the ".comment section" of the object file. The ".comment section" is not loaded into memory when the program is executed.

#ident *preprocessor-token new-line*

Directs the processing system to process the specified *preprocessor-token* as normal text, and the various identifiers currently defined as macro names are replaced with a list of their replacements. All #ident directives after replacement must match their format prior to replacement.

6.1.4 The assert Directive

The assert directive defines predicate names and assigns assertion tokens to predicate names.

Meaning

#assert *identifier* *assertion-specifier*<opt> *new-line*

Defines that the *identifier* following the #assert directive is a predicate name. In addition, the list of identifiers in the *assertion-specifier* following the predicate name are defined to be assertion tokens.

The #assert directive defines predicate names and assigns assertion tokens to predicate names.

An #assert directive with no list of identifiers is effective until a corresponding #unassert directive appears.

Various Identifiers of the # Operator Expression

The various identifiers of the # operator expression correspond to the predicate names and assertion tokens specified in the #assert directive, respectively. If an assertion token is assigned to a predicate name, it evaluates to 1; otherwise it evaluates to 0. For example, #system(unix) becomes 1.

The following assertions are predefined by the processing system (predefined assertions):

- #assert system(unix)

Code Samples:

```
#assert langlevel(ansi)
#assert langlevel(sysv)
```

The above directives define the predicate langlevel and assign the assertions ansi and sysv to the predicate.

```
#assert float
```

The above directive defines the predicate float. It has no assertions.

6.1.5 The unassert Directive

The unassert directive cancels the assignment of predicate names and assertion tokens.

Meaning

#unassert *identifier* *assertion-specifier*<opt> *new-line*

Cancels the assignment of assertion tokens. An #unassert directive with no list of *identifiers* (*assertion-specifiers*) cancels the assignment of all assertion tokens to predicate names.

If the specified *identifier* is not a predicate name, the #unassert directive is ignored. Also, if an unassigned assertion token is specified, the #unassert directive is ignored.

6.1.6 Predefined Macro Names

Some of the predefined macros are changed by the compiler option -std=*level*.

The following tables show the values of some predefined macros. To display all predefined macros, specify the compiler options -E and -dM.

Table 6.1 Predefined Macros whose value is changed by the compiler option -std=level

Identifier	The level of language specifications specified with the compiler option -std= <i>level</i>					
	c89	gnu89	c99	gnu99	c11	gnu11
__STDC_NO_ATOMICS__	-		-		1	
__STDC_NO_THREADS__	-		-		1	
__STDC_VERSION__	-		199901L		201112L	
__STRICT_ANSI__	1	-	1	-	1	-

Identifier	The level of language specifications specified with the compiler option -std= <i>level</i>					
	c89	gnu89	c99	gnu99	c11	gnu11
linux	-	1	-	1	-	1
unix	-	1	-	1	-	1

-: Undefined

Table 6.2 Predefined Macros whose value is not changed by the compiler option -std=level

Identifier	Value
_LP64	1
_OPENMP (*1)	201107
_REENTRANT (*2)	1
__ARM_ARCH	8
__ASSEMBLER__ (*3)	1
__DATE__	The date of compilation (in the format of the asctime function)
__ELF__	1
__EXCEPTIONS (*4)	1
__FCC_major__	The major version number of the compiler
__FCC_minor__	The minor version number of the compiler
__FCC_patchlevel__	The patch level of the compiler
__FCC_version__	The string which represents the version of the compiler
__FILE__	Source file name
__FUJITSU	1
__GNUC__	6
__GNUC_MINOR__	1
__GNUC_PATCHLEVEL__	0
__LINE__	Line number of source file
__LP64__	1
__MT__ (*5)	1
__OPTIMIZE__ (*6)	1
__PIC__	1 (*7)
	2 (*8)
__PRAGMA_REDEFINE_EXTNAME	1
__PTRDIFF_TYPE__	long int
__SIGNED_CHARS__	1 (*9)
	- (*10)
__SIZE_TYPE__	long unsigned int
__STDC__	1
__STDC_HOSTED__	1
__TIME__	The time of compilation (in the format of the asctime function)
__USER_LABEL_PREFIX__	null string

Identifier	Value
<code>__WCHAR_TYPE__</code>	<code>int (*11)</code>
<code>__aarch64__</code>	1
<code>__linux</code>	1
<code>__linux__</code>	1
<code>__pic__</code>	1 (*7)
	2 (*8)
<code>__unix</code>	1
<code>__unix__</code>	1

-: Undefined

*1) When the compiler option `-Kopenmp` or `-fopenmp` is set

*2) When the compiler option `-mt` or `-pthread` is set

*3) When the compiler option `-x assembler-with-cpp` is set, or when the suffix of the input file is `".S"`

*4) When the compiler option `-Nexceptions` or `-fexceptions` is set

*5) When the compiler option `-mt` is set

*6) When the compiler option `-O1` or higher is set

*7) When the compiler option `-fpic` or `-fpie` is set

*8) When the compiler option `-fPIC` or `-fPIE` is set

*9) When the compiler option `-fsigned-char` is set

*10) When the compiler option `-funsigned-char` is set

*11) This value is different from Clang Mode, and unintended behavior may occur when you execute a program linking an object generated in Trad Mode and an object generated in Clang Mode.

6.2 Support Status of Language Specifications

This system supports the following C language standard:

- C89
- C99
- C11

6.2.1 C99 Specifications

The following C99 specifications are not supported by this system.

Array declarators

In the C99 specification, the `static` storage class specifier and the type-qualifiers can be used to declare an array type.

However the `static` storage class specifier and the `const` and `volatile` type-qualifiers cannot be used to declare an array type by the compile command.

#pragma STDC

`#pragma STDC` is not supported. The compile command ignores the pragmas as unknown.

String concatenation between an adjacent string literal token and a wide string literal token

Currently, the compile command does not support the string concatenation between an adjacent string literal token and a wide string literal token.

In C99, the string concatenation is performed in translation phase 6, and then in translation phase 7, the concatenated multibyte sequence is converted to a `wchar_t` array by `mbstowcs`. In this specification the portability of the user program is lost. In C89, this usage causes a syntax error. In C11, if any of the tokens has an encoding prefix, all tokens are treated as having the same prefix.

Floating point environment

floating point environment (`fenv.h`) is not supported.

Type specifier `_Imaginary`

Type specifier `_Imaginary` is not supported.

6.2.2 C11 Specifications

The C11 specifications supported by this system are as follows.

Table 6.3 Support Status of C11 Specifications

Language Features	Support
<code>_Alignas</code> , <code>_Alignof</code> , <code>max_align_t</code> , <code>stdalign.h</code>	Yes
<code>_Atomic</code> , <code>stdatomic.h</code>	Yes ^[a]
<code>_Generic</code>	Yes
<code>_Noreturn</code> , <code>stdnoreturn.h</code>	Yes
<code>_Static_assert</code>	Yes
<code>_Thread_local</code>	Yes
Anonymous struct and union	Yes
Typedef redefinition	Yes
New macros in <code>float.h</code>	Yes
Macros for Complex values	Yes
Unicode strings	Yes

[a] When `-Nclang` is set, this feature is supported. When `-Nnoclang` is set, defines the `__STDC_NO_ATOMICS__` macro.

Chapter 7 Notes on Linking with Different Languages and Trad/Clang Modes

This chapter describes notes about linking with object programs in different programming languages and Trad/Clang Modes.

7.1 Compile Commands and Required Options when Linking

In this system, there are two types of compile commands for the Fortran, C, and C++ languages, a cross compiler and a native compiler. In addition, the compiler commands are used depending on whether the program uses the MPI library as shown in the table below.

The following description uses the compile command of the cross compiler (No use of MPI libraries). When using the MPI library or native compiler, replace the compile commands in the table below.

Programming Language	MPI Library	Cross Compiler	Native Compiler
Fortran	Not Use	frtpx	frt
	Use	mpifrtpx	mpifrt
C	Not Use	fccpx	fcc
	Use	mpifccpx	mpifcc
C++	Not Use	FCCpx	FCC
	Use	mpiFCCpx	mpiFCC

Object programs can be linked in a combination of Fortran, C Trad Mode, C Clang Mode, C++ Trad Mode, and C++ Clang Mode.

Depending on the combination of object programs, some compilation commands can not use at linking. For object programs combinations, and compilation commands/required options at linking, see "[Table 7.1 Object programs combinations, compilation commands, and required options](#)". For more information on the compile commands and options, see the user's guide for each language.

Table 7.1 Object programs combinations, compilation commands, and required options

Object to Link						COARRAY	Compile Commands/Required Options
Fortran	C trad	C clang	C++ trad	C++ clang			
				Use libc++ for C++ Standard Library	Use libstdc++ for C++ Standard Library		
*	*	-	-	-	-	Not Use	frtpx or fccpx --linkfortran
						Use	frtpx -Ncoarray or fccpx --linkcoarray
*	-	*	-	-	-	Not Use	frtpx or fccpx -Nclang --linkfortran
						Use	frtpx -Ncoarray or fccpx -Nclang --linkcoarray
*	-	-	*	-	-	Not Use	frtpx --linkstl=libfjc++ or FCCpx --linkfortran

Object to Link						COARRAY	Compile Commands/Required Options
Fortran	C trad	C clang	C++ trad	C++ clang			
				Use libc++ for C++ Standard Library	Use libstdc++ for C++ Standard Library		
						Use	frtpx --linkstl=libfjc++ -Ncoarray or FCCpx --linkcoarray
*	-	-	-	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray
				-	*	Not Use	frtpx --linkstl=libstdc++ or FCCpx -Nclang --linkfortran
						Use	frtpx --linkstl=libstdc++ -Ncoarray or FCCpx -Nclang --linkcoarray
-	*	*	-	-	-	-	fccpx -Nclang
-	*	-	*	-	-	-	FCCpx
-	*	-	-	*	-	-	FCCpx -Nclang
				-	*		FCCpx -Nclang -stdlib=libstdc++
-	-	*	*	-	-	-	FCCpx
-	-	*	-	*	-	-	FCCpx -Nclang
				-	*		FCCpx -Nclang -stdlib=libstdc++
-	-	-	*	*	-	-	FCCpx -Nclang -stdlib=libc++ (Refer to "Note 1")
				-	*		Can not to be linked. (Refer to "Note 2")
*	*	*	-	-	-	Not Use	frtpx or fccpx -Nclang --linkfortran
						Use	frtpx -Ncoarray or fccpx -Nclang --linkcoarray
*	*	-	*	-	-	Not Use	frtpx --linkstl=libfjc++ or FCCpx --linkfortran
						Use	frtpx --linkstl=libfjc++ -Ncoarray or FCCpx --linkcoarray
*	*	-	-	*	-	Not Use	frtpx --linkstl=libc++ or

Object to Link						COARRAY	Compile Commands/Required Options
Fortran	C trad	C clang	C++ trad	C++ clang			
				Use libc++ for C++ Standard Library	Use libstdc++ for C++ Standard Library		
							FCCpx -Nclang --linkfortran -stdlib=libc++
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray
				-	*	Not Use	frtpx --linkstl=libstdc++ or FCCpx -Nclang --linkfortran
						Use	frtpx --linkstl=libstdc++ -Ncoarray or FCCpx -Nclang --linkcoarray
*	-	*	*	-	-	Not Use	frtpx --linkstl=libfjc++ or FCCpx --linkfortran
						Use	frtpx --linkstl=libfjc++ -Ncoarray or FCCpx --linkcoarray
*	-	*	-	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray
				-	*	Not Use	frtpx --linkstl=libstdc++ or FCCpx -Nclang --linkfortran
						Use	frtpx --linkstl=libstdc++ -Ncoarray or FCCpx -Nclang --linkcoarray
*	-	-	*	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++ (Refer to "Note 1")
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray (Refer to "Note 1")
				-	*	Not Use	Can not to be linked. (Refer to "Note 2")
						Use	

Object to Link						COARRAY	Compile Commands/Required Options
Fortran	C trad	C clang	C++ trad	C++ clang			
				Use libc++ for C++ Standard Library	Use libstdc++ for C++ Standard Library		
-	*	*	*	-	-	-	FCCpx
-	*	*	-	*	-	-	FCCpx -Nclang -stdlib=libc++
				-	*		FCCpx -Nclang -stdlib=libstdc++
-	*	-	*	*	-	-	FCCpx -Nclang -stdlib=libc++ (Refer to "Note 1")
				-	*		Can not to be linked. (Refer to "Note 2")
-	-	*	*	*	-	-	FCCpx -Nclang -stdlib=libc++ (Refer to "Note 1")
				-	*		Can not to be linked. (Refer to "Note 2")
*	*	*	*	-	-	Not Use	frtpx --linkstl=libfjc++ or FCCpx --linkfortran
						Use	frtpx --linkstl=libfjc++ -Ncoarray or FCCpx --linkcoarray
*	*	*	-	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray
				-	*	Not Use	frtpx --linkstl=libstdc++ or FCCpx -Nclang --linkfortran
						Use	frtpx --linkstl=libstdc++ -Ncoarray or FCCpx -Nclang --linkcoarray
*	*	-	*	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++ (Refer to "Note 1")
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray (Refer to "Note 1")
				-	*	Not Use	Can not to be linked. (Refer to "Note 2")
						Use	

Object to Link						COARRAY	Compile Commands/Required Options
Fortran	C trad	C clang	C++ trad	C++ clang			
				Use libc++ for C++ Standard Library	Use libstdc++ for C++ Standard Library		
*	-	*	*	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++ (Refer to "Note 1")
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray (Refer to "Note 1")
				-	*	Not Use	Can not to be linked. (Refer to "Note 2")
						Use	
-	*	*	*	*	-	-	FCCpx -Nclang -stdlib=libc++ (Refer to "Note 1")
				-	*		Can not to be linked. (Refer to "Note 2")
*	*	*	*	*	-	Not Use	frtpx --linkstl=libc++ or FCCpx -Nclang --linkfortran -stdlib=libc++ (Refer to "Note 1")
						Use	frtpx --linkstl=libc++ -Ncoarray or FCCpx -Nclang -stdlib=libc++ --linkcoarray (Refer to "Note 1")
				-	*	Not Use	Can not to be linked. (Refer to "Note 2")
						Use	

7.1.1 Linking C++ Trad Mode with C++ Clang Mode

Whether object programs generated in C++ Trad Mode and object programs generated in C++ Clang Mode can be linked depends on C++ Standard Library used at compilation.

C++ Trad Mode	C++ Clang Mode	Compile Commands/Required Options
(libc++ use only)	Use libc++ for C++ Standard Library	FCCpx -Nclang -stdlib=libc++ (Refer to "Note 1")
	Use libstdc++ for C++ Standard Library	Can not to be linked. (Refer to "Note 2")

Note 1

When all of the following conditions are met, the operation may become indefinite and can not be guaranteed.

1. An object generated in Clang Mode and an object generated in Trad Mode have an interface between functions.
2. An argument or return value of the function in condition 1 is a class type.

Note 2

The default C++ Standard Library that C++ Clang Mode uses is libstdc++.

Specify libc++ instead of the default libstdc++ for C++ Standard Library used in Clang Mode.

7.1.2 Linking C++ Clang Mode with C++ Clang Mode

Whether object programs generated in C++ Trad Mode and object programs generated in C++ Clang Mode can be linked depends on C++ Standard Library used at compilation.

C++ Standard Library used in compilation		Compile Commands/Required Options
libc++	libc++	FCCpx -Nclang -stdlib=libc++
libc++	libstdc++	Can not to be linked.
libstdc++	libstdc++	FCCpx -Nclang -stdlib=libstdc++ (Note that the -stdlib = libstdc++ option can be omitted because it is the default.)

7.1.3 Linking MPI Object Programs and Use Profiler

Specify the following option at linkage:

- Use mpifrtpx command : -lfjprofmpi option
- Use mpifccpx command or mpiFCCpx command : -lfjprofmpif option

7.1.4 Objects Generated in Clang Mode with -flto Option

Objects generated with the "-Nclang -flto" options are in the format used internally by LLVM (different format from normal objects) and can only be linked in Clang Mode.

When including an object generated with the "-Nclang -flto" options at mixed language programming, specify the "-Nclang -flto" options at linking.

7.2 Linking with C++

Note the following:

- To call a C++ function from a C function, the C++ function must be declared with linkage to C specified (via `extern "C"`) in the C++ program.
- To call a C function from a C++ function, the C function must be declared with linkage to C specified (via `extern "C"`) in the C++ program.
- The user can specify any type permitted under C for the arguments and return values of a function to be called in a C specified function specified with `extern "C"`.

Example 1: Passing control to the C++ program first

C++ source: cplusplusmain.cc

```
#include <iostream>

extern "C" {
    void cfunc(short int);
    int cpfunc(int);
}

int cpfunc(int i) {
    std::cout << "C++: cpfunc called, i=" << i << std::endl;
    return i;
}
```

```
int main() {
    std::cout << "C++ main()" << std::endl;
    cfunc(10);
    return 0;
}
```

C source: csub.c

```
#include <stdio.h>

int cpfunc(int);

void cfunc(short int si) {
    printf("C:  cfunc called, si=%d\n", si);
    cpfunc(si);
}
```

Compiling and linking:

```
$ fccpx -c csub.c
$ FCCpx csub.o cplusplusmain.cc
```

Execution:

```
$ ./a.out
```

Result:

```
C++ main()
C:  cfunc called, si=10
C++: cpfunc called, i=10
```

Example 2: Passing control to the C program first

C++ source: cplusplus.cc

```
#include <iostream>

extern "C" {
    void cfunc(short int);
    int cpfunc(int);
}

int cpfunc(int i) {
    std::cout << "C++: cpfunc called, i=" << i << std::endl;
    cfunc(i);
    return i;
}
```

C source: cmain.c

```
#include <stdio.h>

int cpfunc(int);

int main() {
    printf("C main()\n");
    cpfunc(10);
    return 0;
}

void cfunc(short int si) {
    printf("C:  cfunc called, si=%d\n", si);
}
```

Compiling and linking:

```
$ fccpx -c cmain.c
$ FCCpx cmain.o cplussub.cc
```

Execution:

```
$ ./a.out
```

Result:

```
C main()
C++: cpfunc called, i=10
C:  cfunc called, si=10
```

7.3 Linking with Fortran

Note the following:

- If the program to which control is first passed is in C, the function name must be "MAIN__" or "main". For more details, please refer to "Fortran User's Guide".
- In the following cases, the return value of Fortran is set the return value of executable program. For detail of the return value of Fortran, refer to "Fortran User's Guide".
 - The program to which control is first passed is Fortran program.
- Append "_" to the end of the C function name called from the Fortran and to the Fortran function name called from C.
- Other rules are the same as those for interlanguage linkage between C and Fortran. Please refer to "Fortran User's Guide".

When using the COARRAY specification, specify the --linkcoarray option instead of the --linkfortran option. For more information on the COARRAY specification, see "Fortran User's Guide Additional Volume COARRAY".

Example 1: Passing control to the Fortran program first

Fortran source: fortranmain.f95

```
print *, "Fortran: main program"
call cfunc(10)
end
```

C source: csub.c

```
#include <stdio.h>

int cfunc_(int *p) {
    printf(" C: cfunc_ called, *p=%d\n", *p);
    return *p;
}
```

Compiling and linking:

- When using the compile command for the Fortran language at linking

```
$ fccpx -c csub.c
$ frtpx csub.o fortranmain.f95
```

- When using the compile command for the C language at linking

```
$ frtpx -c fortranmain.f95
$ fccpx --linkfortran csub.c fortranmain.o
```

Execution:

```
$ ./a.out
```

Result:

```
Fortran: main program
C: cfunc_ called, *p=10
```

Example 2: Passing control to the C program first (MAIN__)

Fortran source: fortransub.f95

```
integer function func(x)
integer*4 x
print *, "Fortran: func() called"
call cfunc(x)
func=x
end
```

C source: cmain.c

```
#include <stdio.h>

int func_(int *);

int cfunc_(int *p) {
    printf(" C: cfunc_ called. *p=%d\n", *p);
    return *p;
}

int MAIN__() {
    int i=10;
    printf(" C MAIN__()\n");
    func_(&i);
    return 0;
}
```

Compiling and linking:

- When using the compile command for the Fortran language at linking

```
$ fccpx -c cmain.c
$ frtpx cmain.o fortransub.f95
```

- When using the compile command for the C language at linking

```
$ frtpx -c fortransub.f95
$ fccpx --linkfortran cmain.c fortransub.o
```

Execution:

```
$ ./a.out
```

Result

```
C MAIN__()
Fortran: func() called
C: cfunc_ called. *p=10
```

Example 3: Passing control to the C program first (main)

Fortran source: fortransub.f95

```
integer function func(x)
integer*4 x
print *, "Fortran: func() called"
call cfunc(x)
func=x
end
```

C source: cmain.c

```
#include <stdio.h>

int func_(int *);

int cfunc_(int *p) {
    printf(" C: cfunc_ called. *p=%d\n", *p);
    return *p;
}

int main() {
    int i=10;
    printf(" C main()\n");
    func_(&i);
    return 0;
}
```

Compiling and linking:

- When using the compile command for the Fortran language at linking

```
$ fccpx -c cmain.c
$ frtpx -mlcmain=main cmain.o fortransub.f95
```

- When using the compile command for the C language at linking

```
$ frtpx -c fortransub.f95
$ fccpx --linkfortran cmain.c fortransub.o
```

Execution:

```
$ ./a.out
```

Result:

```
C main()
Fortran: func() called
C: cfunc_ called. *p=10
```

Chapter 8 Debugging Functions

A program does not work as expected, for example, it needs to check the cause and to correct.

This chapter describes the debugging functions provided in this system that help checking the cause.

8.1 Functions for Debugging

This section describes the check functions used to debug C programs.

The `-Nquickdbg` option is used for debugging during program execution. It provides the following checks:

- Subscript range checks (see Section "[8.1.1 Subscript Range Checks \(subchk Function\)](#)")
- Heap checks (see Section "[8.1.2 Heap Memory Checks \(heapchk Function\)](#)")

The `-Nquickdbg=inf_detail` and the `-Nquickdbg=inf_simple` options output the diagnostic messages when any error is detected. For the `subchk` function, these options control execution performance and contents of the diagnostic messages.

When an error is detected, if the `-Nquickdbg=inf_detail` option is enabled, a message to identify the cause, line number where the error occurred and the variable name that caused the error are displayed as diagnostic message and the execution is continued.

When an error is detected, if the `-Nquickdbg=inf_simple` option is enabled, a message indicating the error and the line number where the error occurred are displayed as diagnostic message and the execution is terminated. However, if the compiler option `-Nnoline` is enabled, the value of line number is not guaranteed.

By limiting the information included in diagnostic messages, the `-Nquickdbg=inf_simple` option has less impact on execution performance than the `-Nquickdbg=inf_detail` option.

For Heap Memory Checks, even if either of the `-Nquickdbg=inf_detail` or the `-Nquickdbg=inf_simple` option is specified, the message and the line number where the error occurred are displayed as diagnostic messages and the execution is continued. Even if which option is specified, it becomes an equal execution performance.

See Section "[2.2.2.6 -N Option](#)", for details of `-Nquickdbg`.

8.1.1 Subscript Range Checks (subchk Function)

If the `-Nquickdbg=subchk` option is set, the validity of the referenced subscript range is checked against the declared array size when arrays are referenced. If the subscript is not within the range that applied when the array was declared, a diagnostic message is output. When `-Nquickdbg=inf_detail` is set, the diagnostic message is `jwe1601i-w`. When `-Nquickdbg=inf_simple` is set, the diagnostic message is `jwe1606i-u`.

This check is not performed in the following cases:

- An array specified as a parameter of function.
- An array which size is not determined at compilation (Variable-Length Array, Flexible Array, Arrays of Length Zero)
- An expression used as index which has apprehension of occurrence of side effect to refer array.
- Statements and declarations in an expression ^[a] used as index which has apprehension of occurrence of side effect to refer array.

[a] "Statements and Declarations in Expressions" is a part of GNU C Extensions.

8.1.2 Heap Memory Checks (heapchk Function)

If the `-Nquickdbg=heapchk` option is set, the memory allocated by the `malloc`, `calloc`, `realloc`, `valloc`, `posix_memalign`, and `memalign` functions is checked as follows.

- The memory release check and the buffer overrun check are performed when heap memory is released.
- The memory leak check is performed when the program ends.

The check items are explained below.



If memory allocated using functions other than malloc, calloc, realloc, valloc, posix_memalign, and memalign is released, diagnostic message jwe1602i-w or jwe1603i-w may be output.

8.1.2.1 Memory Release Check

When heap memory is released, the following checks are performed for the memory being released:

- Release of memory that has already been released
- Invalid memory release

If a duplicate release of memory is detected, a diagnostic message (jwe1602i-w) is output. If an invalid memory release is detected, a diagnostic message (jwe1603i-w) is output.

8.1.2.2 Buffer Overrun Check

When heap memory is released, the function checks whether or not memory outside the allocated memory block is being written to.

The buffer overrun check performs a write check for the followed 8 bytes of heap memory. If buffer overrun is detected, a diagnostic message (jwe1604i-w) is output.

However, if the followed 8 byte areas of heap memory have the following value, an invalid area write is not detected even if buffer overrun has occurred:

- 0x8b8b8b8b8b8b8b8b

The above value can be changed by specifying the FLIB_HEAPCHK_VALUE environment variable.

```
FLIB_HEAPCHK_VALUE hex
```

For buffer overrun checks, the values used for the check are changed. By specifying hex, hexadecimal values can be specified. The valid range of values is 00<=hex<=FF. If the environment variable is not specified or if hex values are disabled, the system default value takes effect. The default value is 8B.

An example of specifying the FLIB_HEAPCHK_VALUE environment variable is shown below.

```
$ FLIB_HEAPCHK_VALUE=8C
$ export FLIB_HEAPCHK_VALUE
```

In this example, 0x8c8c8c8c8c8c8c8c is applied as the check value used in the outside area write check.

8.1.2.3 Memory Leak Check

When a program ends, the function checks whether or not heap memory remains without being released. If unreleased heap memory remains, a diagnostic message (jwe1605i-w) is output.

8.2 Debugging Programs for Abend

When the -NRtrap option is specified, the signal is caught if an abnormal termination event (hereafter called an abend) occurs during execution of a program. And then the information is output to help the user determine the cause of the abend.

When the -NRnotrap option is enabled, the information is not output.

For details about the -NRtrap and -NRnotrap option, see Section "2.2.2.6 -N Option".

8.2.1 Causes of Abend

"Table 8.1 caught signals and corresponding signal codes" lists the caught signals (*) and the corresponding signal codes.

*: Only signals supported by the Fujitsu CPU A64FX with Arm architecture are targeted. Therefore, enabling the -NRtrap option will not detect unsupported signals. See Section "2.2.2.6 -N Option" for more information about the -NRtrap option.

If the -NRtrap option is not specified, these signals are not caught.

Table 8.1 caught signals and corresponding signal codes

Signal number	Meaning of signal number	Signal code		Meaning of signal code
SIGILL(04)	Incorrect instruction executed	1	ILL_ILLOPC	Illegal opcode
		2	ILL_ILLOPN	Illegal operand
		3	ILL_ILLADR	Illegal addressing mode
		4	ILL_ILLTRP	Illegal trap
		5	ILL_PRVOPC	Illegal privileged opcode
		6	ILL_PRVREG	Illegal privileged register
		7	ILL_COPROC	Coprocessor error
		8	ILL_BADSTK	Internal stack error
SIGFPE(08)	Floating-point exception ^[a]	3	FPE_FLTDIV	Floating-point divide by zero
		4	FPE_FLTOVF	Floating-point overflow
		5	FPE_FLTUND	Floating-point underflow ^[b]
		7	FPE_FLTINV	Invalid floating-point operation
		-	-	Floating-point exception
SIGBUS(10)	Storage protection exception	1	BUS_ADRALN	Invalid address alignment
		2	BUS_ADRERR	Non-existent physical address
		3	BUS_OBJERR	Object-specific hardware error
SIGSEGV(11)	Segmentation exception	1	SEGV_MAPERR	Address not mapped to object
		2	SEGV_ACCERR	Invalid permissions for mapped object
SIGXCPU(30)	CPU time interrupt	-	-	-

[a]: Note that FPE_INTDIV (the integer divide by zero) is not detected in the Fujitsu CPU A64FX with Arm architecture. See Section "C.2.9 Integer Division Exception when the Divisor Is Zero" for details.

[b] This signal is caught when the -NRtrap option and environment variable FLIB_EXCEPT=u is specified.

8.2.2 Information Generated when an Abend Occurs

Depending on the signal, the information is written to the standard error as explained in the following sections.

8.2.2.1 Information Generated for a General Abend

The information generated for a general abend is follows. This information is generated when SIGILL, SIGBUS, or SIGSEGV is caught. A trace back map is printed after this information.

```
jwe0019i-u The program was terminated abnormally with signal number SSSSSS.
      signal identifier = NNNNNNNNNN, (Detailed information.)
```

SSSSSS	SIGILL, SIGBUS, or SIGSEGV
NNNNNNNNNN	Signal code for the abend cause (hexadecimal)
(Detailed information.)	Detailed information of Signal code NNNNNNNNNN

When it is strong possibility that the cause of SIGBUS or SIGSEGV is stack-overflow, add the following information.

```
The cause of this exception may be stack-overflow.
```

8.2.2.2 Information of SIGXCPU

The output when SIGXCPU is detected is follows.

```
jwe0017i-u The program was terminated with signal number SIGXCPU.
```

8.2.2.3 Information when an Abend Occurs Again during Abend Processing

The output when an abend occurs again during abend processing is follows.

```
jwe0020i-u An error was detected during an abnormal termination process.
```

8.3 Hook Function

This section describes the hook function which calls the user-defined function by specified location in a C program, or by regular time interval.

A user program can be monitored via the user-defined function which can output trace information and specific variables values.

8.3.1 User-Defined Function Format

The user-defined function name is fixed (`user_defined_proc`).

Define the user-defined function in the format shown below.

- format:

```
#include "fjhook.h"
void user_defined_proc(int *FLAG, char *NAME, int *LINE, int *THREAD)
```

- Arguments:

- FLAG: Indicate the calling source for the user-defined function.

= 0: Program entry

= 1: Program exit

= 2: function entry

= 3: function exit

= 4: Parallel region (OpenMP or automatic parallelization) entry

= 5: Parallel region (OpenMP or automatic parallelization) exit

= 6: Regular time interval

= 7 to 99: Reserved for system

>= 100: Can be used by the user

- NAME: Indicate the calling source function name.

NAME can be referenced only if the FLAG is 2, 3, 4, 5, or 100 or greater.

- LINE: Indicate the calling source line number.

LINE can be referenced only if the FLAG is 2, 3, 4, 5, or 100 or greater.

- THREAD: With thread parallelization using OpenMP or automatic parallelization, indicate the number of the thread that calls the user-defined function.

THREAD can be referenced only if the FLAG is 2, 3, 4, 5, or 100 or greater.

8.3.2 Notes on Hook Function

The following notes apply when the hook function is used:

- The function name "user_defined_proc" must not be used for any other purpose.
- Operation is not guaranteed if the user-defined function is not defined.
- Values must not be set in the user-defined function arguments.
- If the -Nnoline compile option is set, the LINE argument value is not guaranteed.
- In thread parallelized programs, the user-defined function may be called from multiple threads.
- If the setjmp or longjmp function is called, or if an exception is caught, the user-defined function may not be called.
- The entry and exit of user-defined function except in the case of calling from any location in the program must not be calling source for the user-defined function.
- Operation is not guaranteed if the exit (3) function is executed directly or indirectly in the user-defined function.
- The OpenMP specifications which are available in the user-defined function are the following OpenMP functions.
 - omp_get_active_level
 - omp_get_ancestor_thread_num
 - omp_get_dynamic
 - omp_get_level
 - omp_get_max_active_levels
 - omp_get_max_threads
 - omp_get_nested
 - omp_get_num_procs
 - omp_get_num_threads
 - omp_get_proc_bind
 - omp_get_schedule
 - omp_get_thread_limit
 - omp_get_thread_num
 - omp_get_wtick
 - omp_get_wtime
 - omp_in_final
 - omp_in_parallel
 - omp_set_dynamic
 - omp_set_max_active_levels
 - omp_set_nested
 - omp_set_num_threads
 - omp_set_schedule

8.3.3 Calling the User-Defined Function from a Specified Location

If the -Nhook_func option is set when the object program and executable program are created, the user-defined function can be called from the following locations:

- Program entry and exit
- Function entry and exit

If the -Kopenmp or -Kparallel option is enabled, user-defined function can also be called from the following locations:

- Parallel region (OpenMP or automatic parallelization) entry and exit

When the -Nhook_func is set, operation is as follows for Fortran, and C linked programs:

- If user-defined subroutine/function is defined within the Fortran program and the C program respectively, the user-defined subroutine defined in the Fortran program is called from the Fortran program, and the user-defined function defined in the C program is called from the C program.
- If user-defined subroutine/function is defined within only the Fortran program or only the C program, the defined user-defined subroutine/function is called from both the Fortran program and the C program.

An example using the hook function with location specified is shown below.

User-defined function example program:

```
#include <stdio.h>
#include "fjhook.h"

void sub() {
    printf("SUB\n");
}

int main() {
    printf("HELLO\n");
    sub();
}

void user_defined_proc(int *FLAG, char *NAME, int *LINE, int *THREAD) {
    switch(*FLAG){
        case 0:
            printf("PROGRAM START\n");
            break ;
        case 1:
            printf("PROGRAM END\n");
            break ;
        case 2:
            printf("PROC START: %s LINE: %d\n",NAME,*LINE);
            break ;
        case 3:
            printf("PROC END: %s LINE: %d\n",NAME,*LINE);
            break ;
    }
}
```

Executable program creation

```
$ fccpx -Nhook_func test.c
```

Execution results

```
PROGRAM START
PROC START: main LINE: 8
HELLO
PROC START: sub LINE: 4
SUB
PROC END: sub LINE: 6
PROC END: main LINE: 11
PROGRAM END
```

8.3.3.1 Notes on Calling from a Specified Location

The following notes apply when the `-Nhook_func` option is set:

- If a function with few computations is called repeatedly, calling the user-defined function from a function entry/exit may affect execution performance.
- If a parallel region with few computations is executed repeatedly, calling the user-defined function from a parallel region entry/exit may affect execution performance.

8.3.4 Calling the User-Defined Function at Regular Time Interval

If the `-Nhook_time` option is set at linking, the user-defined function is called by regular user time interval.

The calling interval is specified using the environment variable `FLIB_HOOK_TIME`. If it is not specified, the user-defined function is called once every minute.

See Section "[2.5.1 Environment Variable for Execution](#)" for information about the environment variable `FLIB_HOOK_TIME`.

8.3.4.1 Notes on Calling by Regular Time Interval

Note the following when the `-Nhook_time` option is set:

- The environment variable `FLIB_HOOK_TIME` must not be changed while program execution is in progress.
- Operation is not guaranteed if a value outside the range 0 to 2147483647 is specified in the environment variable `FLIB_HOOK_TIME`.
- Calling the user-defined function may affect execution performance.

Moreover, the following restrictions are applied because calling by regular interval uses asynchronization signal to call the user-defined function.

- No functions excluding `async-signal-safe` functions can be used in the user-defined function.
- No signal handler which handles `SIGVTALRM` must be defined if the `-Nhook_time` option is set.
- When the `-Nhook_time` option is set, `setitimer(2)` must not be used.
- Only volatile `sig_atomic_t` type variables are guaranteed for global variable references by the user-defined function.
- The profiler must not be used for the executable program created with the `-Nhook_time` option.

8.3.5 Calling from Any Location in the Program

The user-defined function can be called from any location in a program by specifying arguments (`FLAG:>=100, NAME, LINE, THREAD`).

Chapter 9 Clang Mode

This chapter describes Clang Mode and the difference with Trad Mode.

9.1 From Compilation to Execution

9.1.1 Compile Command

This section describes the compile command.

9.1.1.1 Syntax of the Compile Command

Clang Mode is available when `-Nclang` option is set.

The format of the compile command is as follows:

Table 9.1 Format of the Compile Command

Command Name		Operands
Cross compiler	<code>fccpx</code>	<code>[_option-list-including-Nclang]_filename-list</code>
Native compiler	<code>fcc</code>	<code>[_option-list-including-Nclang]_filename-list</code>

_: At least one blank is required.



Information

There is no restriction on the order in which *option-list-including-Nclang* and *filename-list* are specified.

For example, *option-list-including-Nclang* may be specified together.

9.1.1.2 Input Files for the Compile Command

The file types that can be specified as input files for the compile command are as follows:

Table 9.2 Format of Input Files

File Type	File Suffix	Passed To
Headers	<code>.h</code>	Preprocessor ^[a]
	<code>.H</code>	
C source file	<code>.c</code>	Preprocessor and Compiler
	<code>.i</code>	
Assembler source file	<code>.s</code>	Assembler
Assembler source file that must be preprocessed	<code>.S</code>	Preprocessor and Assembler
Object file	<code>.o</code>	Linker

[a] When `-E` option is effective.

9.1.1.3 Return Value of the Compile Command

The possible return values of the compile command are as follows:

Table 9.3 Return Values of the Compile Command

Return Value	Meaning
0	Terminated normally.
1	A compiler error or linker error has occurred.

9.1.1.4 Notes on Compilation

This section provides notes on compilation.

9.1.1.4.1 Stack Size on Compilation

When compiling programs with remarkably many operations in loops, the segmentation exception may occur due to insufficient stack area in the compiler.

In this case, the stack area of the compilation environment needs to be extended to appropriate size.

The stack area of the process can be set by `ulimit` (bash built-in command), etc.

9.1.2 Compiler Options

This section describes the format of compiler options and their meanings.

The compiler options can be specified as follows:

- Option arguments of the compile command
- Environment variable (See Section "9.1.3 Environment Variable for Compile Command")
- Compilation profile file (See Section "9.1.4 Compilation Profile File")

See Section "2.4 Compilation Profile File" for the priority of the specified compiler options.



Note

- Almost all Clang/LLVM options are available. However, if you specify an option not described in this chapter, the result is not guaranteed. The result is not guaranteed when using the options not listed in this chapter.
- Trad Mode compilation options not listed in "Table 9.6 Options replaced in Clang Mode" cannot be specified in Clang Mode. If specified, it outputs an error message and aborts the compilation.

9.1.2.1 Syntax of the Compile Command

Compiler options can be specified as operands of the compile command. The format is given below.

General compiler options

```
[ -### ] [ -C ] [ -Dname[=tokens] ] [ -E ] [ -Idir ] [ -Ldir ] [ -M ] [ -MD ] [ -MF filename ] [ -MM ] [ -MMD ] [ -MP ] [ -MT target ] [ -Nopt ] [ -P ] [ -S ] [ -SSL2 ] [ -SSL2BLAMP ] [ -Uname ] [ -Wtool, arg1[, arg2]... ] [ -c ] [ -fopt ] [ {-g|-g0} ] [ -lname ] [ -o pathname ] [ -shared ] [ -v ] [ --coverage ] [ --verbose ]
```

Options for messages

```
[ -Rpass=. * ] [ -Rpass-analysis=. * ] [ -Rpass-missed=. * ] [ -w ] [ -Koptmsg=2 ]
```

Options for optimization

```
[ -O[n] ] [ -m{omit-leaf-frame-pointer|no-omit-leaf-frame-pointer} ] [ -msve-vector-bits={512|scalable} ] [ -fopt ] [ -Kopt ]
```

Options for language specifications

```
[ -std=name ] [ --linkcoarray ] [ --linkfortran ]
```


Options for CPU/architecture

```
[ -march=arch[+features]\... ] [ -Karch ] [ -mcpu=cpuname[+features]\... ] [ -Kcpuname ]
```

Options for code generation

```
[ -fopt ] [ -mmodel=name ] [ -mfj-tls-size={12|24|32|48} ] [ -Kopt ]
```

9.1.2.2 Description of Compiler Options

A description of the compiler options is given below.

9.1.2.2.1 General Options for Compiler

The following options are general options for the compiler.

-###

Shows commands passed by the compile command, but does not execute.

-C

This option executes the preprocessing phase to retain all comments other than those on preprocessing directive lines.

-D*name*[=*tokens*]

Assigns *name* to the specified *tokens* as in the `#define` preprocessor directive.

-E

Performs only preprocessing on the specified C source code and sends the results to the standard output.

-Nopt

When **-Nopt** option is specified, it is replaced with the corresponding **-fopt** option.

For more detail, see "[Table 9.6 Options replaced in Clang Mode](#)".

-I*dir*

Sets the search path for headers with names that do not start with '/' (i.e. headers specified with an absolute path name). The directory specified in *dir* is searched first and then the directory-installed standard headers are searched.

If multiple directories are specified in multiple **-I** options, the directories are searched in the specified order.

-L*dir*

Adds *dir* to the list of directories through which the linker searches for libraries.

This option is passed to the linker.

-M

Specifies to output the dependency of the sources in the format recognized by the make command.

When this option is valid, only the preprocessing is executed.

-MD

Specifies to output the dependency of the sources in the format recognized by the make command.

By default, the result is output to the file with a ".d" suffix.

When **-E** and **-o** options are specified, the result is output to the file specified by **-o** option.

When **-MF** option is specified simultaneously, the result is output to the file specified by **-MF** option.

-MF *filename*

Specifies the *filename* to output the result of the dependency of the source using **-M**, **-MM**, **-MD** or **-MMD** options.

-MM

Specifies to output the dependency of the sources in the format recognized by the make command. However, a standard header is not contained in the output.

When this option is valid, only the preprocessing is executed.

-MMD

Specifies to output the dependency of the sources in the format to recognized by the make command. However, a standard header is not contained in the output.

By default, the result is output to the file with a ".d" suffix.

When -E and -o options are specified, the result is output to the file specified by -o option.

When -MF option is specified simultaneously, the result is output to the file specified by -MF option.

-MP

Specifies to append the phony target of the dependency to the output of the result produced by -M, -MM, -MD or -MMD option.

-MT *target*

Specifies to change the *target* of the dependency in the output of the result produced by -M, -MM, -MD or -MMD option.

-P

Specifies not to include linemarkers in the output of -E option.

-S

Compiles the specified C source file and leaves the assembly-language output in a corresponding file with the suffix ".s".

When this option and -flto option are specified simultaneously, the generated file includes the internal information of the compiler. Therefore, the file cannot be passed to Trad Mode or the assembler (as command).

When this option and either of -g option or -ffj-line option are enabled, the generated file includes information extended for Clang Mode. Therefore, the file cannot be passed to Trad Mode or the assembler (as command).

-SSL2

Specifies to link C-SSL II, C-SSL II thread-parallel capabilities, and BLAS/LAPACK.

-SSL2BLAMP

Specifies to link C-SSL II, C-SSL II thread-parallel capabilities, and BLAS/LAPACK thread-parallel versions.

This option just replaces the sequential BLAS/LAPACK from -SSL2 with the corresponding thread-parallel versions.

-U*name*

Invalidates the definition of *name* as specified in #undef preprocessor directive.

-W*tool*,*arg1*[,*arg2*]...

Specified *arg1*[,*arg2*]... are passed as separate arguments to *tool*. An individual argument must be separated by a comma only.

tool can be one of the following characters:

Character specified as <i>tool</i>	Where <i>arg1</i> [, <i>arg2</i>]... are passed
p	Preprocessor
a	Assembler
l	Linker

Note that -flto option is invalidated when the -Wa and -flto options are specified simultaneously.



Example

To indicate to the linker that delayed link should not be used for symbol resolution of shared library addresses, specify as follows:

```
-Wl,-z,now
```

-c

Suppresses the linking phase.

The files generated when this option and -flto option are specified simultaneously can be linked only when -Nclang option is set.

-fopt

For *opt*, specify one of the following:

```
{ {debug-info-for-profiling|no-debug-info-for-profiling} | {fj-fjcex|fj-no-fjcex} | {fj-fjprof|fj-no-fjprof} | {fj-hook-time|fj-no-hook-time} | {fj-line|fj-no-line} | fj-lst[={p|t}] | fj-lst-out=file | fj-src | profile-dir=dir_name | sanitize={address|undefined} | {unwind-tables|no-unwind-tables} }
```

-f{debug-info-for-profiling|no-debug-info-for-profiling}

The compiler option -fdebug-info-for-profiling generates signatures for additional information available to the Profiler. The signature is the information used by overload resolution. This additional information is required to use the -Minlined option of the Instant Performance Profiler.

If the compiler option -fdebug-info-for-profiling is specified, the signature is generated as additional information. In this case, the Instant Performance Profiler displays the function cost per signature for inlined functions when the -Minlined option is effective.

If the compiler option -fno-debug-info-for-profiling is specified, the signature is not generated as additional information. In this case, the Instant Performance Profiler displays the function cost per function name for inlined functions when the -Minlined option is effective. Also, because no signatures are generated, the size of the object program is smaller than when the compiler option -fdebug-info-for-profiling is specified.

The compiler option -fdebug-info-for-profiling is set by default. The compiler options -fdebug-info-for-profiling and -fno-debug-info-for-profiling require that the compiler option -ffj-line is set.

See the "Profiler User's Guide" about the Profiler.

-f{fj-fjcex|fj-no-fjcex}

-ffj-fjcex specifies to use the Fujitsu Extended Functions in Clang Mode. -ffj-no-fjcex option invalidates -ffj-fjcex option. -ffj-no-fjcex is set by default.

-ffj-fjcex option must be set at both compilation and linking.

For details about the extended functions, see ["Appendix G Fujitsu Extended Functions"](#).

-f{fj-fjprof|fj-no-fjprof}

-ffj-fjprof option specifies to enable the Profiler function. The -ffj-no-fjprof option invalidates the -ffj-fjprof option. These options must be set at linking. The default is -ffj-fjprof.

See the "Profiler User's Guide" for information on the Profiler.

-f{fj-hook-time|fj-no-hook-time}

-ffj-hook-time specifies to use a hook function that is called at regular time interval. -ffj-no-hook-time option invalidates -fj-hook-time option. -ffj-no-hook-time is set by default.

If -ffj-hook-time option is set, the user-defined function is called at regular time interval.

The interval time to call the user-defined function is specified by environment variable FLIB_HOOK_TIME. If environment variable FLIB_HOOK_TIME is not specified, the user-defined function is called at the interval of a minute.

For details about the environment variable FLIB_HOOK_TIME, see Section ["9.1.6.1 Environment Variable for Execution"](#).

This option must be set at linking.

See Section ["8.3.4 Calling the User-Defined Function at Regular Time Interval"](#), for information about the hook function.

-f{fj-line|fj-no-line}

-ffj-line generates additional information required to use the execution time sampling function provided with the Profiler. -ffj-no-line option invalidates -ffj-line option. -ffj-line is set by default.

If the -flto option is specified with -ffj-line option, information about loops required by the Profiler's execution time sampling feature will not be generated. See the "Profiler User's Guide" about the Profiler.

-ffj-lst[={p|t}]

Specifies to output compilation information to file(s) whose name is with suffix ".lst". When more than one C source files are specified, the file names are "*each-filename.lst*".

When the -ffj-lst option is specified without arguments, -ffj-lst=p is set.

-ffj-lst=p

Specifies to write compilation information that consists of source list.

The source list includes annotation of optimization which shows the situation of inline expansion and loop unrolling, etc.

-ffj-lst=t

Specifies to write detail optimization information to source list in addition to the function of -ffj-lst=p.



Note

.....
If the "*each-filename.opt.yaml*" file already exists in the directory where the compilation was performed, the information of compilation is not output.
.....

-ffj-lst-out=file

Specifies the filename to output compilation information.

When this option is specified, -ffj-lst=p option is also effective.

-ffj-src

Specifies to output source list to the standard output.

The source list includes annotation of optimization which shows the situation of automatic parallelization, inline expansion, loop unrolling, etc.

Note that when -ffj-lst or -ffj-lst-out=*file* option is specified, the source list is output to the *file*.

-fprofile-dir=dir_name

This option specifies storage location directory of the .gcdc file which is necessary for the use of the code coverage. For *dir_name*, the storage location directory name is specified by the relative path or the absolute path.

The .gcdc file is generated when you run an executable program that link an object file with information for code coverage. At runtime, the directory specified by *dir_name* is created if it does not already exist.

This option is effective when the --coverage and, -S or -c options are enabled.

For information about the code coverage feature and the default storage location directory, see "[9.9 Code Coverage](#)".

-fsanitize={address|undefined}

-fsanitize option is used to debug C programs. The debugging function includes embedded information for debugging to a C object file, and checks are performed automatically during execution. Checks are performed if -fsanitize=undefined or -fsanitize=address option is set.

Multiple debugging features can be combined by specifying multiple -fsanitize options.

This option must be set when compiling and linking programs.

This option invalidates -ffj-largepage option.

-fsanitize=address

Specifies whether to check the heap for improper use of memory, buffer overrun, and memory leaks.

-fsanitize=undefined

Specifies whether to check the validity of the referenced subscript range against the declared array size when arrays are referenced.

-f{unwind-tables|no-unwind-tables}

-funwind-tables specifies to generate unwind tables for storing information necessary for debugging, such as stack trace information.

-fno-unwind-tables option invalidates -funwind-tables option.

-funwind-tables is set by default.

{-g|-g0}

-g option generates additional information used by the debugger.

-g0 option invalidates -g option.

-g0 is set by default.

-lname

Specifies to link a library named *libname.so* or *libname.a*.

The position of this option in the command line is important, since libraries are searched for in the order in which the other libraries and object files appear in the command line. This option must be specified after source files.

This option is passed to the linker.

-o *pathname*

Creates a file of the name specified in *pathname*.

-shared

Specifies to create a shared object. This option is passed to the linker.

-v

This option specifies to display the executing command which compile command called as compiler, assembler and linker.

--coverage

The --coverage option specifies to generate the information for the code coverage. --coverage option should be specified when programs are compiled and linked.

When the --coverage option is effective, the -flto option is invalid.

If the --coverage option is specified, the execution performance may decrease due to the instructions that measure the execution time are added in the object program.

For details about the code coverage, see section "[9.9 Code Coverage](#)".

--verbose

This option specifies to display the executing command which compile command called as compiler, assembler and linker.

9.1.2.2.2 Options for Messages

The options for messages are given below.

-Rpass=.*

-Rpass=.* specifies to output the messages of applied optimizations.



Example

Usage of -Rpass=.*

```
$ gccpx -Nclang test01.c -Ofast -Rpass=.* -S
test01.c:9:20: remark: sinking zext [-Rpass=licm]
    a_array[i] = b_array[i] * c_array[i];
                   ^
test01.c:8:3: remark: vectorized loop (vectorization width: 4, interleaved count: 2) [-Rpass=loop-vectorize]
```

```
for(i = 0; i < N; i++) {  
^
```

-Rpass-analysis=.*

-Rpass-analysis=.* specifies to output the reason why the compiler did not apply optimization.



Example

Usage of **-Rpass-analysis=.***

```
$ fccpx -Nclang test02.c -Ofast -Rpass-analysis=.* -S  
test02.c:9:18: remark: loop not vectorized: call instruction cannot be vectorized [-Rpass-  
analysis=loop-vectorize]  
    for(i = 0; i < foo(); i++) {  
        ^  
test02.c:9:3: remark: loop not vectorized: could not determine number of loop iterations [-Rpass-  
analysis=loop-vectorize]  
    for(i = 0; i < foo(); i++) {  
        ^
```

-Rpass-missed=.*

-Rpass-missed=.* specifies to output that the optimization was not applied. However, the reason why the compiler did not apply optimization is not output.



Example

Usage of **-Rpass-missed=.***

```
$ fccpx -Nclang test02.c -Ofast -Rpass-missed=.* -S  
test02.c:9:3: remark: loop not vectorized [-Rpass-missed=loop-vectorize]  
    for(i = 0; i < foo(); i++) {  
        ^
```

-w

Suppresses the output of warning messages.

-Koptmsg=2

When **-Koptmsg=2** option is specified, it is replaced with **-Rpass=.*** option.

9.1.2.2.3 Options for Optimization

The optimization options are described below.

-O[n]

In *n*, specify the level of optimization as 0, 1, 2, 3 or fast. If **-O** option is specified without *n*, **-O2** is set. As the optimization level increases, the execution time decreases and the compilation time increases. The higher levels of optimization functionally include the lower levels of optimization.

If **-O[n]** option is not specified, **-O0** is set.

- Optimization level 0

No optimization is performed.

- Optimization level 1

Basic optimization is performed. This may reduce the object size and the execution time compared with optimization level 0.

- Optimization level 2

In addition to the optimization of level 1, the following optimizations are performed:

- Create an object using SIMD extension instructions (-fvectorize)
- Loop unrolling (-funroll-loops)
- Loop fission (-ffj-loop-fission)
- Using prefetch instructions (-ffj-prefetch-sequential)
- Inline expansion (-finline-functions)
- Superword Level Parallelism(-fslp-vectorize)
- Repeated application of optimization functions
Repeated application of optimization functions means that the optimization functions performed in optimization level 1 are repeatedly performed until there is no room for further optimization.

- Optimization level 3

In addition to the optimization of level 2, the more highly optimization is performed.

The compilation time and the size of the object program may increase more than the optimization level 2.

- Optimization level fast

Specifies the optimizations for high speed execution on the target machine.

This option is equivalent to the following:

`-O3 -ffj-fast-matmul -ffast-math -ffp-contract=fast -ffj-fp-relaxed -ffj-ilfunc -fomit-frame-pointer -finline-functions`

When this option is set, side effects may occur in the execution results due to effect of -ffj-fast-matmul, -ffast-math, -ffp-contract=fast, -ffj-fp-relaxed, and -ffj-ilfunc options. See "[9.1.2.3.2 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

`-m{omit-leaf-frame-pointer|no-omit-leaf-frame-pointer}`

Indicates whether or not to perform the optimization that the frame pointer register is not kept for leaf functions (functions that do not call any other functions).

When -momit-leaf-frame-pointer option is specified, this optimization is performed. Trace back information is not kept.

When -mno-omit-leaf-frame-pointer option is specified, this optimization is not performed and the execution performance may decrease by saving and restoring instructions for keeping frame pointer register in leaf functions.

-momit-leaf-frame-pointer option is set by default.

The following is the relationship to the -f{omit-frame-pointer|no-omit-frame-pointer} options.

- -momit-leaf-frame-pointer option is induced by -fomit-frame-pointer option.
- -mno-omit-leaf-frame-pointer option is induced and enabled only when -fno-omit-frame-pointer option is specified explicitly. Specifying only -mno-omit-leaf-frame-pointer option has no effect.

`-msve-vector-bits={512|scalable}`

Specifies the size of the SVE vector register. Units are bits.

When the -msve-vector-bits=512 option is specified, optimizations are performed on the assumption that the vector register size is a fixed value specified as the option at compilation time. Therefore, optimizations are promoted and the improvement of the execution performance is expected.

However, the generated executable program works normally only on CPU architecture which has the same size of the SVE vector register as the size specified at compilation time. For details, see Section "[9.1.2.3.5 Notes on Specified SVE Vector Register Size](#)".

When the -msve-vector-bits=scalable option is specified, the SVE vector register is not considered to be a specific size, and the executable program decides the vector register size at execution time. The executable program does not depend on the SVE vector register size on the CPU architecture.

-msve-vector-bits=scalable is set by default.

This option is effective when sve is enabled at *features* of -march option.

Note that -fslp-vectorize option is invalidated when the -msve-vector-bits=512 options is specified. And, the SIMD built-in functions cannot be used when the -msve-vector-bits=512 option is specified.

If both the -msve-vector-bits=512 and -flt0 options are specified, the -msve-vector-bits=512 option is disabled and the -msve-vector-bits=scalable option is enabled.

-fopt

For *opt*, specify one of the following:

```
{ {fj-eval-concurrent|fj-no-eval-concurrent} | {fj-fast-matmul|fj-no-fast-matmul} | {fj-fp-precision|fj-no-fp-precision} | {fj-fp-relaxed|fj-no-fp-relaxed} | {fj-hpctag|fj-no-hpctag} | {fj-ilfunc[={loop|procedure}]|fj-no-ilfunc} | {fj-interleave-loop-insns[=N]|fj-no-interleave-loop-insns} | {fj-loop-fission|fj-no-loop-fission} | fj-loop-fission-threshold=N | fj-no-prefetch | {fj-ocl|fj-no-ocl} | {fj-optimlib-string|fj-no-optimlib-string} | {fj-preex|fj-no-preex} | fj-prefetch-cache-level={1|2|all} | {fj-prefetch-conditional|fj-no-prefetch-conditional} | fj-prefetch-iteration=N | fj-prefetch-iteration-L2=N | fj-prefetch-line=N | fj-prefetch-line-L2=N | {fj-prefetch-sequential[={auto|soft}]|fj-no-prefetch-sequential} | {fj-prefetch-stride|fj-no-prefetch-stride} | {fj-prefetch-strong|fj-no-prefetch-strong} | {fj-prefetch-strong-L2|fj-no-prefetch-strong-L2} | {fj-promote-licm-addressing|fj-no-promote-licm-addressing} | {fj-regalloc-using-latency|fj-no-regalloc-using-latency} | {fj-sched-insn-contiguous|fj-no-sched-insn-contiguous} | {fj-swp|fj-no-swp} | {fj-zfill[=N]|fj-no-zfill} | {builtin|no-builtin} | {fast-math|no-fast-math} | {finite-math-only|no-finite-math-only} | fp-contract={fast|on|off} | {inline-functions|no-inline-functions} | {lto|no-lto} | {omit-frame-pointer|no-omit-frame-pointer} | {openmp|no-openmp} | {openmp-simd|no-openmp-simd} | {reroll-loops|no-reroll-loops} | {signed-char|unsigned-char} | {slp-vectorize|no-slp-vectorize} | {strict-aliasing|no-strict-aliasing} | {unroll-loops|no-unroll-loops} | {vectorize|no-vectorize} }
```

-{fj-eval-concurrent|fj-no-eval-concurrent}

-ffj-eval-concurrent specifies to improve the parallelism of floating-point arithmetic instructions using the optimization of tree-height-reduction. -ffj-no-eval-concurrent gives priority to the optimization using FMA instructions over tree-height-reduction. -ffj-no-eval-concurrent is set by default.

-ffj-eval-concurrent and -ffj-no-eval-concurrent options require that -O1 option or higher is effective and the -ffast-math option is effective.

-{fj-fast-matmul|fj-no-fast-matmul}

The compiler option -ffj-fast-matmul performs the optimization that converts the loop of matrix multiplication into a high speed library call. The compiler option -ffj-no-fast-matmul invalidates the compiler option -ffj-fast-matmul. The compiler option -ffj-no-fast-matmul is set by default.

When the compiler option -ffj-fast-matmul is set, side effects (calculation errors) may occur in the execution results. See "[9.1.2.3.2 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

The compiler options -ffj-fast-matmul and -ffj-no-fast-matmul require that the compiler option -O2 or higher is set.

The compiler option -ffj-fast-matmul cannot be specified simultaneously with -shared option.

The compiler option -ffj-fast-matmul is needed if an object program compiled with it exists in the command line as an input file.

-{fj-fp-precision|fj-no-fp-precision}

-ffj-fp-precision specifies to induce the combination of optimization options that do not cause calculation errors in floating-point operations. -ffj-no-fp-precision option invalidates -ffj-fp-precision option. -ffj-no-fp-precision is set by default.

The -ffj-fp-precision option is equivalent to replacing the -ffj-fp-precision option with the following options:

-fno-fast-math -ffp-contract=off -ffj-no-fast-matmul -ffj-no-fp-relaxed -ffj-no-ilfunc

Therefore, if the -ffj-fp-precision option is enabled, some optimizations are limited and may decrease the performance.

The -ffj-no-fp-precision option does not affect the individual options induced by -ffj-fp-precision option, even if they are specified together.

`-f{fj-fp-relaxed|fj-no-fp-relaxed}`

`-ffj-fp-relaxed` specifies to convert floating point division operations and `sqrt` functions into the reciprocal approximation operation with the reciprocal approximation instructions and the Floating-Point Multiply-Add/Subtract instructions. This conversion is applied to the single-precision and double-precision real type calculation. `-ffj-no-fp-relaxed` option invalidates `-ffj-fp-relaxed` option. `-ffj-no-fp-relaxed` is set by default.

When `-ffj-fp-relaxed` is set, side effects may occur in the execution results. See "[9.1.2.3.2 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

`-ffj-fp-relaxed` and `-ffj-no-fp-relaxed` options require that `-O1` option or higher is set.

`-f{fj-hpctag|fj-no-hpctag}`

`-ffj-hpctag` specifies to use the HPC tag address override feature of the A64FX processor. If the `-ffj-hpctag` option is specified, the HPC tag address override feature is used. The default is `-ffj-hpctag`.

Using the HPC tag address override feature, the hardware prefetch assistance feature become effective.

This option is effective when the `-mcpu=a64fx` option is set.

`-ffj-hpctag` and `-ffj-no-hpctag` options should be specified when programs are compiled and linked.

`-f{fj-ilfunc={loop|procedure}}|fj-no-ilfunc}`

`-ffj-ilfunc` performs the inline expansion for the math functions. `-ffj-no-ilfunc` option invalidates `-ffj-ilfunc` option. `-fbuiltin` option is required for `-ffj-ilfunc` option to take effect. `-ffj-no-ilfunc` is set by default.

If the argument `={loop|procedure}` is omitted, `-ffj-ilfunc=procedure` is set.

When `-ffj-ilfunc` option is set, side effects may occur in the execution results. See "[9.1.2.3.2 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

`-ffj-ilfunc` and `-ffj-no-ilfunc` options require that `-O1` option or higher is set.

The following shows the functions for which inline expansion can be performed.

- Single-precision and double-precision real type function

<code>atan, atan2, cos, exp, log, log10, pow, sin, tan</code>

- Single-precision and double-precision complex type function

<code>abs, exp</code>

`-ffj-ilfunc=loop`

Specifies to perform the inline expansion for the math functions in a loop. Inline expansion is applied when the optimization may be promoted by checking a data type, an existence of function call and so on.

`-ffj-ilfunc=procedure`

Specifies to perform the inline expansion for all the math functions in a function.

`-f{fj-interleave-loop-insns[=N]|fj-no-interleave-loop-insns}`

`-ffj-interleave-loop-insns` specifies to apply loop interleaving to SIMD applied loops using SVE.

N is the interleaving count which should be from 2 to 10. If `=N` is omitted, the compiler determines a suitable value for *N*. The `-ffj-no-interleave-loop-insns` option invalidates the `-ffj-interleave-loop-insns` option. `-ffj-no-interleave-loop-insns` option is set by default.

This option is effective only when `-fvectorize` option is effective and `sve` is enabled at *features* of `-march` option.

Loop interleaving performs the same optimization as loop striping in Trad Mode.

`-f{fj-loop-fission|fj-no-loop-fission}`

`-ffj-loop-fission` option specifies to perform the automatic loop fission optimization for the software pipelining, SIMD Extensions, and the resolution of register shortage. `-ffj-no-loop-fission` option invalidates `-ffj-loop-fission` option.

This loop fission optimization is performed for the loop with the `loop loop_fission_target` specifier when `-ffj-loop-fission` option and `-ffj-ocl` option are enabled. `-ffj-no-loop-fission` option is always applied when `-O1` option is set. `-ffj-loop-fission` option is set by default when `-O2` option or higher is set.

`-ffj-loop-fission` option works only when used in combination with the `loop loop_fission_target` specifier.

The algorithm of the loop fission optimization by `-ffj-loop-fission` option is correspond to the clustering algorithm of Trad Mode.

The loop fission optimization by `-ffj-loop-fission` option tends to use large stack area of the memory.

`-ffj-loop-fission-threshold=N`

Specifies the threshold *N* to decide the granularity of loops after the automatic loop fission.

N is an integer value from 1 to 100. If the value of *N* is reduced, the fissioned loops tend to become small and the number of the fissioned loops tends to increase. `-ffj-loop-fission-threshold=50` is set by default.

This option requires that `-ffj-loop-fission` option, `-ffj-ocl` option, and `-O2` option or higher are set and the `loop loop_fission_target` specifier is specified in the optimization control line.

`-ffj-no-prefetch`

Specifies to generate an object without a prefetch instruction.

`-f{fj-ocl|fj-no-ocl}`

`-ffj-ocl` specifies that the optimization control line that is supported as Fujitsu-specific pragma supported by Clang Mode is effective.

`-ffj-no-ocl` option invalidates `-ffj-ocl` option. `-ffj-no-ocl` is set by default.

`-ffj-ocl` and `-ffj-no-ocl` options require that `-O1` option or higher is set.

`-f{fj-stdlib-string|fj-no-stdlib-string}`

Indicates whether or not to link the optimized version of the library in the string manipulation function. Statically links the optimized library if the `-ffj-stdlib-string` option is specified. If omitted, the `-ffj-no-stdlib-string` option applies.

The `-ffj-stdlib-string` option is enabled when specified with the `-mcpu=a64fx[+sve]` option.

The `-ffj-stdlib-string` option cannot be specified simultaneously with the `-flto` option when the `-shared` option is specified.

This option must be specified when compiling and linking the program.

The following string manipulation functions are supported:

<code>bcopy</code> , <code>bzero</code> , <code>memchr</code> , <code>memcmp</code> , <code>memcpy</code> , <code>memccpy</code> , <code>memcpy</code> , <code>memmove</code> , <code>memset</code> , <code>strcat</code> , <code>strcmp</code> , <code>strcpy</code> , <code>strlen</code> , <code>strncmp</code> , <code>strncpy</code> , <code>strncat</code>
--

`-f{fj-preex|fj-no-preex}`

`-ffj-preex` specifies to evaluate invariant expressions in advance. `-ffj-no-preex` option invalidates `-ffj-preex` option. `-ffj-no-preex` is set by default.

When `-ffj-preex` option is set, side effects (calculation error or runtime exceptions) may occur in the execution results because instructions that may not to be executed based on the logic of the program are likely to be executed.

`-ffj-preex` and `-ffj-no-preex` options require that `-O1` option or higher is set.

`-ffj-prefetch-cache-level={1|2|all}`

Specifies the data cache-level of prefetch instructions. By default, `-ffj-prefetch-cache-level=all` is set.

This option requires that at least one of `-ffj-prefetch-sequential` or `-ffj-prefetch-stride` options is set.

`-ffj-prefetch-cache-level=1`

Data is prefetched in the first level cache. Normal prefetch instructions are used.

`-ffj-prefetch-cache-level=2`

Data is prefetched in the second level cache.

`-ffj-prefetch-cache-level=all`

Both `-ffj-prefetch-cache-level=1` and `-ffj-prefetch-cache-level=2` functions are effective. By using two levels of prefetch instructions, the prefetch function becomes more sophisticated.

-f{fj-prefetch-conditional|fj-no-prefetch-conditional}

-ffj-prefetch-conditional specifies to generate prefetch instructions for array data in the block included in an "if" statement and a "switch" statement. -ffj-no-prefetch-conditional option invalidates -ffj-prefetch-conditional option. -ffj-no-prefetch-conditional is set by default.

-ffj-prefetch-conditional and -ffj-no-prefetch-conditional options require that at least one of -ffj-prefetch-sequential or -ffj-prefetch-stride options is set.

-ffj-prefetch-iteration=*N*

Specifies to prefetch data which is referred to or defined in a loop after *N* iteration(s). *N* should be from 1 to 10000.

This option specifies to prefetch data to only the first level cache.

This option requires that at least one of -ffj-prefetch-sequential or -ffj-prefetch-stride options is set, and -ffj-prefetch-cache-level=1 or -ffj-prefetch-cache-level=all is set.

This option cannot be specified simultaneously with -ffj-prefetch-line option.

-ffj-prefetch-iteration-L2=*N*

Specifies to prefetch data which is referred to or defined in a loop after *N* iteration(s). *N* should be from 1 to 10000.

This option specifies to prefetch data to only the second level cache.

This option requires that at least one of -ffj-prefetch-sequential or -ffj-prefetch-stride options is set, and -ffj-prefetch-cache-level=2 or -ffj-prefetch-cache-level=all is set.

This option cannot be specified simultaneously with -ffj-prefetch-line-L2 option.

-ffj-prefetch-line=*N*

Specifies to prefetch data which is referred to or defined in a line after *N* line(s). *N* should be from 1 to 100. This option specifies to prefetch data to only the first level cache.

This option requires that -ffj-prefetch-sequential option is set, and -ffj-prefetch-cache-level=1 or -ffj-prefetch-cache-level=all is set.

This option cannot be specified simultaneously with -ffj-prefetch-iteration option.

-ffj-prefetch-line-L2=*N*

Specifies to prefetch data which is referred to or defined in a line after *N* line(s). *N* should be from 1 to 100. This option specifies to prefetch data to only the second level cache.

This option requires that -ffj-prefetch-sequential option is set, and -ffj-prefetch-cache-level=2 or -ffj-prefetch-cache-level=all is set.

This option cannot be specified simultaneously with -ffj-prefetch-iteration-L2 option.

-f{fj-prefetch-sequential[={auto|soft}]|fj-no-prefetch-sequential}

-ffj-prefetch-sequential specifies to generate prefetch instructions for prefetch array data accessed sequentially within a loop. -ffj-no-prefetch-sequential option invalidates -ffj-prefetch-sequential option. When -O1 option is set, -ffj-prefetch-nosequential is set by default. When -O2 or higher option is set, -ffj-prefetch-sequential is set by default.

If the argument = {auto|soft} is omitted, -ffj-prefetch-sequential=auto is assumed.

-ffj-prefetch-sequential and -ffj-no-prefetch-sequential options require that -O1 option or higher is set.

-ffj-prefetch-sequential=auto

The compiler automatically selects whether to use the hardware prefetch function or to output the prefetch instruction for array data accessed sequentially within a loop.

-ffj-prefetch-sequential=soft

The compiler specifies to output the prefetch instruction for array data accessed sequentially within a loop without using the hardware prefetch function.

-f{fj-prefetch-stride|fj-no-prefetch-stride}

-ffj-prefetch-stride specifies to generate prefetch instructions for prefetch array data with a stride larger than the cache line size in its loop. In addition, the loops include prefetch instructions to prefetch addresses not determined clearly during compilation. -ffj-no-prefetch-stride option invalidates -ffj-prefetch-stride option. -ffj-no-prefetch-stride is set by default.

-ffj-prefetch-stride and -ffj-no-prefetch-stride options require that -O1 option or higher is set.

-f{fj-prefetch-strong|fj-no-prefetch-strong}

-ffj-prefetch-strong specifies to generate strong prefetch instructions. -ffj-no-prefetch-strong option invalidates -ffj-prefetch-strong option. -ffj-prefetch-strong is set by default.

-ffj-prefetch-strong option specifies to prefetch data to only the first level cache.

-ffj-prefetch-strong option requires that at least one of -ffj-prefetch-sequential or -ffj-prefetch-stride options is set, and -ffj-prefetch-cache-level=1 or -ffj-prefetch-cache-level=all option is set.

-ffj-no-prefetch-strong is effective if -ffj-hpctag option is set.

-f{fj-prefetch-strong-L2|fj-no-prefetch-strong-L2}

This option specifies to generate strong prefetch instructions. -ffj-no-prefetch-strong-L2 option invalidates -ffj-prefetch-strong-L2 option. -ffj-prefetch-strong-L2 is set by default.

-ffj-prefetch-strong-L2 option specifies to prefetch data to only the second level cache.

-ffj-prefetch-strong-L2 option requires that at least one of -ffj-prefetch-sequential or -ffj-prefetch-stride options is set, and -ffj-prefetch-cache-level=2 or -ffj-prefetch-cache-level=all option is set.

-ffj-no-prefetch-strong-L2 is effective if -ffj-hpctag option is set.

-f{fj-promote-licm-addressing|fj-no-promote-licm-addressing}

-ffj-promote-licm-addressing specifies to perform before SIMDization the optimization which extract loop invariant terms from address calculations to promote movement of invariant expressions.

-ffj-no-promote-licm-addressing option invalidates -ffj-promote-licm-addressing option.

-ffj-promote-licm-addressing option is set by default when -O1 option or higher is set.

-ffj-promote-licm-addressing option requires -O1 option or higher is set.

-f{fj-regalloc-using-latency|fj-no-regalloc-using-latency}

-ffj-regalloc-using-latency specifies that at the register allocation for instructions in a loop, the registers for spill are determined in consideration of the instruction latency.

-ffj-no-regalloc-using-latency option invalidates -ffj-regalloc-using-latency option.

-ffj-regalloc-using-latency option is set by default when -O1 option or higher is set.

-ffj-regalloc-using-latency option requires -O1 option or higher is set.

-f{fj-sched-insn-contiguous|fj-no-sched-insn-contiguous}

-ffj-sched-insn-contiguous specifies that the instruction scheduler before and after the register allocation order each of floating-point arithmetic instructions, load instructions, and store instructions sequentially if possible.

-ffj-no-sched-insn-contiguous option invalidates -ffj-sched-insn-contiguous option.

-ffj-sched-insn-contiguous option is set by default when -O1 option or higher is set.

-ffj-sched-insn-contiguous option requires -O1 option or higher is set.

-f{fj-swp|fj-no-swp}

The -ffj-swp option performs software pipelining. The -ffj-no-swp option invalidates the -ffj-swp option. The -ffj-no-swp option is set by default. These options are effective only if the -O1 option or higher and -mcpu=a64fx option are specified.

No optimization is performed for the following loops.

- Pipelining is ineffective.
- Vectorized by the -msve-vector-bits=scalable option.

The -ffj-swp option induces the -msve-vector-bits=512 option. The induction is disabled when the -flt option is enabled. Also, the induction is disabled when the -msve-vector-bits=scalable option is specified after the -ffj-swp option. The induction of the -msve-vector-bits=512 option does not depend on the existence of the `loop swp` specifier.

Pipelining may not be performed if the -g option is set.

When specifying the `-Rpass=.*`, `-Rpass-analysis=.*`, and `-Rpass-missed=.*` options, the optimization messages for software pipelining are for one or more loops generated by the compiler from a single loop explicitly written in the source program. Therefore, if multiple loops are generated by the compiler, different multiple messages may be emitted for a single loop on the source file.

Specifying the `-Rpass=.*` option, without the `-Rpass-analysis=.*` and `-Rpass-missed=.*` options, can avoid confusion in the messages resulting from the loops generated by the compiler, and in this case the message "remark: loop pipelined" means that software pipelining is applied to one of these loops.

`-f{fj-zfill[=N]|fj-no-zfill}`

The `-ffj-zfill` option specifies to perform zfill optimization. The zfill optimization speeds up write operations for array data that is only written in a loop, by using an instruction that allocates space on the cache for writing (DC ZVA) without loading data from the memory. The zfill optimization works on the data *N* blocks ahead of the address pointed to by the target store instruction where one block is 256 byte-long and *N* is an integer value between 1 and 100. If a value is not specified for *N*, the compiler will automatically determine a value. The `-ffj-no-zfill` option invalidates the `-ffj-zfill` option. The `-ffj-no-zfill` option is always applied when the `-O1` option is effective. The `-ffj-no-zfill` option is set by default when the `-O2` option or higher is set.

The `-ffj-zfill` option requires that the `-mcpu=a64fx` option, `-msve-vector-bits=512` option, and `-O2` option or higher are enabled. The `-ffj-zfill` option induces the `-msve-vector-bits=512` option. The induction is disabled when the `-flto` option is enabled. Also, the induction is disabled when the `-msve-vector-bits=scalable` option is specified after the `-ffj-zfill` option. The induction of the `-msve-vector-bits=512` option does not depend on the existence of the `loop zfill` specifier.

Note that if an object program compiled with the `-ffj-zfill` option is executed on a CPU other than the one on which a single cache write operation of a DC ZVA instruction is 256 bytes, the execution may be terminated abnormally or an incorrect result may occur. The amount of the cache write operation for a DC ZVA instruction on an A64FX processor is 256 bytes.

Performance may also be reduced under the following conditions:

- The program is not affected by memory bandwidth bottleneck.
- When the loop iteration count is too few.
- When the block size is explicitly specified with the `-ffj-zfill=N` option and the memory size where the data is written in the loop is smaller than *N* blocks.

This optimization is not expected to be effective for all the loops in the program, so it is recommended to use the `loop zfill` specifier for each loop rather than the `-ffj-zfill[=N]` option for the entire program.

For details about zfill, see Section "3.3.5 zfill".

`-f{builtin|no-builtin}`

`-fbuiltin` promotes optimization by recognizing the operation of standard library functions. The `-fno-builtin` option invalidates `-fbuiltin` option. `-fbuiltin` is set by default.

If the user-defined function with the same name as a standard library function is used, unexpected results may occur.

The recognized standard library functions are as follows:

```
abort, abs, acos, acosf, acoshf, asin, asinf, asinhf, atan, atan2, atan2f, atanf, atanhf,
calloc, cbrt, cbrtf, ceil, ceilf, clearerr, copysign, copysignf, cos, cosf, cosh, coshf, csqrt,
csqrtf, erf, erfc, erfcf, erff, exit, exp, exp2, exp2f, expf, expmlf, fabs, fabsf, fclose, feof,
ferror, fflush, fgetc, fgetpos, fgets, floor, floorf, fma, fmaf, fmax, fmaxf, fmin, fminf, fmod,
fmodf, fputc, fputs, free, frexp, fseek, fwrite, getenv, hypotf, ilogb, ilogbf, isalnum,
isalpha, iscntrl, isdigit, isgraph, islower, ispunct, isspace, isupper, isxdigit, ldexp,
lgammaf, log, log10, log10f, loglpf, log2, log2f, logbf, logf, malloc, memchr, memcmp, memcpy,
memmove, memset, modf, nextafterf, perror, pow, powf, printf, putchar, rand, realloc,
remainderf, remove, rename, rint, rintf, scalbnf, scanf, setvbuf, sin, sinf, sinh, sinh,
sprintf, sqrt, sqrtf, srand, sscanf, strcat, strchr, strcmp, strcpy, strcspn, strerror, strlen,
strncat, strncmp, strncpy, strpbrk, strchr, strspn, strstr, strtod, strtok, strtol, strtoul,
system, tan, tanf, tanh, tanhf, tolower, toupper, vprintf, vsprintf
```

`-f{fast-math|no-fast-math}`

`-ffast-math` performs the optimization that changes the method of operator evaluation. `-fno-fast-math` option invalidates `-ffast-math` option. `-fno-fast-math` is set by default.

When the `-ffast-math` option is specified, calculation errors or the runtime exceptions may occur in the execution results and produce unexpected results due to the effects of optimization or flush-to-zero mode which replaces the denormalized numbers with zero with the same sign.

See "[9.1.2.3.2 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

The `-ffast-math` and `-fno-fast-math` options require that `-O1` option or higher is set.

The `-ffast-math` and `-fno-fast-math` options must be set at compiling and linking.

`-f{finite-math-only|no-finite-math-only}`

`-ffinite-math-only` option promote the optimization of floating point arithmetic based on the assumption that the argument or operation result is only a finite numerical value.

`-ffast-math` option induces the `-ffinite-math-only` option.

`-ffp-contract={fast|on|off}`

`-ffp-contract=fast` specifies to perform optimization using Floating-Point Multiply-Add/Subtract instructions. `-ffp-contract=on` specifies to perform optimization using Floating-Point Multiply-Add/Subtract instructions as if `#pragma STDC FP_CONTRACT ON` (language standard) was specified in the source code. `-ffp-contract=off` option invalidates `-ffp-contract={fast|on}`. `-ffp-contract=off` is set by default.

When `-ffp-contract={fast|on}` option is effective, side effects (calculation errors in rounding error extent) may occur in the execution results and produce unexpected results. See "[9.1.2.3.2 Side Effect of Optimizations for Floating-Point Operation](#)" for the side effect of optimization.

`-ffp-contract={fast|off}` options require that `-O1` option or higher is set.

`-ffp-contract=on` option requires that `-O0` option or higher is set.

`-f{inline-functions|no-inline-functions}`

`-finline-functions` specifies to perform Inline expansion of function calls defined in source code.

`-finline-functions` is set by default when `-O2` option or higher is set.

This option is effective only if `-O2` option or higher is set.

Therefore, the inline function does not become the target of the inline expansion when the compiler judges that inline expansion should not be performed.

`-f{lto|no-lto}`

`-flto` specifies to perform the link time optimization. `-fno-lto` option invalidates `-flto` option. `-flto` option should be specified when programs are compiled and linked. `-fno-lto` option is assumed by default.

If the `-ffj-line` option is specified with `-flto` option, information about loops required by the Profiler's execution time sampling feature will not be generated. See the "Profiler User's Guide" about the Profiler.

If `-flto` option is set, the only options for optimization that can be specified at the same time are the following.

- `-O[n]` option
- The options described to be set at linking in Section "[9.1.2.2.3 Options for Optimization](#)"

The `-flto` option cannot be specified simultaneously with the `-ffj-stdlib-string` option when the `-shared` option is specified.

If both the `-flto` and `-msve-vector-bits=512` options are specified, the `-msve-vector-bits=512` option is disabled and the `-msve-vector-bits=scalable` option is enabled.

`-f{omit-frame-pointer|no-omit-frame-pointer}`

Indicates whether or not to perform the optimization that the frame pointer register is not kept for a function call.

`-fomit-frame-pointer` option does the optimization. Trace back information is not kept.

If `-fomit-frame-pointer` option is specified, `-momit-leaf-frame-pointer` option is induced.

If `-fno-omit-frame-pointer` option is specified, `-mno-omit-leaf-frame-pointer` option is induced.

By default, `-fno-omit-frame-pointer` option is set, but in this case `-mno-omit-leaf-frame-pointer` option is not induced.

-f{openmp|no-openmp}

-fopenmp specifies to accept the Specification of OpenMP Application Program Interface. -fno-openmp option invalidates -fopenmp option. -fno-openmp is set by default.

-fopenmp option is needed if an object program compiled with it exists in the command line as an input file.

-f{openmp-simd|no-openmp-simd}

The -fopenmp-simd specifies that only the OpenMP simd construct, the declare reduction construct, and the ordered construct with the simd clause are enabled. -fno-openmp-simd is set by default.

When the -fopenmp-simd option is specified, only the SIMD Extensions based on the OpenMP specifications are applied, and the parallelization is not applied.

When the -fno-openmp-simd option is specified, the -fopenmp_simd option is disabled.

When the -O1 option or less is enabled, SIMD Extensions are not used even if simd construct is enabled.

-f{reroll-loops|no-reroll-loops}

-freroll-loops specifies to apply loop rerolling. -fno-reroll-loops is set by default.

-fsigned-char

Specifies to treat char type variable as signed char type.

-funsigned-char

Specifies to treat char type variable as unsigned char type.

-f{slp-vectorize|no-slp-vectorize}

-fslp-vectorize specifies to apply SLP(Superword Level Parallelism) using SIMD instructions.

When the -msve-vector-bits=512 option is effective, the -fslp-vectorize option is invalidated.

When -O1 option is set, -fno-slp-vectorize is set by default. When -O2 option or higher is set, -fslp-vectorize is set by default.

-f{strict-aliasing|no-strict-aliasing}

-fstrict-aliasing option specifies to perform optimizations with considering overlaps of memory regions according to the strict aliasing rules in the language standards. -fno-strict-aliasing option invalidates -fstrict-aliasing option. When -O0 option is set, -fno-strict-aliasing is always set. When -O1 or higher option is set, -fstrict-aliasing is set by default.

-fstrict-aliasing and -fno-strict-aliasing options require that -O1 option or higher is set.

-f{unroll-loops|no-unroll-loops}

-funroll-loops specifies to apply loop unrolling. The suitable multiplicity of loop unrolling is determined by the compiler.

-fno-unroll-loops option invalidates -funroll-loops option. -fno-unroll-loops is set by default when -O1 option is set. -funroll-loops is set by default when -O2 option or higher is set.

-f{vectorize|no-vectorize}

-fvectorize specifies to perform optimization using the SIMD Extensions for loop. -fno-vectorize option invalidates -fvectorize option.

When -O1 option is set, -fno-vectorize is set by default. When -O2 option or higher is set, -fvectorize is set by default.

-Kopt

-Kopt listed in "[Table 9.6 Options replaced in Clang Mode](#)" is replaced with the corresponding option of Clang Mode.

9.1.2.2.4 Options for Language Specifications

This section explains the options relating to language specifications.

-std=*name*

This option specifies the level of language specifications to be interpreted by the compiler (including the preprocessor).

For *name*, specify one of the following:

{ c89 c99 c11 gnu89 gnu99 gnu11 }

-std=gnu11 is set by default.

-std=c89

The C language specification used at compilation time is based on the C89 specifications.

-std=c99

The C language specification used at compilation time is based on the C99 specifications.

-std=c11

The C language specification used at compilation time is based on the C11 specifications.

-std=gnu89

In addition to the C89 specifications, GNU C Extensions is used at compilation time.

-std=gnu99

In addition to the C99 specifications, GNU C Extensions is used at compilation time.

-std=gnu11

In addition to the C11 specifications, GNU C Extensions is used at compilation time.

--linkcoarray

Directs to search for a library required for linkage with Fortran that does use the COARRAY specification.

If this option is not specified, search processing is not performed, enabling a reduction in compilation time.

--linkfortran

Directs to search for a library required for linkage with Fortran that does not use the COARRAY specification.

If this option is not specified, search processing is not performed, enabling a reduction in compilation time.

9.1.2.2.5 Options for CPU/Architecture

This section describes about the optimization options of the processor and the architecture.

-march=*arch*[+*features*]...

Specifies to generate the object file that is suitable for the given name of architecture and extension.

For *arch*, specify one of the following:

{ armv8-a armv8.1-a armv8.2-a armv8.3-a }

-march=armv8-a

Specifies to generate object files using instructions in Armv8-A.

-march=armv8.1-a

Specifies to generate object files using instructions in Armv8-A and Armv8.1-A.

-march=armv8.2-a

Specifies to generate object files using instructions in Armv8-A, Armv8.1-A, and Armv8.2-A.

-march=armv8.3-a

Specifies to generate object files using instructions in Armv8-A, Armv8.1-A, Armv8.2-A, and Armv8.3-A.

For *features*, specify one or more of the following by separating them with + :

{ sve nosve fp16 }

sve

Specifies to output object files using SVE, which is an Armv8-A extension.

nosve

Specifies to output object files without the use of SVE, which is an Armv8-A extension.

Specify the argument +nosve to generate object files for a processor that does not support SVE.

fp16

Specifies to generate the object file using the extension of the half-precision floating-point. About the language specifications, see Section "9.5.2 Half-Precision (16 bit) Floating-Point Type".

If this option is omitted, the values corresponding to the -mcpu option are set in *arch* and *features* of -march option.

Table 9.4 Default Values of -march Option

Effective Value of -mcpu Option	Default Value of -march Option
-mcpu=a64fx[+sve]	-march=armv8.2-a+sve+fp16
-mcpu=a64fx+nosve	-march=armv8.2-a+nosve+fp16
-mcpu=generic[+nosve]	-march=armv8-a+nosve
-mcpu=thunderx[+nosve]	
-mcpu=generic+sve	-march=armv8-a+sve
-mcpu=thunderx+sve	
-mcpu=thunderx2t99[+nosve]	-march=armv8.1-a+nosve
-mcpu=thunderx2t99+sve	-march=armv8.1-a+sve



Note

If the -mcpu and -march options are specified simultaneously, the architecture and *features* specified by the -march option take precedence.

-Karch

-Karch option listed in "Table 9.6 Options replaced in Clang Mode" is replaced with the corresponding -march option of Clang Mode.

-mcpu=*cpu*name[+*features*]...

Specifies to generate the object file that is suitable for the given name of processor.

For *cpu*name, specify one of the following:

```
{ a64fx | generic | thunderx | thunderx2t99 }
```

-mcpu=a64fx

-mcpu=a64fx specifies to output the object file for the A64FX processor.

-mcpu=generic

-mcpu=generic specifies to output the object file for the Arm processor.

-mcpu=thunderx

-mcpu=thunderx specifies to output the object file for the ThunderX processor.

-mcpu=thunderx2t99

-mcpu=thunderx2t99 specifies to output the object file for the ThunderX2 processor.

For *features*, specify one or more of the following by separating them with +:

```
{ sve | nosve | fp16 }
```

sve

Specifies to output object files using SVE, which is an Armv8-A extension.

nosve

Specifies to output object files without the use of SVE, which is an Armv8-A extension.

Specify the argument `+nosve` to generate object files for a processor that does not support SVE.

fp16

Specifies to generate the object file using the extension of the half-precision floating-point. About the language specifications, see Section "[9.5.2 Half-Precision \(16 bit\) Floating-Point Type](#)".

By default, `-mcpu=a64fx` is set.



Note

If the `-mcpu` and `-march` options are specified simultaneously, the architecture and *features* specified by the `-march` option take precedence.

-Kcpu`name`

`-Kcpuname` option listed in "[Table 9.6 Options replaced in Clang Mode](#)" is replaced with the corresponding `-mcpu` option of Clang Mode.

9.1.2.2.6 Options for Code Generation

This section describes about the options of code generation.

-fopt

For *opt*, specify one of the following:

{ {PIC pic} {ffj-largepage ffj-no-largepage} }
--

-f{PIC|pic}

Specifies to generate position-independent code (PIC).

-f{ffj-largepage|ffj-no-largepage}

`-ffj-largepage` creates an executable program which uses the large page function. `-ffj-no-largepage` option invalidates `-ffj-largepage` option. `-ffj-largepage` is set by default.

When `-fsanitize` option is specified, `-ffj-largepage` option is invalidated.

`-ffj-largepage` and `-ffj-no-largepage` option must be set at linking.

-mcmode=*name*

This option specifies the possible largest size of the text area and the static data area in an executable program or a shared object.

For *name*, specify one of the following:

{small large}

`-mcmode=small` is set by default.

-mcmode=small

The total size of the text area and the static data area is limited to 4GB at linking. This option creates an efficient object program.

-mcmode=large

Only the size of the text area is limited to 4GB at linking. This option is used when the static data area is large and an error occurs at linking.

-mfj-tls-size={12|24|32|48}

Specifies the size of an offset necessary for the access to Thread-Local Storage. Units are bits.

As the size of Thread-Local Storage, `-mfj-tls-size=12` (4K bytes), `-mfj-tls-size=24` (16M bytes), `-mfj-tls-size=32` (4G bytes) or `-mfj-tls-size=48` (256T bytes) can be specified.

When the size of the Thread-Local Storage exceeds the range of the offset, an error occurs at link time.

This option cannot be specified simultaneously with the `-flto` option.

`-Kopt`

`-Kopt` option listed in "[Table 9.6 Options replaced in Clang Mode](#)" is replaced with the corresponding option of Clang Mode.

9.1.2.3 Notes on Compile Options

This section describes notes on the compile options.

9.1.2.3.1 Correspondences of Compile Options in Clang Mode and Trad Mode

The compile option system of Clang Mode is different from that of Trad Mode. You should pay attention to change the compiler mode when it used for compiling.

The compile options whose option name and function overview are the same in Clang Mode and Trad Mode are described in "[Table 9.5 Common compile options in Clang/Trad Mode](#)". The compile options whose function overview is same but option name is different is described in "[Table 9.6 Options replaced in Clang Mode](#)".

- A compile option of Trad Mode which is not described in "[Table 9.6 Options replaced in Clang Mode](#)" can not be specified in Clang Mode. If it is specified, an error message is output and the compilation is canceled.
 - The `-K{SVE|NOSVE}` options of Trad Mode cannot be specified in Clang Mode. If specified, it outputs an error message and aborts the compilation. To indicate whether to use SVE, add `+{sve|nosve}` to the `-mcpu` or `-march` option.
- If a compile option of Trad Mode which is described in "[Table 9.6 Options replaced in Clang Mode](#)", a corresponding compile option of Clang Mode is assumed to be specified.
- Even if a compile option whose function overview is same is specified, an optimization result in Clang Mode may be different from that in Trad Mode.
 - If the `-Kfast` option is specified, the induced options and the architecture set by default are different from the Trad Mode.
 - For included options, see the `-Ofast` option in "[9.1.2.2.3 Options for Optimization](#)" for Clang Mode and the `-Kfast` option in "[2.2.2.5 -K Option](#)" for Trad Mode.
 - For architectures that are set by default, see the `-march` option in "[9.1.2.2.5 Options for CPU/Architecture](#)" for Clang Mode and the `-Karchi` option in "[2.2.2.5 -K Option](#)" for Trad Mode.
- The following options have different default behavior in Trad Mode and Clang Mode.
 - `-K{strict_aliasing|nostrict_aliasing}`
 Trad Mode : `-Knostrict_aliasing`
 Clang Mode : (if the `-O0` option is set) `-fnostrict-aliasing`, (if the `-O1` option or higher is set) `-fstrict-aliasing`
 - `-K{omitfp|noomitfp}`
 Trad Mode : `-Knoomitfp`
 Clang Mode : `-fno-omit-frame-pointer`, and (only if `-m {omit-leaf-frame-pointer | no-omit-leaf-frame-pointer}` is not specified) `-momit-leaf-frame-pointer`

Table 9.5 Common compile options in Clang/Trad Mode

Option Name
<code>-C</code>
<code>-Dname [=tokens]</code>
<code>-E</code>
<code>-Idir</code>

Option Name
<code>-Ldir</code>
<code>-M</code>
<code>-MD</code>
<code>-MFfilename</code>
<code>-MM</code>
<code>-MMD</code>
<code>-MP</code>
<code>-MTtarget</code>
<code>-P</code>
<code>-S</code>
<code>-SSL2</code>
<code>-SSL2BLAMP</code>
<code>-Uname</code>
<code>-Wtool, arg1[, arg2]...</code>
<code>-c</code>
<code>-g</code>
<code>-g0</code>
<code>-lname</code>
<code>-o pathname</code>
<code>-shared</code>
<code>-v</code>
<code>-w</code>

Table 9.6 Options replaced in Clang Mode

Option name in Trad Mode	Option name in Clang Mode
<code>-KA64FX</code>	<code>-mcpu=a64fx</code>
<code>-KARMV8_1_A</code>	<code>-march=armv8.1-a</code>
<code>-KARMV8_2_A</code>	<code>-march=armv8.2-a</code>
<code>-KARMV8_3_A</code>	<code>-march=armv8.3-a</code>
<code>-KARMV8_A</code>	<code>-march=armv8-a</code>
<code>-KGENERIC_CPU</code>	<code>-mcpu=generic</code>
<code>-K{PIC pic}</code>	<code>-f{PIC pic}</code>

Option name in Trad Mode	Option name in Clang Mode
-Kcmodel={small large}	-mcmodel={small large}
-K{eval noeval}	-f{fast-math no-fast-math}
-K{eval_concurrent eval_noconcurrent}	-ffj-{eval-concurrent no-eval-concurrent}
-Kfast	-Ofast
-K{fast_matmul nofast_matmul}	-ffj-{fast-matmul no-fast-matmul}
-K{fp_contract nofp_contract}	-ffp-contract={fast off}
-K{fp_relaxed nofp_relaxed}	-ffj-{fp-relaxed no-fp-relaxed}
-K{ilfunc[={loop procedure}] noilfunc}	-ffj-{ilfunc[={loop procedure}] no-ilfunc}
-K{largepage nolargepage}	-ffj-{largepage no-largepage}
-K{lib nolib}	-f{builtin no-builtin}
-K{loop_fission loop_nofission}	-ffj-{loop-fission no-loop-fission}
-Kloop_fission_threshold=N	-ffj-loop-fission-threshold=N
-Knoprefetch	-ffj-no-prefetch
-K{ocl noocl}	-ffj-{ocl no-ocl}
-K{omitfp noomitfp}	-f{omit-frame-pointer no-omit-frame-pointer}
-K{openmp noopenmp}	-f{openmp no-openmp}
-K{openmp_simd noopenmp_simd}	-f{openmp-simd no-openmp-simd}
-Koptmsg=2	-Rpass=.*
-K{preex nopreex}	-ffj-{preex no-preex}
-Kprefetch_cache_level={1 2 all}	-ffj-prefetch-cache-level={1 2 all}
-K{prefetch_conditional prefetch_noconditional}	-ffj-{prefetch-conditional no-prefetch-conditional}
-Kprefetch_iteration=N	-ffj-prefetch-iteration=N
-Kprefetch_iteration_L2=N	-ffj-prefetch-iteration-L2=N
-Kprefetch_line=N	-ffj-prefetch-line=N
-Kprefetch_line_L2=N	-ffj-prefetch-line-L2=N
-K{prefetch_sequential[={auto soft}] prefetch_nosequential}	-ffj-{prefetch-sequential[={auto soft}] no-prefetch-sequential}
-K{prefetch_stride prefetch_nostride}	-ffj-{prefetch-stride no-prefetch-stride}
-K{prefetch_strong prefetch_nostrong}	-ffj-{prefetch-strong no-prefetch-strong}
-K{prefetch_strong_L2 prefetch_nostrong_L2}	-ffj-{prefetch-strong-L2 no-prefetch-strong-L2}
-K{simd nosimd}	-f{vectorize no-vectorize}

Option name in Trad Mode	Option name in Clang Mode
-K{strict_aliasing nostrict_aliasing}	-f{strict-aliasing no-strict-aliasing}
-K{swp noswp}	-ffj-{swp no-swp}
-Kswp_strong	-ffj-swp
-Kswp_weak	
-K{unroll nounroll}	-f{unroll-loops no-unroll-loops}
-K{zfill[=N] nozfill}	-ffj-{zfill[=N] no-zfill}
-N{exceptions noexceptions}	-f{exceptions no-exceptions}
-N{fjcex nofjcex}	-ffj-{fjcex no-fjcex}
-N{fjprof nofjprof}	-ffj-{fjprof no-fjprof}
-N{hook_time nohook_time}	-ffj-{hook-time no-hook-time}
-N{line noline}	-ffj-{line no-line}
-Nlst	-ffj-lst
-Nlst=p	-ffj-lst=p
-Nlst=t	-ffj-lst=t
-Nlst_out=file	-ffj-lst-out=file
-Nsrc	-ffj-src
-V	--version
{-x- -x0}	-f{inline-functions no-inline-functions}

9.1.2.3.2 Side Effect of Optimizations for Floating-Point Operation


Optimizations for floating-point operation might cause the side effect. This section explains the side effect (mainly, computation error).

This system basically creates objects which comply with IEEE 754 arithmetic. Note that the numerical operations may not comply with IEEE 754 arithmetic due to the optimizations with calculation errors in "[Table 9.7 Side effect of optimization for floating-point operation](#)".

See Section "[9.1.2.2.3 Options for Optimization](#)" for compiler options. See Section "[9.2.2.1.2 Optimization Control Specifier](#)" for optimization control specifiers.

Table 9.7 Side effect of optimization for floating-point operation

Compiler Option	Optimization Control Specifier	Side Effect
-ffj-fast-matmul	-	Calculation errors may occur when using high speed library call for the loop of matrix multiplication.
-ffj-ilfunc[={loop procedure}]	-	Side effects similar to the ones caused by -ffj-fp-relaxed may occur when using inline expansion for the math functions, because reciprocal approximation instructions and trigonometric instructions, etc. are used. Plus, use of reciprocal approximation instructions or Floating-Point Multiply-Add/Subtract instructions regardless of set of -ffj-contract=off and/or -ffj-no-fp-relaxed options or their setting order.

Compiler Option	Optimization Control Specifier	Side Effect
<code>-ffj-fp-relaxed</code>	-	<p>Side effects may occur because reciprocal approximation operation instructions are used on single-precision or double-precision floating point division or <code>sqrt</code> functions.</p> <p>The side effects that may occur are:</p> <ul style="list-style-type: none"> - Rounding errors. - Replacing denormalized numbers found in the arguments or the return value with zero. - Replacing negative zeroes found in the arguments or the return value with positive zeroes. - Behaviors not conforming to IEEE 754 when NaN, positive or negative Inf, numbers which are close to maximal normalized number or numbers which are close to minimal normalized number are found in the arguments or the return value.
<code>-Ofast</code>	-	-Ofast option induces the <code>-ffast-math</code> option, <code>-ffj-ilfunc</code> option, and so on.
<code>-ffast-math</code>	-	<p>Calculation errors may occur when optimization that changes the method of operator evaluation. Moreover, the behaviors may not conforming to IEEE754.</p> <p> Example</p> <p>.....</p> <div style="border: 1px solid black; padding: 5px; width: fit-content;"> $x*y + x*z \rightarrow x*(y+z)$ </div> <p>.....</p> <p>Moreover, flush-to-zero mode is used by <code>-ffast-math</code> option. If a result or a source operand is a denormalized number, flush-to-zero mode replaces it with zero with the same sign.</p>
<code>-ffp-contract={fast on}</code>	<code>fp contract({fast on})</code>	Rounding errors may occur when optimization using Floating-Point Multiply-Add/Subtract operation instructions is performed on source programs.

-: None

Since `#pragma STDC FENV_ACCESS ON` is not supported in this system, even if `#pragma STDC FENV_ACCESS ON` is specified in the program, a floating-point exception, that is not raised based on the logic of the program, may be raised at execution time. In this case, the following phenomena may occur.

- If a program uses the `fetestexcept` function included in standard library functions and accesses a floating-point status flag by it, the flag which should not be set based on the logic of the program is set.
- If a program uses the `feenableexcept` function included in the GNU C Library and enables a trap of a floating-point exception by it, the `SIGFPE` signal which should not be raised based on the logic of the program is raised.

This phenomenon may be avoided by setting the compiler option `-O0`, which suppresses optimizations.

9.1.2.3.3 Note on Using SVE

Even if the `-ffj-prefetch-sequential` or `-ffj-prefetch-stride` options are set, the prefetch instruction will not be generated for the loops which meet all of the following conditions:

- The induction variable increment is greater than 1.
- The size of the array type is less than 8 bytes.

9.1.2.3.4 Notes on Using SIMD Built-in Functions

This section provides notes on using SIMD built-in functions.

- If the shared library contains a function that has the SIMD built-in function with vector type as an argument, take one of the following actions. If the following actions are not taken, errors may occur in the execution results.
 - Specify the `-Wl,-z,now` options when the program linked.
 - Set 1 to the environment variable `LD_BIND_NOW` at runtime,

Refer to "[9.1.6.1 Environment Variable for Execution](#)" for the environment variable `LD_BIND_NOW`.



Example

SIMD built-in function with vector type as an argument

```
#include <arm_sve.h>
extern void subl(svint64_t p_val);
int main() {
    svint64_t p_val;
    p_val = svdup_n_s64(123);
    subl(p_val);    /* Call function with vector type as argument */
    return 0;
}
```

- Note the SIMD built-in functions cannot be used when the `-msve-vector-bits=512` option is specified.

9.1.2.3.5 Notes on Specified SVE Vector Register Size

`-msve-vector-bits=512` option specifies the bitwise SVE vector register size. When this option is specified, the object program is generated considering the SVE vector register size as a fixed value at compilation. Therefore, the generated executable program works normally on the CPU architecture which has the same size of the SVE vector register as the specified size.

When `-msve-vector-bits=512` option is specified, and the program is executed on the CPU whose implemented SVE vector register size is different from 512 bits, an abnormal end occurs. And, the result of executions is not guaranteed.

When `-msve-vector-bits=scalable` option is effective, the executable program does not depend on the SVE vector register size.

See also Section "[C.2.8 Changing SVE Vector Register Size](#)" for notes on changing the SVE vector register size.

9.1.3 Environment Variable for Compile Command

This section explains environment variables that are valid only in Clang Mode.

For information about common environment variables in Clang/Trad Mode, refer to "[2.3 Environment Variable for Compile Command](#)".

`fccpx_clang_ENV`
`fcc_clang_ENV`

Environment variables to set compiler options.

The environment variable `fccpx_clang_ENV` is for the cross compiler and the environment variable `fcc_clang_ENV` is for the native compiler.

The user can specify Clang Mode specific and Clang/Trad Mode common compiler options for the value of these environment variables. Refer to "[9.1.2.3.1 Correspondences of Compile Options in Clang Mode and Trad Mode](#)" for a description of the common compiler options for Clang/Trad Mode.

Compiler options specified to these environment variables are valid only in Clang Mode.

Refer to "[2.4 Compilation Profile File](#)" for the priority of compiler options.

9.1.4 Compilation Profile File

The default values of compiler options can be changed by specifying the compilation profile file.

The following compilation profile files are available:

Table 9.8 Compilation Profile Files (Clang Mode)

Kind	Mode	File Name
Cross compiler	Clang/Trad Mode common	/etc/opt/FJSVxtclanga/fccpx_PROF
	Clang Mode specific	/etc/opt/FJSVxtclanga/fccpx_clang_PROF
Native compiler	Clang/Trad Mode common	/etc/opt/FJSVxtclanga/fcc_PROF
	Clang Mode specific	/etc/opt/FJSVxtclanga/fcc_clang_PROF

Common compilation profile files for Clang/Trad Mode can specify common compiler options for Clang/Trad Mode. Refer to "[9.1.2.3.1 Correspondences of Compile Options in Clang Mode and Trad Mode](#)" for a description of the common compiler options for Clang/Trad Mode.

Clang Mode specific compilation profile files can specify Clang Mode specific and Clang/Trad Mode common compiler options.

For information about description formats of the compilation profile file and the priority of compiler options, refer to "[2.4 Compilation Profile File](#)".

9.1.5 Predefined Macro Names

Some of the predefined macros are changed by the compiler option `-std=name`.

The following tables show the values of some predefined macros. To display all predefined macros, specify the compiler options `-E` and `-dM`.

Table 9.9 Predefined Macros whose value is changed by the compiler option `-std=name`

Identifier	The level of language specifications specified with the compiler option <code>-std=name</code>					
	c89	gnu89	c99	gnu99	c11	gnu11
<code>__STDC_VERSION__</code>	-		199901L		201112L	
<code>__STRICT_ANSI__</code>	1	-	1	-	1	-
<code>linux</code>	-	1	-	1	-	1
<code>unix</code>	-	1	-	1	-	1

-: Undefined

Table 9.10 Predefined Macros whose value is not changed by the compiler option `-std=name`

Identifier	Value
<code>_LP64</code>	1
<code>_OPENMP (*1)</code>	201511
<code>_REENTRANT (*2)</code>	1
<code>__ARM_ARCH</code>	8
<code>__ASSEMBLER__ (*3)</code>	1
<code>__CLANG_FUJITSU</code>	1
<code>__DATE__</code>	The date of compilation (in the format of the <code>asctime</code> function)
<code>__ELF__</code>	1
<code>__EXCEPTIONS (*4)</code>	1
<code>__FCC_major__</code>	The major version number of the compiler

Identifier	Value
__FCC_minor__	The minor version number of the compiler
__FCC_patchlevel__	The patch level of the compiler
__FCC_version__	The string which represents the version of the compiler
__FILE__	Source file name
__GNUC__	4
__GNUC_MINOR__	2
__GNUC_PATCHLEVEL__	1
__LINE__	Line number of source file
__LP64__	1
__OPTIMIZE__ (*5)	1
__PIC__	1 (*6)
	2 (*7)
__PRAGMA_REDEFINE_EXTNAME	1
__PTRDIFF_TYPE__	long int
__SIZE_TYPE__	long unsigned int
__STDC__	1
__STDC_HOSTED__	1
__TIME__	The time of compilation (in the format of the asctime function)
__USER_LABEL_PREFIX__	null string
__WCHAR_TYPE__	unsigned int (*8)
__aarch64__	1
__clang_major__	The major version number of Clang/LLVM
__clang_minor__	The minor version number of Clang/LLVM
__clang_patchlevel__	The patch level of Clang/LLVM
__clang_version__	The string which represents the version of Clang/LLVM
__linux	1
__linux__	1
__pic__	1 (*6)
	2 (*7)
__unix	1
__unix__	1

-: Undefined

*1) When the compiler option -Kopenmp or -fopenmp is set

*2) When the compiler option -pthread is set

*3) When the compiler option -x assembler-with-cpp is set, or when the suffix of the input file is ".S"

*4) When the compiler option -Nexceptions or -fexceptions is set

*5) When the compiler option -O1 or higher is set

*6) When the compiler option -fpic or -fpie is set

*7) When the compiler option -fPIC or -fPIE is set

*8) This value is different from Trad Mode, and unintended behavior may occur when you execute a program linking an object generated in Trad Mode and an object generated in Clang Mode.

9.1.6 Procedure of Execution

This section explains the procedure of executing C language programs.

9.1.6.1 Environment Variable for Execution

Runtime controls can be changed by setting environment variables. "Table 9.11 Environment Variables for Execution (Clang Mode)" shows the environment variables that can be set at runtime.

For details about OpenMP environment variables, see Section "4.3.2.2 Environment Variable for OpenMP Specifications".

Table 9.11 Environment Variables for Execution (Clang Mode)

Environment Variable Name	Operands	Description
FLIB_HOOK_TIME	<i>time</i>	The user-defined function is called at interval of <i>time</i> millisecond(s). <i>time</i> can be a value between 0 and 2147483647. If 0 is specified for <i>time</i> , calling at regular interval is disabled. See Section "8.3 Hook Function", for information about the hook function.
LD_BIND_NOW	1	Indicates not to use the delay link for symbol resolution of shared library addresses.

9.1.6.2 Notes for Execution

When executing programs created on this system, note the following.

- Variable allocation at execution

Programs created by this system allocate the local and private variables in each function to the stack region. When much more space is required for local and private variables in the function, the stack area needs to be extended to the appropriate size.

The stack area of the process can be set by `ulimit` (bash built-in command), etc.

9.2 Optimization

9.2.1 Overview of Optimization

Refer to the Section "3.1 Overview of Optimization" for overview of optimization.



Note

Result of optimization

The optimization result by Clang Mode may be different from that of Trad Mode.

Compilation message

In Clang Mode, the messages are in English only. And, the message format of Clang Mode is different from that of Trad Mode.

9.2.2 Using Optimization Functions

This section describes techniques for effectively using the optimization functions.

9.2.2.1 Using Optimization Control Line (Pragma Directives)

It may be possible to increase the optimization level using certain `#pragma` directives in the source code.

In Clang Mode, the following two types of pragma directives can be used.

- Fujitsu-specific pragma supported by Clang Mode
- Clang/LLVM pragma supported by Clang Mode

The -ffj-ocl option enables the Fujitsu-specific pragma supported by Clang Mode, and the -ffj-no-ocl option suppresses its effect.

The Clang/LLVM pragma supported by Clang Mode is always valid.



Note

The result is not guaranteed when using the pragma not listed in this section.

9.2.2.1.1 Types of Optimization Control Lines

The optimization control line of Fujitsu-specific pragma supported by Clang Mode can be specified in the following format. Enables the specified optimization control specifier for just before the loop.

Description Format

```
#pragma fj Optimization-control-specifier
```

Insert Position

This is positioned just before the corresponding loop.

Also, the loop line of the Clang/LLVM pragma or another Fujitsu-specific pragma can be interposed between the loop and this pragma.

For Clang/LLVM pragma supported by Clang Mode, the following format can be used. Enables the specified optimization control specifier for just before the loop or the compound statement.

Description Format

```
#pragma clang Optimization-control-specifier
```

Insert Position

- In the case of an optimization control specifier that begins with loop

Put it just before the corresponding loop.

Also, the loop line of the Clang/LLVM pragma or another Fujitsu-specific pragma can be interposed between the loop.

- In the case of an optimization control specifier that begins with fp

Put it before all explicit declarations and statements inside the corresponding compound statement.

9.2.2.1.2 Optimization Control Specifier

The optimization control specifier that can be specified in Clang Mode are shown in "[Table 9.12 Optimization Control Specifiers that can be specified to optimization control line](#)".



Note

The optimization control specifier of Trad Mode is replaced with that of Clang Mode.

The list of the optimization control specifier to be replaced is shown in "[Table 9.13 Optimization control specifiers replaced in Clang Mode](#)".

If the optimization control specifier that is not listed in "[Table 9.12 Optimization Control Specifiers that can be specified to optimization control line](#)" is specified, it is ignored.

Table 9.12 Optimization Control Specifiers that can be specified to optimization control line

Optimization Control Specifiers	Format of pragma	Explanation
loop clone var==n(*1)	#pragma fj loop clone var==n	Directs to generate conditional branches on the expression of the specified arguments and to generate loop copies in the conditional blocks.
loop eval_concurrent	#pragma fj loop eval_concurrent	Performs the optimization that improves the parallelism of floating-point arithmetic instructions using the optimization of tree-height-reduction. The optimization performed by -ffj-eval-concurrent option can be specified at the loop unit.
loop eval_noconcurrent	#pragma fj loop eval_noconcurrent	Suppresses the optimization that is performed by -ffj-eval-concurrent option.
loop loop_fission_target	#pragma fj loop loop_fission_target [c1]	Performs automatic loop fission optimization. This specifier can be specified only for innermost loops. The compiler behavior when c1 is written is the same as when not written.
loop loop_fission_threshold n	#pragma fj loop loop_fission_threshold n	Specifies the threshold <i>n</i> to decide the granularity of loop after automatic loop fission. <i>n</i> is an integer value from 1 to 100. This specifier can be specified only for innermost loops.
loop preex	#pragma fj loop preex	Optimizes by evaluating an invariant first. The optimization performed by -ffj-preex option can be specified at the loop unit.
loop nopreex	#pragma fj loop nopreex	Suppresses the optimization of evaluating an invariant first.
loop prefetch	#pragma fj loop prefetch	Performs the automatic prefetch function of the compiler. The optimizations performed by -ffj-prefetch-sequential or -ffj-prefetch-stride option can be specified at the loop unit.
loop noprefetch	#pragma fj loop noprefetch	Suppresses the automatic prefetch function of the compiler.
loop prefetch_sequential [auto soft]	#pragma fj loop prefetch_sequential [auto soft]	Directs that prefetch instructions are created for array data that is accessed sequentially. The optimizations performed by -ffj-prefetch-sequential option can be specified at the loop unit.
loop prefetch_nosequential	#pragma fj loop prefetch_nosequential	Directs that prefetch instructions should not be generated for array data that is accessed sequentially.
loop prefetch_stride	#pragma fj loop prefetch_stride	Directs that prefetch instructions are created for array data accessed with a stride larger than the cache line size used in the loop.
loop prefetch_nostride	#pragma fj loop prefetch_nostride	Directs that prefetch instructions for stride access should not be generated.
loop prefetch_strong	#pragma fj loop prefetch_strong	Directs that the prefetch instructions for the first level cache are to be the strong prefetch. The optimizations performed by -ffj-prefetch-strong option can be specified at the loop unit.

Optimization Control Specifiers	Format of pragma	Explanation
loop prefetch_nostrong	#pragma fj loop prefetch_nostrong	Directs that the prefetch instructions generated for the first level cache will not be strong prefetch.
loop prefetch_strong_L2	#pragma fj loop prefetch_strong_L2	Directs that the prefetch instructions generated for the second level cache are to be strong prefetch. The optimizations performed by -ffj-prefetch-strong-L2 option can be specified at the loop unit.
loop prefetch_nostrong_L2	#pragma fj loop prefetch_nostrong_L2	Directs that the prefetch instructions generated for the second level cache are not to be strong prefetch.
loop swp	#pragma fj loop swp	Performs software pipelining.
loop noswp	#pragma fj loop noswp	Suppresses software pipelining.
loop zfill [N]	#pragma fj loop zfill [N]	Directs that the zfill optimization to be performed. The optional parameter <i>N</i> is an integer between 1 and 100 that specifies the number of blocks the DC ZVA instruction writes.
loop nozfill	#pragma fj loop nozfill	Directs that the zfill optimization not to be performed.
fp contract(fast)	#pragma clang fp contract(fast)	Performs optimizations using the Floating-Point Multiply-Add/Subtract instructions. The optimizations performed by -ffp-contract=fast option can be specified at the section unit.
fp contract(on)	#pragma clang fp contract(on)	Performs optimizations using the Floating-Point Multiply-Add/Subtract instructions. This pragma is identical to using #pragma STDC FP_CONTRACT(ON).
fp contract(off)	#pragma clang fp contract(off)	Suppresses optimizations using the Floating-Point Multiply-Add/Subtract instructions.
loop unroll(enable)	#pragma clang loop unroll(enable)	Performs unrolling for the corresponding loop.
loop unroll_count(n)	#pragma clang loop unroll_count(n)	The optimizations performed by -funroll-loops option can be specified at the loop unit. When unroll(full) is set, all statements in the target loop is unrolled.
loop unroll(full)	#pragma clang loop unroll(full)	
loop unroll(disable)	#pragma clang loop unroll(disable)	Suppresses unrolling for the corresponding loop.
loop vectorize(assume_safety)	#pragma clang loop vectorize(assume_safety)	Indicates that the loop has no dependencies on array elements or pointer variables.
loop vectorize(enable)	#pragma clang loop vectorize(enable)	Performs the optimization that uses SIMD Extensions. The optimizations performed by -fvectorize option can be specified at the loop unit. When vectorize_width(n, scalable) is set to use SIMD Extensions through SVE, the SIMD width becomes a multiple of <i>n</i> .
loop vectorize_width(n, scalable)	#pragma clang loop vectorize_width(n, scalable)	
loop vectorize(disable)	#pragma clang loop vectorize(disable)	Suppresses the optimization that uses SIMD Extensions.

*1) *var* must be declared before it is used.

loop clone specifier

The `loop clone` specifier instructs to generate conditional branches on the expression of the specified arguments and to generate loop copies in the conditional blocks. If multiple `loop clone` specifiers are specified for the same loop, the order of the generated branches is the order of the corresponding arguments. For the nested loops, the `loop clone` specifier cannot be specified for loops of different nesting levels. This specifier promotes other optimizations, such as full unrolling. Note that the size of the object program and compile time may increase because the clone optimization makes copies of loops. This specifier is effective when the `-O3` option is set.

Variable *var* is a variable of type `int`. You cannot specify following variable of type `int`.

- Structure member variable
- Union member variable
- Array elements
- Threadprivate variable

An integer value from -2147483647 to 2147483647 can be specified as constant values *n*.



Example

Specification of the optimization control line

- Example 1: Specifying only one `loop clone` specifier

```
#pragma fj loop clone N==10
for(i = 0; i < N; ++i) {
    a[i] = i;
}
```

[Optimized pseudo-code]

```
if(N == 10) {
    for(i = 0; i < N; ++i) {
        a[i] = i;
    }
} else {
    for(i = 0; i < N; ++i) {
        a[i] = i;
    }
}
```

Loop is copied with "if" statements in the order of the specification.

- Example 2: Specifying multiple `loop clone` specifiers for the same loop

```
#pragma fj loop clone N==10
#pragma fj loop clone M==20
for(i = 0; i < N; i++) {
    a[i] = M;
}
```

[Optimized pseudo-code]

```
if(N == 10) {
    for(i = 0; i < N; ++i) {
        a[i] = M;
    }
} else if (M == 20) {
    for(i = 0; i < N; ++i) {
        a[i] = M;
    }
}
```

```

} else {
    for(i = 0; i < N; ++i) {
        a[i] = M;
    }
}

```

Loop is copied with "if" statements in the order of the multiple specifications.

- Example 3: Using a flag to specify loop clone specifier

```

int flag = (start == 0 && end == 10) ? 1 : 0;
#pragma fj loop clone flag==1
for(i = start; i < end; ++i) {
    a[i] = i;
}

```

[Optimized pseudo-code]

```

int flag = (start == 0 && end == 10) ? 1 : 0;
if(flag == 1) {
    for(i = start; i < end; ++i) {
        a[i] = i;
    }
} else {
    for(i = start; i < end; ++i) {
        a[i] = i;
    }
}

```

Loop is copied with "if" statements in the order of the specifications.

- Example 4: Incorrect specification of the loop clone specifier

```

#pragma fj loop clone M==10
for(i = 0; i < M; ++i) {
    #pragma fj loop clone N==20
    for(j = 0; j < N; ++j) {
        a[i][j] = 0;
    }
}

```

For the nested loops, the loop clone specifier cannot be specified for loops of different nesting levels.

loop eval_concurrent specifier

The loop eval_concurrent specifier instructs to perform the optimization that improves the parallelism of floating-point arithmetic instructions using the optimization of tree-height-reduction.



Example

Specification of the optimization control line

```

#pragma fj loop eval_concurrent
for(i = 0; i < n; i++) {
    x[i] = a[i] * b[i] + c[i] * d[i] + e[i] * f[i] + g[i] * h[i];
}

```

loop eval_noconcurrent specifier

The loop eval_noconcurrent specifier instructs to suppress the optimization that improves the parallelism of floating-point arithmetic instructions using the optimization of tree-height-reduction.



Example

Specification of the optimization control line

```
#pragma fj loop eval_noconcurrent
for(i = 0; i < n; i++) {
    x[i] = a[i] * b[i] + c[i] * d[i] + e[i] * f[i] + g[i] * h[i];
}
```

loop loop_fission_target specifier

The loop `loop_fission_target` specifier instructs to perform automatic loop fission for target loop.

This specifier can be specified only for innermost loops.

The compiler behavior when `cl` is written after the specifier is the same as when not written.



Example

Specification of the optimization control line

```
#pragma fj loop loop_fission_target
for (i = 0; i < n; i++) {
    ...
}
```

loop loop_fission_threshold specifier

The loop `loop_fission_threshold` specifier specifies the granularity of loop after automatic loop fission by the threshold `n` written after the specifier. `n` is an integer value from 1 to 100.

This specifier can be specified only for innermost loops.



Example

Specification of the optimization control line

```
#pragma fj loop loop_fission_target
#pragma fj loop loop_fission_threshold 10
for (i = 0; i < n; i++) {
    ...
}
```

loop preex specifier

The loop `preex` specifier instructs to optimize of evaluating the invariant first.



Example

Specification of the optimization control line

```
#pragma fj loop preex
for(i = 0; i < n; i++) {
    if(m[i] != 0) {
        a[i] = a[i] / b[k];
    }
}
```

[Optimized pseudo-code]

```
t = 1 / b[k];
for(i = 0; i < n; i++) {
    if(m[i] != 0) {
        a[i] = a[i] * t;
    }
}
```

The optimization of evaluating the invariant first is performed in the target loop.

loop nopreex specifier

The loop nopreex specifier instructs to suppress the optimization of evaluating the invariant first.



Example

Specification of the optimization control line

```
#pragma fj loop nopreex
for(i = 0; i < n; i++) {
    if(m[i] != 0) {
        a[i] = a[i] / b[k];
    }
}
```

The optimization of evaluating the invariant first is suppressed in the target loop.

loop prefetch specifier

The loop prefetch specifier instructs to perform the automatic prefetch function. This function inserts the prefetch instruction in the optimal position to execute, as determined by the compiler.



Example

Specification of the optimization control line

```
#pragma fj loop prefetch
for(i = 0; i < 10; i=i+m) {
    a[i] = b[i];
}
```

loop noprefetch specifier

The loop noprefetch specifier suppresses the automatic prefetch function.



Example

Specification of the optimization control line

```
#pragma fj loop noprefetch
for(i = 0; i < 10; i=i+m) {
    a[i] = b[i];
}
```

The automatic prefetch function is suppressed in the loop.

loop prefetch_sequential specifier

The `loop prefetch_sequential` specifier directs that prefetch instructions are created for array data that is accessed sequentially.

When `prefetch_sequential auto` specifier is used, the compiler automatically selects whether to use hardware-prefetch or to create prefetch instructions for array data that is accessed sequentially in the loop.

When `prefetch_sequential soft` specifier is used, the compiler does not use hardware-prefetch, but rather creates prefetch instructions for array data that is accessed sequentially in the loop.

If neither `auto` nor `soft` is specified, `auto` is set by default.



Example

Specification of the optimization control line

- Example 1:

```
#pragma fj loop prefetch_sequential auto
for(i = 0; i < n; i++) {
    a1[i] = a2[i] + a3[i] + a4[i] + a5[i]
    + a6[i] + a7[i] + a8[i] + a9[i] + a10[i]
    + a11[i] + a12[i] + a13[i] + a14[i] + a15[i]
    + a16[i] + a17[i];
}
```

Hardware prefetch is used for the target loop rather than generating prefetch instructions.

- Example 2:

```
#pragma fj loop prefetch_sequential soft
for(i = 0; i < n; i++) {
    a[i] = a[i] + b[i];
}
```

Prefetch instructions are generated for the target loop rather than using hardware-prefetch.

loop prefetch_nosequential specifier

The `loop prefetch_nosequential` specifier directs that prefetch instructions should not be generated for array data that is accessed sequentially.



Example

Specification of the optimization control line

```
#pragma fj loop prefetch_nosequential
for(i = 0; i < n; i++) {
    a[i] = a[i] + b[i];
}
```

Prefetch instructions are not generated for array data that is accessed sequentially in the target loop.

loop prefetch_stride specifier

The `loop prefetch_stride` specifier directs that prefetch instructions are created for array data accessed with a stride larger than the cache line size used in the loop.



Example

Specification of the optimization control line

```
#pragma fj loop prefetch_stride
for(i = 0; i < n; i=i+m) {
    a[i] = b[i];
}
```

Prefetch instructions are generated for the target loop.

loop prefetch_nostride specifier

The `loop prefetch_nostride` specifier directs that prefetch instructions are not created for array data accessed with a stride larger than the cache line size used in the loop.



Example

Specification of the optimization control line

```
#pragma fj loop prefetch_nostride
for(i = 0; i < n; i=i+m) {
    a[i] = b[i];
}
```

Prefetch instructions are not generated for the target loop.

loop prefetch_strong specifier

The `loop prefetch_strong` specifier directs that prefetch instructions for the first level cache are to be the strong prefetch.



Example

Specification of the optimization control line

```
#pragma fj loop prefetch_strong
for(i = 0; i < n; i++) {
    a[i] = a[i] + b[i];
}
```

Prefetch instructions for the first level cache generated for the target loop will be strong prefetch.

loop prefetch_nostrong specifier

The `loop prefetch_nostrong` specifier directs that prefetch instructions generated for the first level cache will not be strong prefetch.



Example

Specification of the optimization control line

```
#pragma fj loop prefetch_nostrong
for(i = 0; i < n; i++) {
    a[i] = a[i] + b[i];
}
```

Prefetch instructions for the first level cache generated for the target loop will not be strong prefetch.

loop prefetch_strong_L2 specifier

The `loop prefetch_strong_L2` specifier directs that prefetch instructions for the second level cache are to be the strong prefetch.



Example

Specification of the optimization control line

```
#pragma fj loop prefetch_strong_L2
for(i = 0; i < n; i++) {
    a[i] = a[i] + b[i];
}
```

Prefetch instructions for the second level cache generated for the target loop will be strong prefetch.

loop prefetch_nostrong_L2 specifier

The `loop prefetch_nostrong_L2` specifier directs that prefetch instructions generated for the first level cache will not be strong prefetch.



Example

Specification of the optimization control line

```
#pragma fj loop prefetch_nostrong_L2
for(i = 0; i < n; i++) {
    a[i] = a[i] + b[i];
}
```

Prefetch instructions for the second level cache generated for the target loop will not be strong prefetch.

loop swp specifier

The `loop swp` specifier performs software pipelining.



Example

Specification of the optimization control line

```
#pragma fj loop swp
for (i = 0; i < n; i++) {
    a[i] = b[i]/c[i];
}
```

loop noswp specifier

The `loop noswp` specifier suppresses software pipelining.



Example

Specification of the optimization control line

```
#pragma fj loop noswp
for (i = 0; i < n; i++) {
    a[i] = b[i]/c[i];
}
```

loop zfill specifier

The `loop zfill` specifier directs to apply the zfill optimization. The zfill optimization speeds up write operations for array data that is only written in a loop, by using an instruction that allocates space on the cache for writing without loading data from the memory. The zfill optimization works on the data N blocks ahead of the address pointed to by the target store instruction where one block is 256 byte-long and N is an integer value between 1 and 100. If a value is not specified for N , the compiler will automatically determine a value.

For details about zfill, see "[3.3.5 zfill](#)".



Example

Specification of the optimization control line

- Example 1

```
#pragma fj loop zfill
for(i = 0; i < n; i++) {
    ...
}
```

This specifies to improve the cache utilization for the data on certain blocks ahead, and the number is determined by the compiler.

- Example 2

```
#pragma fj loop zfill 1
for(i = 0; i < n; i++) {
    ...
}
```

This specifies to improve the cache utilization for the data on 1 block ahead.

loop nozfill specifier

The `loop nozfill` specifier directs not to apply the zfill optimization.



Example

Specification of the optimization control line

```
#pragma fj loop nozfill
for(i = 0; i < n; i++) {
    ...
}
```

This specifies that the zfill optimization will not be performed.

fp contract(fast) specifier

The `fp contract(fast)` specifier instructs to apply optimizations using the Floating-Point Multiply-Add/Subtract instructions. However, using these instructions may cause calculation errors in the result due to rounding errors.



Example

Specification of the optimization control line

```
for(i = 0; i < n; i++) {
    #pragma clang fp contract(fast)
    a[i] = a[i] + b[i] * c[i];
}
```

The optimization that uses the Floating-Point Multiply-Add/Subtract instructions is performed for the target compound statement.

fp contract(on) specifier

The `fp contract(on)` specifier instructs to apply optimizations using the Floating-Point Multiply-Add/Subtract instructions. It is performed as if the `#pragma STDC FP_CONTRACT ON` (language standard) was specified. However, using these instructions may cause calculation errors in the result due to rounding errors.



Example

Specification of the optimization control line

```
for(i = 0; i < n; i++) {  
    #pragma clang fp contract(on)  
    a[i] = a[i] + b[i] * c[i];  
}
```

The optimization that uses the Floating-Point Multiply-Add/Subtract instructions is performed for the target compound statement.

fp contract(off) specifier

The `fp contract(off)` specifier instructs to suppress optimizations using the Floating-Point Multiply-Add/Subtract instructions.



Example

Specification of the optimization control line

```
for(i = 0; i < n; i++) {  
    #pragma clang fp contract(off)  
    a[i] = a[i] + b[i] * c[i];  
}
```

The optimization that uses the Floating-Point Multiply-Add/Subtract instructions is not performed for the target compound statement.

loop unroll(enable) specifier

The `loop unroll(enable)` specifier instructs the compiler to perform the optimization of loop unrolling for the corresponding loop. The number to be unrolled is determined by the compiler automatically.

Note that the `loop unroll(enable)` specifier targets only the loop specified immediately after.



Example

Specification of the optimization control line

```
#pragma clang loop unroll(enable)  
for(i = 0; i < n; i++) {  
    ...  
}
```

loop unroll(full) specifier

The `loop unroll(full)` specifier instructs the compiler to perform the optimization of loop unrolling for the corresponding loop.

When the `loop unroll(full)` specifier is used, the statement is unrolled up to the number of the iteration in the specified loop.

If the number of the iteration is unknown at compilation, no optimization of loop unrolling is performed.

Note that the `loop unroll(full)` specifier targets only the loop specified immediately after.



Example

Specification of the optimization control line

```
#pragma clang loop unroll(full)
for(i = 0; i < 8; i++) {
    ...
}
```

All statements in the target loop is unrolled.

loop unroll_count specifier

The number from 2 to 100 following the specifier instructs the upper bound on the number of loops to be unrolled.

Cannot omit that upper bound.

Note that the `loop unroll_count` specifier targets only the loop specified immediately after.



Example

Specification of the optimization control line

```
#pragma clang loop unroll_count(8)
for(i = 0; i < n; i++) {
    ...
}
```

The "statement" is unrolled 8 times in the target loop.

loop unroll(disable) specifier

The `loop unroll(disable)` specifier instructs the compiler to suppress unrolling for the corresponding loop.

Note that the `loop unroll(disable)` specifier targets only the loop specified immediately after.



Example

Specification of the optimization control line

```
#pragma clang loop unroll(disable)
for(i = 0; i < n; i++) {
    ...
}
```

Loop unrolling is suppressed in the target loop.

loop vectorize(assume_safety) specifier

The `loop vectorize(assume_safety)` specifier indicates that there is no dependency on an element of the array or the pointer variable in the loop. This specification causes SIMD Extensions to be performed on loops that cannot be optimized because the data dependency is unknown. Depending on the operation type and the loop structure, it may not be subject to SIMD.



Note

If the loop `vectorize(assume_safety)` specifier is incorrectly specified in a loop for the element of the dependent array or the pointer variable, the result will be unpredictable.



Example

Specification of the optimization control line

```
#pragma clang loop vectorize(assume_safety)
for (int i = 0; i < count; i++) {
    a[index[i]] = a[index[i]] + 1;
}
```

loop vectorize(enable) specifier

The loop `vectorize(enable)` specifier instructs to perform the optimization that uses SIMD Extensions. However, SIMD Extensions may not be used depending on the type of operation and the loop structure.

The compiler automatically determines a suitable SIMD width.



Example

Specification of the optimization control line

```
double a[10], b[10];
...
#pragma clang loop vectorize(enable)
for(i = 0; i < 10; i++) {
    a[i] = a[i] + b[i];
}
```

The `simd` specifier is in effect for the data in the loop.

loop vectorize_width(n, scalable) specifier

The loop `vectorize(n, scalable)` specifier instructs to perform the optimization that uses SIMD Extensions through SVE. The SVE vector register is not considered to be a specific size. However, SIMD Extensions may not be used depending on the type of operation and the loop structure.

This specifier is effective only when the `-msve-vector-bits=scalable` option is effective.

The number of elements of data which one SIMD instruction processes if the size of the SVE vector register is 128-bit (the minimum) is specified by *n*. The actual number of elements of data which one SIMD instruction processes, which is called a SIMD width, is decided at execution time depending on the SVE vector register size on the CPU architecture. For example, if the SVE vector register size is 512-bit, the SIMD width is $(512/128)*n = 4*n$. If one SIMD instruction cannot process the SIMD width at once, the instruction is used more than once. You can specify 2 or 4 for *n*.



Example

Specification of the optimization control line

```
float a[100];
double b[100];
...
#pragma clang loop vectorize_width(4, scalable)
for(i = 0; i < 100; i++) {
    a[i] += 1.0;
}
```

```

    b[i] += 1.0;
}

```

If the SVE register size is 512-bit for example, data from a[0] to a[15], from b[0] to b[7], and from b[8] to b[15] are each processed in one SIMD instruction.

loop vectorize(disable) specifier

The `loop vectorize(disable)` specifier instructs to suppress the optimization that uses SIMD Extensions.



Example

Specification of the optimization control line

```

double a[10], b[10];
...
#pragma clang loop vectorize(disable)
for(i = 0; i < 10; i++) {
    a[i] = a[i] + b[i];
}

```


9.2.2.1.3 Notes on Optimization Control Lines and Optimization Control Specifiers



This section describes notes on the optimization control lines and the optimization control specifiers.

Differences in behaviors depending on types of optimization control lines


Depending on types of optimization control lines described in "9.2.2.1.1 Types of Optimization Control Lines", the compiler behaves differently in some cases.

- When multiple optimization control lines of the same type are specified at the same location

Specified optimization control specifiers		Compiler behavior	
		Clang/LLVM optimization control lines	Fujitsu-specific optimization control lines
Same optimization control specifiers	loop clone specifier	(No corresponding specifier)	It can be compiled. All loop clone specifiers are enabled.
	Others	It causes a compilation error.	It can be compiled.
Conflicting optimization control specifiers, such as applying and suppressing a certain optimization	Following optimization control specifiers - fp contract (fast) - fp contract (on) - fp contract (off)	It can be compiled. The last optimization control line is enabled.	The last optimization control line is enabled.
	Others  Example loop unroll (enable) specifier and loop unroll (disable) specifier	It causes a compilation error.	
Different optimization control specifiers for an optimization		It causes a compilation error.	

Specified optimization control specifiers	Compiler behavior	
	Clang/LLVM optimization control lines	Fujitsu-specific optimization control lines
 Example loop unroll(full) specifier and loop unroll_count(n) specifier		
Optimization control specifiers with different parameter values  Example loop unroll_count(n) specifier	It causes a compilation error.	

- When there is an error in an optimization control line

Type of the error	Compiler behavior	
	Clang/LLVM optimization control lines	Fujitsu-specific optimization control lines
Error in a specifier in an optimization control line or an element which follows a specifier  Example - Example 1 <pre>#pragma loop unrecognized_specifier</pre> - Example 2 <pre>#pragma loop prefetch_sequential unrecognized_parameter</pre>	It causes a compilation error.	It can be compiled. A warning message is output and the optimization control line is ignored.

Correspondences of optimization control specifiers in Clang Mode and Trad Mode

In Clang Mode and Trad Mode, the optimization control specifiers whose type of optimization is equivalent but name is different are described by the loop line format in "Table 9.13 Optimization control specifiers replaced in Clang Mode".

- When the optimization control specifiers of Trad Mode described in the table is used, that of Clang Mode is assumed to be used.
- An optimization result may be different even if the type of optimization is equivalent.
- If an optimization control specifier of Trad Mode which is not described in the table is used, it is ignored and a warning message is output.

Table 9.13 Optimization control specifiers replaced in Clang Mode

Trad Mode	Clang Mode
#pragma loop clone var==n	#pragma fj loop clone var==n ^[a]
#pragma loop eval_concurrent	#pragma fj loop eval_concurrent
#pragma loop eval_noconcurrent	#pragma fj loop eval_noconcurrent
#pragma loop loop_fission_target [cl]	#pragma fj loop loop_fission_target [cl]

Trad Mode	Clang Mode
#pragma loop loop_fission_threshold <i>n</i>	#pragma fj loop loop_fission_threshold <i>n</i>
#pragma loop preex	#pragma fj loop preex
#pragma loop nopreex	#pragma fj loop nopreex
#pragma loop prefetch	#pragma fj loop prefetch
#pragma loop noprefetch	#pragma fj loop noprefetch
#pragma loop prefetch_sequential [auto soft]	#pragma fj loop prefetch_sequential [auto soft]
#pragma loop prefetch_nosequential	#pragma fj loop prefetch_nosequential
#pragma loop prefetch_stride	#pragma fj loop prefetch_stride
#pragma loop prefetch_nostride	#pragma fj loop prefetch_nostride
#pragma loop prefetch_strong	#pragma fj loop prefetch_strong
#pragma loop prefetch_nostrong	#pragma fj loop prefetch_nostrong
#pragma loop prefetch_strong_L2	#pragma fj loop prefetch_strong_L2
#pramga loop prefetch_nostrong_L2	#pragma fj loop prefetch_nostrong_L2
#pragma loop swp	#pragma fj loop swp
#pragma loop noswp	#pragma fj loop noswp
#pragma loop zfill [<i>N</i>]	#pragma fj loop zfill [<i>N</i>]
#pragma loop nozfill	#pragma fj loop nozfill
#pragma[fj] loop simd ^[b]	#pragma clang loop vectorize(enable)
#pragma[fj] loop nosimd	#pragma clang loop vectorize(disable)
#pragma[fj] loop unroll	#pragma clang loop unroll(enable)
#pragma[fj] loop unroll <i>n</i>	#pragma clang loop unroll_count(<i>n</i>)
#pragma[fj] loop unroll "full"	#pragma clang loop unroll(full)
#pragma[fj] loop nounroll	#pragma clang loop unroll(disable)

[a] Trad Mode and Clang Mode have different notations, types, and values for their arguments.

[b] No argument

9.2.3 SIMD

Refer to the Section "[3.2.7.1 Normal SIMD](#)" for normal SIMD.

9.2.3.1 Math Functions that SIMD Extensions can be applied to

The following shows the math functions that SIMD extensions can be applied to when -fvectorize option is effective.

```
ceil, ceilf, copysign, copysignf, fabs, fabsf, floor, floorf, fmax, fmaxf, fmin, fminf, round, roundf,
trunc, truncf
```

The following shows the math functions that SIMD extensions can be applied to when -fvectorize option and -fbuiltin option are effective.

```
abs, cimag, cimagf, creal, crealf, fma, fmaf, nearbyint, nearbyintf, rint, rintf
```

The following shows the math functions that SIMD extensions can be applied to when -fvectorize option, -fbuiltin option, and -ffast-math option are effective.

```
cabs, cabsf, sqrt, sqrtf
```

When -fvectorize option, -fbuiltin option, and -ffj-ilfunc option are effective, SIMD extensions can be applied to the math functions that are subject to the inline expansion.

See also the description of the -ffj-ilfunc options in the section "[9.1.2.2.3 Options for Optimization](#)" for information on math functions that are subject to the inline expansion.

9.3 Multiprocessing

9.3.1 Overview of Multiprocessing

What is Multiprocessing?

In this document, multiprocessing means that one program is executed on two or more CPUs that can work independently at the same time. As used here, it does not mean executing two or more programs simultaneously.

Effect of Multiprocessing

The effect of multiprocessing is to reduce elapsed execution time by using two or more CPUs simultaneously. For instance, if a loop can be executed in parallel by dividing it, the execution time of this loop may be cut in half.

Although the elapsed time will usually decrease with multiprocessing, the total CPU time required to execute the program may increase. This is because the total CPU time is at least as large as the CPU time when the program is executed on a single processor, and the overhead time for multiprocessing may increase the total CPU time.

The cause of the overhead of parallel execution is the parallel region that is one executed by a team consisting of more than one thread. One parallelization loop is generated for one parallel region. The cost of generating the parallel region is the overhead of parallel execution.

The performance of a multiprocessing program is usually evaluated by the reduction of the elapsed time. The total CPU time of a program using multiprocessing is larger than that required by serial processing. This is because of the overhead time required by multiprocessing.

Requirements for Effective Multiprocessing

A computer environment that can use two or more CPUs simultaneously is necessary to reduce elapsed time by multiprocessing. A multiprocessing program can be executed on hardware with only a single CPU; however, the elapsed time will not be less than the execution time for a comparable program written without multiprocessing features. Moreover, even if the program is executed on hardware with two or more CPUs, shortening the elapsed time is difficult when other jobs are executing on the same CPUs. The reason for this is that the probability that two or more CPUs can be allocated at the same time decreases.

To achieve effective multiprocessing, it is necessary to run a multiprocessing program on hardware with multiple CPUs and within a system environment that has room for multiprocessing.

To reduce the relative overhead of multiprocessing, it is necessary that a loop have many iterations and many statements in its body.

9.3.2 Parallelization by OpenMP Specification

See Section "[4.3 Parallelization by OpenMP Specification](#)" for the multiprocessing.

9.4 Emitting Information

9.4.1 Emitting Information at Compilation

This section explains the information emitted by this system at compilation.

9.4.1.1 Header

When either the `-ffj-lst`, `-ffj-lst-out=file` or `-ffj-src` compilation option is specified, a header is emitted for each type of information.

Header Format

Fujitsu C/C++ Version <i>version date</i>	
<i>version</i>	Language processing system version.
<i>date</i>	Compilation date and time in the format of the <code>asctime</code> function.

9.4.1.2 Source List

When the program is compiled, the source list is put out according to the following options.

- `-ffj-lst[={p|t}]`

Specifies to output source list to file(s).

- `-ffj-lst-out=file`

Specifies the filename to output source list.

When this option is specified, the `-ffj-lst=p` option is also effective.

- `-ffj-src`

Specifies to output source list to the standard output.

Note that when the `-ffj-lst` or `-ffj-lst-out=file` option is specified, the source list is output to the *file*.

The source list includes symbols indicating optimization if they are performed.

9.4.1.2.1 Output Format

The source list is emitted as shown in the following format.

Source List Format

Compilation information	
Current directory : <i>directory-name</i>	
Source file : <i>source-file-name</i>	
(line-no.)(optimize)	
<i>nnnnnnnn i mmmmv source</i>	
...	
<<< Loop-information Start >>>	
<<< <i>details optimization information</i>	
<<< Loop-information End >>>	
<i>nnnnnnnn i mmmmv source</i>	
...	

<i>directory-name</i>	The name of directory where source file is stored
<i>source-file-name</i>	Source file name
<i>nnnnnnnn</i>	Line number of source (variable length)
<i>i</i>	Symbols for inline expansion
<i>mmm</i>	The number of loop unrolling (variable length)

<i>v</i>	Symbols for using SIMD Extensions
<i>source</i>	Source line
<<< Loop-information Start >>> ^[a]	Details of optimization and parallelization information header
<i>details optimization information</i> ^[a]	Details of optimization and parallelization information which has been performed to the next statement.
<<< Loop-information End >>> ^[a]	Details of optimization and parallelization information footer

[a] Emitted only if the -ffj-lst=t option is effective.

9.4.1.2.2 Information Included in Source List

Line number of source

Line number of source is emitted.

Symbols for inline expansion

Symbols for inline expansion are emitted at the line including function call.

Symbol	Description
i	Indicates inline expansion is performed
Blank	Indicates inline expansion is not performed

The number of loop unrolling

If loop unrolling is performed, multiplicity of loop unrolling is emitted.

If the loop full unrolled, "f" is emitted instead of multiplicity of loop unrolling.

Otherwise, blank is emitted.

Symbols for using SIMD Extensions

Symbols for the optimization that uses SIMD Extensions are emitted at the line including loop control statements.

The loop control statement is "for" statement, "while" statement, "do-while" statement, and "if-goto" statement.

Symbol	Description
v	Indicates all statements in this loop have been optimized using SIMD Extensions.
m	Indicates that only some of the statements in this loop have been optimized using SIMD Extensions.
s	Indicates none of the statements in this loop have been optimized using SIMD Extensions.
Blank	Indicates none of the statements in this loop are targeted for optimization using SIMD Extensions.

Details Optimization Information

When the -ffj-lst=t option is specified, the source list containing details of optimization and parallelization information which has been performed are written to file(s).

The following optimization information are put just before each loop.

Optimization Information In Loop Units

- FISSION(num: *N*)

It means that loop was split.

- *N* is the number of loops after fission.

- SOFTWARE PIPELINING

It means that loop was pipelined.

- SIMD

It means that SIMD instructions were generated for the loop. Displayed as one of the followings:

- SIMD(VL: *length*[,*length*]... Interleave: *num*[,*num*])

A SIMD instruction treats *length* elements of array. When loop fission is applied to the loop and the values of *length* for the each loop are different, two or more lengths are displayed.

num displays the number of instructions expanded in the loop after SIMD to make memory access after SIMD more efficient.

- SIMD(VL: AGNOSTIC; VL: *length*[,*length*]... in 128-bit Interleave: *num*[,*num*])

It means that the vector register of SVE is SIMD without regard to a specific size. The *length* represents the number of elements of the array processed in 1 SIMD instruction, assuming that the vector register size of the SVE is 128 bits. If the *length* is different for each loop that is loop divided, multiple *lengths* are displayed.

num displays the number of instructions expanded in the loop after SIMD to make memory access after SIMD more efficient.

- PATTERN MATCHING(matmul)

It means that loop was changed to library function call (matmul).

- FULL UNROLLING

It means that loop was fully unrolled.

- CLONE

It means that loop was loop cloned.

Information That Relates To The Prefetch

Prefetch instruction

- PREFETCH(SOFT) : *N*

It indicates the number of all prefetch instructions that exist in the loop.

In addition, the number and the array name of the prefetch instruction of each access type of the prefetch are shown in the form of the following.

access type: <i>N</i> array name: <i>N</i> , ...

access type

- SEQUENTIAL : *N*

It indicates the number of prefetch instructions for array data accessed sequentially within a loop.

- STRIDE : *N*

It indicates the number of prefetch instructions for array data that is accessed with a stride larger than the cache line size used in the loop.

array name

- array name: *N*,...

It indicates the number of prefetch instructions for the array name. The array which was generated by compiler is shown as "(unknown)".

Information That Relates To The Register

- SPILLS :

It indicates the number of register save/restore instructions in the innermost loop for each register type.

- GENERAL : SPILL *N* FILL *N*

It indicates the number of save/restore instructions to the memory of the general-purpose register.

- SIMD&FP : SPILL *N* FILL *N*

It indicates the number of instructions for saving/restoring SIMD and floating-point registers to memory.

- SCALABLE : SPILL *N* FILL *N*

It indicates the number of instructions for saving/restoring extended register to memory.

- PREDICATE : SPILL *N* FILL *N*

It indicates the number of instructions for saving/restoring the predicate register to memory.

9.4.1.2.3 Example of Source Listing

Example of source listing is shown in following list.



Example

Example 1: Source listing

```
Compilation information
Current directory : directory-name
Source file : source-file-name
(line-no.)(optimize)
1          #include <stdio.h>
2
3          float sub(int i);
4
5          int main() {
6              int i;
7              float a[1000];
8
9              v   for(i = 0; i < 1000; i++) {
10                 a[i] = i;
11             }
12
13             printf("%lf\n", a[0]);
14             i = 0;
15             _loop:
16                 if(i < 1000) {
17                     goto _next;
18                 }
19                 a[i] = i;
20                 i++;
21                 goto _loop;
22             _next:
23
24             printf("%lf\n", a[0]);
25
26             v   for(i = 0; i < 1000; i++) {
27 i             a[i] = sub(i);
28             }
29             f   for(i = 0; i < 2; i++) {
30                 a[i] = i;
31             }
32             printf("%lf\n", a[0]);
33             return 0;
34         }
35
36         float sub(int j) {
```

```

37         return (float)j;
38     }

```

In this sample, following information can be read.

- The statements included in the "for"-statement at line-9 are optimized using SIMD Extensions.
- The statements included in the "for"-statement at line-26 are optimized using SIMD Extensions.
- The function called at line-27 is optimized with inline expansion.
- The statements included in the "for"-statement at line-29 are full unrolled.



Example

Example 2: Source listing

```

Compilation information
Current directory : directory-name
Source file : source-file-name
(line-no.)(optimize)
1         #include <stdio.h>
2
3         float sub(int i);
4
5         int main() {
6             int i;
7             float a[10000];
8
9             <<< Loop-information Start >>>
10            <<< [OPTIMIZATION]
11            <<<     SIMD(VL: 4 Interleave: 1)
12            <<< Loop-information End >>>
13            v   for(i = 0; i < 10000; i++) {
14                a[i] = i;
15            }
16            printf("%lf\n", a[0]);
17
18            i = 0;
19            _loop:
20            if(i < 10000) {
21                goto _next;
22            }
23            a[i] = i;
24            i++;
25            goto _loop;
26            _next:
27            printf("%lf\n", a[0]);
28            <<< Loop-information Start >>>
29            <<< [OPTIMIZATION]
30            <<<     SIMD(VL: 4 Interleave: 1)
31            <<< Loop-information End >>>
32            v   for(i = 0; i < 10000; i++) {
33                i
34                a[i] = sub(i);
35            }
36            <<< Loop-information Start >>>
37            <<< [OPTIMIZATION]
38            <<<     FULL UNROLLING
39            <<< Loop-information End >>>
40            f   for(i = 0; i < 8; i++) {
41                a[i] = i;
42            }

```

```

31          printf("%lf\n", a[0]);
32          return 0;
33      }
34
35      float sub(int j) {
36          return(float)j;
37      }

```

In this sample, following information can be read.

- The statements included in the "for"-statement at line-9 are optimized using SIMD Extensions.
- The loop by "for"-statement at line-25 is optimized using SIMD Extensions.
- The function called at line-26 is optimized with inline expansion.
- The statements included in the "for"-statement at line-28 are full unrolled.

9.4.1.2.4 Notes on Information at Compilation (Source List)

The -ffj-lst=p option, -ffj-lst=t option and -ffj-src option could output wrong compilation information (optimization information) in the following cases.

- When a function is inline expanded, different optimizations could be applied to each line. In this case, the compiler could output information incorrectly as follows.
 - Output messages redundantly.
 - Output optimization messages inconsistent with compilation information (optimization information).
 - Output no compilation information (optimization information).

When a function in a loop is expanded inline, the prefetch number of the function is recorded in the compilation time information (optimization information) of the loop.

- The compiler applies multiple optimizations such as SIMD Extensions, loop fusion, and loop distribution to one loop. In this case, the compiler could output information incorrectly as follows.
 - Output optimization information on line number to an incorrect line.
 - Output compilation information (optimization information) inconsistent with optimization messages.
- Loop unswitching optimization creates a loop in each "TRUE" and "FALSE" blocks of the "if" statement, but the compiler outputs optimization message for only one of the loops. In addition, optimization information on line number may not be output.
- When multiple loops are written in the same line in a program, the compiler outputs optimization information for only one of the loops. Please make sure to write one loop in one line in order to output optimization information.
- Detailed optimization information for a loop which consists of "if-goto" statements is not output. Optimization information on line number could be output to an incorrect line.
- When link time optimization is applied, the following phenomena may occur.
 - The optimization which is different from the optimization displayed in compilation information may be applied at runtime.
 - Some compilation information (Optimization Information) will not be output.
- Since the interleave function uses a part of the SIMD function, the SIMD Extensions symbol "v" is displayed even if only the interleave function is operated without SIMD Extensions. In this case, "VL: *length*" is not displayed as detailed optimization information, so it can be judged that SIMD Extensions are not operated.



Example

- When the SIMD Extensions and interleave function are operated

```

<<< Loop-information Start >>>
<<< [OPTIMIZATION]

```

```

          <<<   SIMD(VL: 4 Interleave: 1)
          <<< Loop-information End >>>
9          v   for(i = 0; i < 10000; i++) {
10         a[i] = i;
11         }

```

- When the interleave function is operated without SIMD Extensions

```

          <<< Loop-information Start >>>
          <<< [OPTIMIZATION]
          <<<   SIMD(Interleave: 1)
          <<< Loop-information End >>>
9          v   for(i = 0; i < 10000; i++) {
10         a[i] = i;
11         }

```

9.5 Language Specifications

9.5.1 Supported Language Standard

This section describes support status of the language standard.

Clang Mode supports the following language standard.

- C89
- C99
- C11

9.5.2 Half-Precision (16 bit) Floating-Point Type

The following two types of half precision (16bit) floating-point type are supported.

__fp16 data type: Defined in Arm C language extension (IEEE 754-2008)

__fp16 data type is not an arithmetic data type. __fp16 data type is used for storing a value and converting a type. Therefore, when a value declared with __fp16 data type is used at an arithmetic operation, it is converted to single-precision floating-point data type automatically. And, after the arithmetic operation, single-precision floating-point data type is converted to __fp16 data type.

_Float16 data type: Defined in extension of C11 (ISO/IEC TS 18661-3:2015)

_Float16 data type is an arithmetic data type. When an operation is performed by using a value declared with _Float16 data type, a half-precision arithmetic operation is used. Using _Float16 data type may increase the execution performance because a type conversion to single-precision floating-point data type is unnecessary.

However, there is no implicit data conversion between _Float16 data type and single-precision floating-point data type. Therefore, an explicit type conversion is necessary when a value declared with _Float16 data type is passed as an argument with single-precision floating-point data type.



Example

Explicit type conversion between _Float16 and single-precision floating-point data type

```

void half_function(void)
{
    _Float16 value = 1.0f16;
    printf("%f", (float)value);
}

```

9.6 Notes on Linking with Different Languages and Trad/Clang Modes

This section describes notes about linking with object programs in different programming languages and Trad/Clang Modes.

9.6.1 Compile commands and required options when linking

Refer to ["7.1 Compile Commands and Required Options when Linking"](#).

9.6.2 Linking with C++

Note the following:

- To call a C++ function from a C function, the C++ function must be declared with linkage to C specified (via `extern "C"`) in the C++ program.
- To call a C function from a C++ function, the C function must be declared with linkage to C specified (via `extern "C"`) in the C++ program.
- The user can specify any type permitted under C for the arguments and return values of a function to be called in a C specified function specified with `extern "C"`.

Example 1: Passing control to the C++ program first

C++ source: `cplusmain.cc`

```
#include <iostream>

extern "C" {
    void cfunc(short int);
    int cpfunc(int);
}

int cpfunc(int i) {
    std::cout << "C++: cpfunc called, i=" << i << std::endl;
    return i;
}

int main() {
    std::cout << "C++ main()" << std::endl;
    cfunc(10);
    return 0;
}
```

C source: `csub.c`

```
#include <stdio.h>

int cpfunc(int);

void cfunc(short int si) {
    printf("C:  cfunc called, si=%d\n", si);
    cpfunc(si);
}
```

Compiling and linking:

```
$ fccpx -Nclang -c csub.c
$ FCCpx -Nclang csub.o cplusmain.cc
```

Execution:

```
$ ./a.out
```

Result:

```
C++ main()  
C:   cfunc called, si=10  
C++: cpfunc called, i=10
```

Example 2: Passing control to the C program first

C++ source: cplussub.cc

```
#include <iostream>  
  
extern "C" {  
    void cfunc(short int);  
    int cpfunc(int);  
}  
  
int cpfunc(int i) {  
    std::cout << "C++: cpfunc called, i=" << i << std::endl;  
    cfunc(i);  
    return i;  
}
```

C source: cmain.c

```
#include <stdio.h>  
  
int cpfunc(int);  
  
int main() {  
    printf("C main()\n");  
    cpfunc(10);  
    return 0;  
}  
  
void cfunc(short int si) {  
    printf("C:   cfunc called, si=%d\n", si);  
}
```

Compiling and linking:

```
$ fccpx -Nclang -c cmain.c  
$ FCCpx -Nclang cmain.o cplussub.cc
```

Execution:

```
$ ./a.out
```

Result:

```
C main()  
C++: cpfunc called, i=10  
C:   cfunc called, si=10
```

9.6.3 Linking with Fortran

Note the following:

- If the program to which control is first passed is C, the function main must be "MAIN__" or "main". For more details, refer to "Fortran User's Guide".
- If the program to which control is first passed is Fortran, the return value of Fortran is set the return value of executable program. For more details, refer to "Fortran User's Guide".
- Append "_" to the end of the C function called from the Fortran and the Fortran function name called from C.

- Other rules are the same as those for interlanguage linkage between C and Fortran. Please refer to "Fortran User's Guide".

When using the COARRAY specification, specify the `--linkcoarray` option instead of the `--linkfortran` option. For more information on the COARRAY specification, see "Fortran User's Guide Additional Volume COARRAY".



Example

Example 1: Passing control to the Fortran program first

Fortran source: `fortranmain.f95`

```
print *, "Fortran: main program"
call cfunc(10)
end
```

C source: `csub.c`

```
#include <stdio.h>
int cfunc_(int *p) {
    printf(" C: cfunc_ called, *p=%d\n", *p);
    return *p;
}
```

Compiling and linking

- When using the compile command for the Fortran language at linking

```
$ fccpx -Nclang -c csub.c
$ frtpx csub.o fortranmain.f95
```

- When using the compile command for the C language at linking

```
$ frtpx -c fortranmain.f95
$ fccpx -Nclang --linkfortran csub.c fortranmain.o
```

Execution

```
$ ./a.out
```

Result

```
Fortran: main program
C: cfunc_ called, *p=10
```

Example 2: Passing control to the C program first (MAIN__)

Fortran source: `fortransub.f95`

```
integer function func(x)
integer*4 x
print *, "Fortran: func() called"
call cfunc(x)
func=x
end
```

C source: `cmain.c`

```
#include <stdio.h>
int func_(int *);
int cfunc_(int *p) {
    printf(" C: cfunc_ called. *p=%d\n", *p);
    return *p;
}
int MAIN__() {
```

```

    int i=10;
    printf(" C MAIN__()\n");
    func_(&i);
    return 0;
}

```

Compiling and linking

- When using the compile command for the Fortran language at linking

```

$ fccpx -Nclang -c cmain.c
$ frtpx cmain.o fortransub.f95

```

- When using the compile command for the C language at linking

```

$ frtpx -c fortransub.f95
$ fccpx -Nclang --linkfortran cmain.c fortransub.o

```

Execution

```

$ ./a.out

```

Result

```

C MAIN__()
Fortran: func() called
C: cfunc_ called. *p=10

```

Example 3: Passing control to the C program first (main)

Fortran source: fortransub.f95

```

integer function func(x)
integer*4 x
print *, "Fortran: func() called"
call cfunc(x)
func=x
end

```

C SOURCE: cmain.c

```

#include <stdio.h>
int func_(int *);
int cfunc_(int *p) {
    printf(" C: cfunc_ called. *p=%d\n", *p);
    return *p;
}
int main() {
    int i=10;
    printf(" C main()\n");
    func_(&i);
    return 0;
}

```

Compiling and linking

- When using the compile command for the Fortran language at linking

```

$ fccpx -Nclang -c cmain.c
$ frtpx -mlcmain=main cmain.o fortransub.f95

```


- When using the compile command for the C language at linking

```
$ frtprx -c fortransub.f95
$ fccprx -Nclang --linkfortran cmain.c fortransub.o
```

Execution

```
$ ./a.out
```

Result

```
C main()
Fortran: func() called
C: cfunc_ called. *p=10
```

9.7 SIMD Built-in Functions

Clang Mode supports SIMD (Single Instruction Multi Data) built-in functions.

For details of SIMD built-in functions, refer to the document "ARM C Language Extensions for SVE" published by Arm for developers.



Note

The Clang Mode supports the SIMD built-in functions documented in the document version 00bet1 (First public release). Note that the SIMD built-in functions added in document version 00bet2 and later are not supported. For more information, see "ARM C Language Extensions for SVE - 1.1.2. Change history".

9.8 Compatibility with GNU C Specifications

This chapter describes the GNU C language specifications (GNU C Extensions) and compile options (GNU C compatible options) in Clang Mode.

9.8.1 GNU C Extensions

For the GNU C extensions in Clang Mode, refer to the clang and GCC website.

9.8.2 GNU C Compatible Options

Clang Mode supports the following compatible options (GNU C compatible options). For details of GNU C compatible options, refer to the GCC website.

`{--print-file-name|-print-file-name}=include`

This option specifies to print the include directory.

`{--print-prog-name|-print-prog-name}={as|ld|objdump|ranlib|ar}`

This option specifies to print the name of programs which are called by compile command.

`--shared`

This option makes the linker create shared objects and not dynamic link executable files.

This option is passed to the linker.

`--version`

This option specifies to emit the version and copyright information of the compiler to the standard output.

`-Wp,-MD,filename`

This option is equivalent to the following options:

`-MD -MF filename`

-Xlinker *option*

This option directs that the *option* is to be passed as arguments to the linker.

-dM

This option specifies to output all macro definitions when the -E option is effective.

-f{exceptions|no-exceptions}

The -fexceptions option defines the __EXCEPTIONS macro. The -fno-exceptions option invalidates the -fexceptions option. -fno-exceptions is set by default.

-fno-common

This option specifies that the global variable without the initial value is allocated to the data section in the object file.

This option is useful for variables declared with the same name without the "extern" specifier in two or more sources, as this can be detected as an error at linking time.

-f{optimize-sibling-calls|no-optimize-sibling-calls}

The compiler option -foptimize-sibling-calls specifies to perform optimization of the sibling call. The compiler option -fno-optimize-sibling-calls invalidates the compiler option -foptimize-sibling-calls option. The compiler option -fno-optimize-sibling-calls is set by default. When the compiler option -O1 or higher is set, the compiler option -foptimize-sibling-calls is set by default.

The compiler options -foptimize-sibling-calls and -fno-optimize-sibling-calls require that the compiler option -O1 or higher is set.

-f{pie|PIE}

This option specifies to generate position-independent code.

If the -fPIE option is specified, a slower object is generated with a long instruction sequence. However a number of unique global symbols are able to be used in the final shared library that the object will be a part of. If the -fpie option is specified, a faster object is generated with a short instruction sequence. However few of unique global symbols are able to be used in the final shared library that the object will be a part of. The number of the unique external symbols that can be referred for these cases is the total about all libraries united at the same time.

If -fPIE and -fpie options are specified, only the later one is effective.

This option is effective at compilation time only.

The difference between these options and the -f{pic|PIC} options is that the objects can be linked with the -pie linker option to generate a position independent executable.

-fvisibility={default|internal|hidden|protected}

This option specifies the visibility attribute of global symbols in components (executable or shared libraries). -fvisibility=default is set by default.

-fvisibility=default

The symbol is able to be referred from other components.

-fvisibility=hidden

The symbol is not able to be referred from other components. However, it is able to be referred from the other components if the symbol address can be handled in the other components.

-fvisibility=internal

This option is equivalent to the -fvisibility=hidden option.

-fvisibility=protected

The symbol is able to be referred from other components. However, the symbol is not overridden by other symbol with the same name in other components.

-g{dwarf|dwarf-4}

This option specifies to add DWARF4 debugging information to the object file.

This option is equivalent to the -g option.

-idirafter *dir*

This option adds the *dir* directory to the end of the standard search paths to search the headers.

If multiple directories are specified in multiple -idirafter options, the directories are searched in the specified order.

For details of the search order of the headers, refer to the description of -I option in Section "[9.1.2.2.1 General Options for Compiler](#)".

If a header is specified with an absolute path name, only the specified absolute path name is searched. If the specified directory does not exist, this option is invalidated.

-include *file*

This option specifies to include the *file* at the top of the source file.

If multiple files are specified in multiple -include options, the files are included in the specified order.

-isystem *dir*

This option adds the *dir* directory to the search paths to search the standard headers.

If multiple directories are specified in multiple -isystem options, the directories are searched in the specified order.

If a header is specified with an absolute path name, only the specified absolute path name is searched. If the specified directory does not exist, this option is invalidated.

-nostartfiles

This option specifies not to use standard system startup files at linking.

-nostdinc

This option specifies not to use the standard directory to search headers.

-nostdlib

This option specifies not to use standard system startup files and standard libraries at linking.

-pthread

This option specifies to create thread safe objects by using the POSIX thread library.

-rdynamic

This option specifies to the linker to add all symbols to the dynamic symbol table. This option passes the -export-dynamic option to the linker.

-undef

This option specifies to suppress the system specific and GNU C specific predefined macros.

-w

This option suppresses the output of warning messages.

-x *language*

This option specifies the type of input files specified after this option. "c" or "assembler-with-cpp" can be specified as *language*. If "c" is specified, it is treated as a C source file. If "assembler-with-cpp" is specified, it is treated as an assembler source file that requires preprocessing.

This option cannot be specified simultaneously with the options -ffj-lst[={p|t}], -ffj-lst-out=*file*, and -ffj-src.

9.9 Code Coverage

This section explains the code coverage.

The code coverage is the function to measure the execution frequency of the executable statement at execution, and to examine the code coverage rate of the program.

The code coverage uses the coverage tool based on LLVM compiler infrastructure (hereafter called llvm-cov), which is an open source software.

Refer to an online manual of the llvm-cov command by the man command for the use of the llvm-cov command.

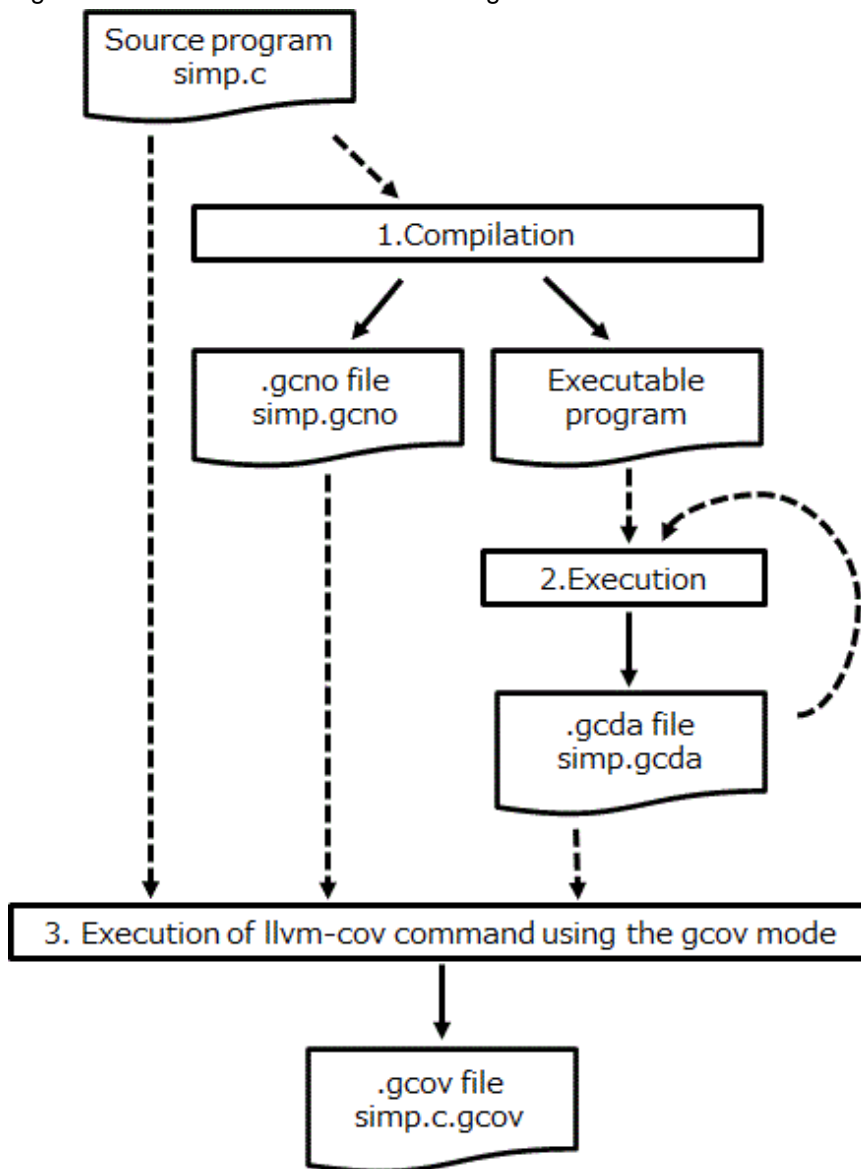
9.9.1 How to Use Code Coverage

The code coverage is used the following procedures.

1. Compilation
2. Execution
3. Execution of llvm-cov command using the gcov mode

A following figure shows the procedure of the code coverage.

Figure 9.1 Procedure of the code coverage



9.9.2 Necessary Files to Use Code Coverage

This section explains the necessary files to use the code coverage.

9.9.2.1 The ".gcno" File

The .gcno file is a binary file which contains the information obtained at compilation.

File generation step	The file is generated at compilation
----------------------	--------------------------------------

File name	The file name is the same as the source program with file extension .gcno.
File generated location	The .gcno file is generated in the directory where the compilation is performed.

An example of generating .gcno file is shown as follows.



Example

```
$ gcc -Nclang --coverage ../simp.c
$ ls
a.out simp.gcno
```

When the assembler source file and the object file are generated with specifying the -o option, the -o option is also applied to the .gcno file.

An example of generating files with specifying the -o option is shown as follows.



Example

```
$ gcc -Nclang --coverage ../simp.c -c -o www/yyy.o
$ ls www/
yyy.gcno yyy.o
```

9.9.2.2 The ".gcda" File

The .gcda file is a binary file which contains the information obtained at execution.

File generation step	The file is generated at execution. The execution frequency is accumulated when the .gcda file with the same name already exists.
File name	The file name is the same as the source program with file extension .gcda.
File generated location	The .gcda file is generated in the directory where the compilation is performed.

An example of generating .gcda file is shown as follows.



Example

```
$ gcc -Nclang --coverage ../simp.c
$ ls
a.out simp.gcno
$ ./a.out
$ ls
a.out simp.gcda simp.gcno
```

When the assembler source file and the object file are generated with specifying the -o option, the -o option is also applied to the .gcda file.

An example of generating files with specifying the -o option is shown as follows.



Example

```
$ gcc -Nclang --coverage ../simp.c -c -o www/yyy.o
$ ls www/
yyy.gcno yyy.o
$ gcc -Nclang --coverage www/yyy.o
$ ./a.out
```

```
$ ls www/  
yyy.gcda yyy.gcno yyy.o
```

The storage location directory of the .gcda file can be changed by specifying the `-fprofile-dir=dir_name` option.

An example of generating files with specifying the `-fprofile-dir=dir_name` option is shown as follows.



Example

```
$ gcc -Nclang --coverage ../simp.c -c -o www/yyy.o -fprofile-dir=/tmp  
$ ls www/  
yyy.gcno yyy.o  
$ gcc -Nclang --coverage www/yyy.o  
$ ./a.out  
$ ls www/  
yyy.gcno yyy.o  
$ ls /tmp/www/  
yyy.gcda
```

9.9.3 Notes on Code Coverage

This section explains notes of the code coverage.

- The execution performance may decrease because the instructions that measure the execution frequency are added in the object program.
- The storage directory of the .gcda file, which is generated at execution, is determined at compilation.
 - The location of the .gcda file storage directory does not change even if the executable program is moved after compilation.
 - Use the `-fprofile-dir=dir_name` option at compilation to change the location of the .gcda file storage directory.
- The code coverage function is not available for programs that need the `-mmodel=large` option.
- The execution frequency may be measured incorrectly in the following cases:
 - Two or more executable statements are included in one line.
 - The `exit(3)` function is called.
 - The `setjmp` or `longjmp` function is called.
 - An exception is caught.
 - The `#line` directive is included in the source program.
 - Optimizations of the compiler are applied.

9.10 Restrictions and Notes

This section describes the restrictions and notes of Clang Mode.

9.10.1 Restrictions of Macros `CMPLX`, `CMPLXF`, and `CMPLXL` in `complex.h`

The macros `CMPLX`, `CMPLXF`, and `CMPLXL` in `complex.h` are not defined.

If a complex type variable is initialized using the macros `CMPLX`, `CMPLXF`, or `CMPLXL` in `complex.h`, the following occurs.

- C program : compilation error or link error
- C++ program : compilation error

Please initialize complex type variables without using the macros `CMPLX`, `CMPLXF`, and `CMPLXL` in `complex.h`.



Example

Initialization without using macro CMPLX

C program

```
double complex a = 1.0 + 2.0 * I;
```

C++ program

```
std::complex<double> a(1.0, 2.0);
```

Appendix A Implementation-dependent Specifications

This appendix describes operations related to correct program structural elements and data that are specific to the characteristics of this system.

Details on how to process and use implementation-dependent items are provided below.

A.1 Compiling

Diagnostic messages are written to the standard error.

The format of the diagnostic messages is as follows:

```
"filename", line nn: message-text
```

filename: Name of the file in which the error occurred.

nn: Number of the line where the error occurred.

A.2 Environment

The real arguments to the main function are as follows:

```
int main(int argc, char *argv[]) { /*...*/ }
```

argc: Number of real arguments

argv: Array of pointers to the strings that become real arguments
(*argv*[0] is the program name or null string)

An interactive device consists of

consoles, terminals and other display devices.

A.3 Identifiers

The number of significant initial characters (31 or more) in an identifier without external linkage.

Up to 4096 characters. Identifiers that are external links are case-sensitive.

The number of significant letters (6 or more) in an identifier that is an external link.

Up to 4096 characters.

Which additional multibyte characters may appear in identifiers and their correspondence to universal character names.

An identifier cannot contain multibyte characters.

A.4 Characters

The members of the source and execution character sets, except as explicitly specified in the standard.

Other than the values of elements of the source code character sets and executed character sets are the values of the environment variable, which is one of ja_JP-EUC, ja_JP.UTF-8.

The shift states used for the encoding of multibyte characters.

The encoding of multibyte characters is not shift state-dependent encoding.

The number of bits in a character in the execution character set.

There are 8 bits per byte.

The mapping of members of the source character set (in character and string literals) to members of the execution character set.

The mapping is identical between source and execution characters.

The value of an integer character constant that contains a character or escape sequence not represented in the basic execution character set or the extended character set for a wide character constant.

The values of character codes corresponding to the Character Codes General Description are used.

In the event that undefined extended notation appears in a character constant or string literal, only \ is ignored and the remainder is handled simply as characters.

The value of an integer character constant that contains more than one character or a wide character constant that contains more than one multibyte character.

If two or more characters are coded as a single character constant, four characters are extracted starting from the left, and the constant takes the following value when the characters are 'C1C2...Cn' ($2 \leq n \leq 5$):

$$\sum_{k=1}^{n-1} 256^{k-1} \times ('Ck' \& 255) + 256^{n-1} \times 'Cn', (2 \leq n \leq 4)$$

If there are five or more characters, the fifth and subsequent characters are ignored. In the case of wide character constants, if more than one multibyte character is coded, one character is extracted starting from the left, and its value is changed to the value of the corresponding code in ASCII. If there are two or more characters, the second and subsequent characters are ignored.

See coding examples below:

Value of the expression '\x123 '	: '\x23 '
Value of the expression '\0223 '	: '\022' + '3' x 256
Value of the expression L'\1234 '	: L'\123 '

The current locale used to convert multibyte characters into corresponding wide characters (codes) for a wide character constant.

Specified by the environment variable LANG.

Whether a plain char has the same range of values as signed char or unsigned char.

It has the same range of values as unsigned char.

The values of the members of the execution character set.

The execution character set is same as source character set at compilation time.

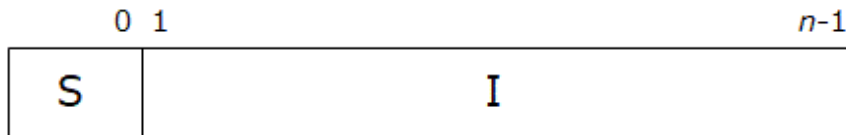
The value of a string literal containing a multibyte character or escape sequence not represented in the execution character set.

The value interpreted by the implementation is stored as itself.

A.5 Integers

The representations and sets of values of the various types of integers.

The representations and set of values of the ordinary integer types are given below.



S : Sign (positive if 0, negative of 1)

I : Integer part

$$x = -S \times 2^{n-1} + \sum_{k=1}^{n-1} i_k \times 2^{n-k-1}$$

If a character set-enumerated quantity other than any of the requested elements of the source code character set is stored in a char type object, the action performed is that the least significant byte of the quantity number is stored and the value is handled as a signed integer.

The result of converting an integer to a shorter signed integer, or the result of converting an unsigned integer to a signed integer of equal length, if the value cannot be represented.

The value becomes the bit pattern interpreted as a two's complement expression with the same width as the converted type.

The results of bitwise operations on signed integers.

The result of bitwise operation has the correct sign bit because of there is no padding bit.

The sign of the remainder on integer division.

If the result of an operation among integers is not divided evenly, if both operands are positive values, the result of the / operator is the largest integer smaller than the true value of the quotient, and the result of the % operator is positive.

If one of the operands is negative, the result of the / operator is the smallest integer greater than the true value of the quotient. The sign of the result of the % operator is the same as the sign of the dividend.

If both operands are negative values, the result of the / operator is the largest integer smaller than the true value of the quotient, and the sign of the result of the % operator is the same as the sign of the dividend.

The result of a right shift of a negative-valued signed integral type.

The result of $E1 \gg E2$ is $E1$ shifted to the right by $E2$ bits. If $E1$ is signed and has a negative value, the result is the value calculated by the following formula:

$$E1/2^{E2} - 1 + (E1 \% 2^{E2} == 0)$$

Any extended integer types that exist in the implementation.

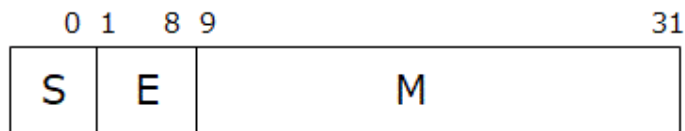
There is no extended integer type.

A.6 Floating-Point Numbers

The representations and sets of values of the various types of floating-point numbers.

The representations and set of values of the floating-point type are given below.

float type



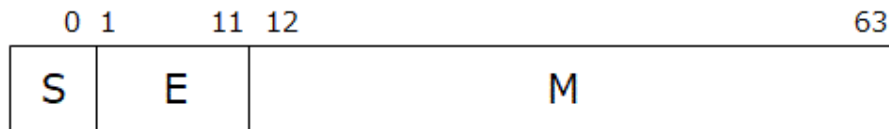
S : Sign (positive if 0, negative if 1)

E : Exponent part

M : Mantissa part

$$x = (-1)^S \times (1 + m/2^{23}) \times 2^{e-127}$$

double type



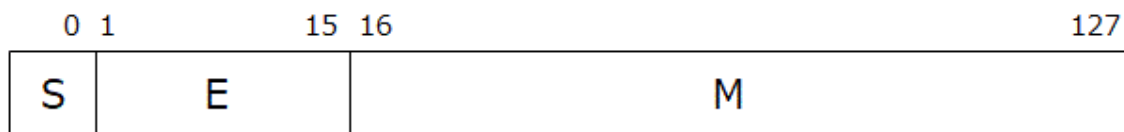
S : Sign (positive if 0, negative if 1)

E : Exponent part

M : Mantissa part

$$x = (-1)^S \times (1 + m/2^{52}) \times 2^{e-1023}$$

long double type



S : Sign (positive if 0, negative if 1)

E : Exponent part

M : Mantissa part

$$x = (-1)^S \times (1 + m/2^{112}) \times 2^{e-16383}$$

The direction of truncation when an integral number is converted to a floating-point number that cannot exactly represent the original value.

Rounding is performed according to the value of `FLT_ROUNDS` at that time. The initial value of `FLT_ROUNDS` is 1 (closest).

The direction of truncation or rounding when a floating-point number is converted to a narrower floating-point number.

Rounding is performed according to the value of `FLT_ROUNDS` at that time. The initial value of `FLT_ROUNDS` is 1 (closest).

A.7 Arrays and Pointers

The type of integer required to hold the maximum size of an array - that is, the type of the `sizeof` operator, `size_t`.

The type of `size_t` is handled as unsigned long int type.

The result of casting an integer to a pointer or vice versa.

A pointer can be converted into an integer type. The required size is of `unsigned int` type. The value of the result is the same value as when the internal expression of the `unsigned int` type matches the internal expression of the pointer (the bit pattern is not converted in the conversion).

Any integer can be converted to a pointer. The value of the result is the value of the destination type that has an internal expression matching the internal expression after conversion to an `unsigned int` type (the bit pattern is not converted in the conversion).

The type of integer required to hold the difference between two pointers to elements of the same array, `ptrdiff_t`.

The type of `ptrdiff_t` is long type.

A.8 Registers

The extent to which objects can actually be placed in registers by use of the `register` storage-class specifier.

This specification is effective when the type of the object to be stored in registers is arithmetic type (excluding `long double` type) and pointer. This is the case when, regardless of the specification of `register`, the registers are allocated to objects in order starting with objects that are most effective, according to the usage.

A.9 Structures, Unions, Enumerated Types, and Bit Fields

A member of a union object is accessed using a member of a different type.

The value is interpreted according to the type of the member that performs the access.

The padding and alignment of members of structures.

Structure members are padded internally so that every element is aligned on the appropriate boundary.

See "[Table A.1 data size and alignment](#)" for size and boundary alignment of arithmetic types. Units are bytes.

Table A.1 data size and alignment

data type	size	alignment
char	1	1
signed char	1	1
unsigned char	1	1
signed short int	2	2
unsigned short int	2	2
signed int	4	4
unsigned int	4	4
signed long int	8	8
unsigned long int	8	8
signed long long int	8	8
unsigned long long int	8	8
float	4	4
double	8	8
long double	16	16

Whether a "plain" int bit-field is treated as a `signed int` bit-field or as an `unsigned int` bit-field.

It is handled as an `unsigned int`.

The order of allocation of bit-fields within a unit.

The bits within a unit are allocated in order from the most-significant bit of the specified type.

Whether a bit-field can straddle a storage-unit boundary.

A bit field is allocated to the lowest-addressed memory unit of a size that is sufficient to hold it. If there is sufficient space remaining, a different bit field immediately after the bit field within a structure is packed into adjacent bits in the same unit. If there is not sufficient space, the bit field that does not fit is packed into the next unit so that it does not lie across the boundary with the adjacent unit.

The integer type chosen to represent the values of an enumeration type.

The type used to express the value of enumerated type data is `int` type.

Allowable bit-field types other than `_Bool`, `signed int`, and `unsigned int`.

All integer types can be used.

A.10 Qualifiers

What constitutes an access to an object that has volatile-qualified type.

The `volatile` declaration can be used to code objects corresponding to I/O ports allocated to memory and objects accessed using asynchronous interrupt functions. The operation of objects declared in this manner is not subject to "optimization" by the processing system, nor do they take modification of the order of evaluation outside the range permitted by the rules for evaluation of expressions.

A.11 Declarators

The maximum number of declarators that may modify an arithmetic, structure or union type.

Unlimited (limited only by the size of the memory area).

A.12 Statements

The maximum number of `case` values in a `switch` statement.

Unlimited (limited only by the size of the memory area).

A.13 Preprocessor Directives

Whether the value of a single-character character constant in a constant expression that controls conditional inclusion matches the value of the same character constant in the execution character set. Whether such a character constant may have a negative value.

It matches. In addition, a character constant consisting of a single character does not take a negative value in the basic character set. Other values may be negative.

The method for locating includable source files.

See the explanation of the `-I` option in Section "[2.2.2 Description of Compiler Options](#)".

The support of quoted names for includable source files.

See the explanation of the `-I` option in Section "[2.2.2 Description of Compiler Options](#)".

The mapping of source file character sequences.

The names are case-sensitive.

The behavior of each recognized `#pragma` directive.

See the explanation in Section "[6.1.2 The pragma Directive](#)".

The definitions for `__DATE__` and `__TIME__` when respectively, the date and time of translation are not available.

Same as that generated by the `asctime` function.

Whether the value of a single-character character constant in a constant expression that controls conditional inclusion may have a negative value.

It can have negative value.

Whether the # operator inserts a \ character before the \ character that begins a universal character name in a character constant or string literal.

operator adds \ character.

A.14 Standard Library Functions

For details on how to use the operation of the implementation-dependent specifications of standard library functions, refer to the reference manual provided with the system or use the man command.

A.15 OpenMP Specifications

See Section "[4.3.3 Implementation-Dependent Specifications](#)".

Appendix B Compilation Limitations

"Table B.1 Limit of Compilation" shows the upper limits when compilation using this system.

Table B.1 Limit of Compilation

Item	C Language Standard (each value or above)	Value of this System
Number of nesting levels in compound statements, iteration statements, and selection statements	127	Unlimited (depends on storage size)
Number of nesting levels of conditional inclusion	63	Unlimited (depends on storage size)
Number of (option combination of) pointers, arrays and function declarators that qualify the arithmetic type, structure type, union type or incomplete type in a declaration	12	Unlimited (depends on storage size)
Number of nesting levels of parenthesized declarators within a full declarator	63	Unlimited (depends on storage size)
Number of nesting levels of parenthesized expressions within a full expression	63	Unlimited (depends on storage size)
Number of significant initial characters in an identifier or a macro name	63	4096
Number of significant initial characters in an identifier	31	4096
Number of external identifiers in one translation unit	4095	Unlimited (depends on storage size)
Number of identifiers with scope declared in one block	511	Unlimited (depends on storage size)
Number of macro identifiers simultaneously defined in one preprocessing translation unit	4095	Unlimited (depends on storage size)
Number of parameters in one function definition	127	Unlimited (depends on storage size)
Number of arguments in one function call	127	Unlimited (depends on storage size)
Number of parameters in one macro definition	127	Unlimited (depends on storage size)
Number of arguments in one macro invocation	127	Unlimited (depends on storage size)
Number of characters in a logical source line	4095	Unlimited (depends on storage size)
Number of characters in a character string literal or wide string literal (after concatenation)	4095	Unlimited (depends on storage size)
Number of bytes in an object (in a hosted environment only)	65535	Unlimited (depends on storage size)
Number of nesting levels for #include files	15	511
Number of case labels for a switch statement (excluding those for any nested switch statements)	1023	Unlimited (depends on storage size)
Number of members in a single structure or union	1023	Unlimited (depends on storage size)
Number of enumeration constants in a single enumeration	1023	Unlimited (depends on storage size)

Item	C Language Standard (each value or above)	Value of this System
Number of levels of nested structure or union definitions in a single struct-declaration-list	63	Unlimited (depends on storage size)

Appendix C Restrictions and Notes

C.1 Restriction

None.

C.2 Notes

C.2.1 Variable Arguments of Function

In this system, the data with complex type, `struct` and `union` cannot be specified directly at the variable arguments of a function. If you want to specify the data of these types, please specify the pointer of the data in the arguments.

C.2.2 Expression Result of Signed Integer Type

Do not generate overflow when using a signed integer, as the expression result for this type will be undefined if the evaluation of the expression exceeds the range of its type.

C.2.3 Undefined Behavior

The "undefined behavior" in the C specification is not guaranteed.

C.2.4 Arrays of Variable Length

Programs created by this system allocate arrays of variable length in functions to the stack region.

When the arrays of variable length is huge or inline expansion of function calls is applied to the functions which have arrays of variable length, the stack area needs to be extended to the appropriate size.

The stack area of the process can be set by `ulimit` (bash built-in command), etc.

C.2.5 Restriction of Register Definition for Floating Point Types with `asm` Keyword

In this system, the data with a long double type or complex type cannot be defined in certain registers with `asm` keyword.

C.2.6 Restriction of Floating Point Types Expression as a Register Constraint for `asm` Operands

In this system, the data with a long double type or complex type cannot be specified as a register constraint for `asm` operands.

C.2.7 Debug Using Debugger

This section explains the notes when the executable file generated in Trad Mode is debugged by using the debugger.

Notes on using the debugger

In the following cases, debugging is restricted.

- Variables, functions, and types that are declared under the C11 specifications.
- Variables, functions, and types that are declared under the GNU C Extensions.

Debugging of optimized program

- Debugging of the SIMD type variables is restricted.

- Debugging of the parameters and the global variables is guaranteed at the entry of each function.
- Debugging of the local variables is not guaranteed.
- Debugging of the variables of which type is the following is restricted.
 - complex
 - long double
 - array
 - structure
 - union
- Debugging of the parameters which are not referred in its function is restricted.
- Debugging of the inlined functions is restricted.
- The displaying of line number is not guaranteed.

C.2.8 Changing SVE Vector Register Size

This system assumes that the SVE vector register size is not changed during executing a program.

If the effective size of the SVE vector register is changed during executing a program using the system call `prctl(2)` or in other ways, the behavior is undefined.

If you want to change the SVE vector register size, you should set it before executing a program that uses SVE.

C.2.9 Integer Division Exception when the Divisor Is Zero

Note that "Integer divide by zero" is not detected in the Fujitsu CPU A64FX with Arm architecture.

If the compiler option `-NRtrap` is specified and the divisor is 0, the program that expects "Integer divide by zero" to be detected may have different results.

It is recommended to check the value of the divisor just before the integer division.



Example

Example of checking the value of a divisor just before integer division

```
#include <stdio.h>
int main()
{
    int a,b,c;
    a = 1;
    b = 0;
    if (b == 0) {
        fprintf(stderr, "*** Integer divide by zero ***\n");
        exit(1);
    }
    return 0;
}
```

C.2.10 Link Error (undefined reference to)

When compiling a program, if the following operation is performed by configure etc., a link error (undefined reference to) may occur because the combining of this system original objects that normally performed at linking is suppressed.

- The `-L` option which is passed to the linker by this system is directly specified on the compile command to pass the linker by user.



Example

Example of a link error

```
test.o: In function `__fjc_check_hpctag':  
(.text+0x1620): undefined reference to `__jwe_check_hpctag'
```

This can be avoided by setting the environment variable FCOMP_LINK_FJOBJ. Refer to "[2.3 Environment Variable for Compile Command](#)" for the environment variable FCOMP_LINK_FJOBJ.

C.2.11 Restriction of registers with asm keyword and on an inline asm clobber lists

In this system, the following registers reserved by the compiler cannot be specified in the register name of asm keyword and on an inline asm clobber lists.

Compiler reserved registers: x19, w19, r19, x29, w29, r29, sp, xzr, xsp, wzr, wsp

Appendix D Compatibility with GNU C Specifications

This system partially supports the GNU C Compiler specification (version 6.1).

This chapter describes the GNU C language specifications (GNU C Extensions) and compile options (GNU C compatible options).

D.1 GNU C Extensions

The GNU C Extensions supported by this system are as follows. For more detail about GNU C Extensions, refer to the GCC website.

- Statements and Declarations in Expressions
- Locally Declared Labels
- Labels as Values
- Referring to a Type with `typeof`
- Conditionals with Omitted Operands
- Double-Word Integers
- Complex Numbers
Complex integral types are not supported.
- Hex Floats
- Arrays of Length Zero
- Arrays of Variable Length
- Macros with a Variable Number of Arguments
- Slightly Looser Rules for Escaped Newlines
- Non-Lvalue Arrays May Have Subscripts
- Arithmetic on void- and Function-Pointers
- Non-Constant Initializers
- Compound Literals
- Designated Initializers
- Case Ranges
- Cast to a Union Type
- Declaring Attributes of Functions/Specifying Attributes of Variables/Specifying Attributes of Types
For the list of attributes that can be used, see "[D.1.1 Attributes](#)".
- The Character ESC in Constants
- Inquiring on Alignment of Types or Variables
- Alternate Keywords
- Incomplete enum Types
- Function Names as Strings
- Builtins
- Unnamed struct/union fields within structs/unions
- Assembler Instructions with C Expression Operands
- Structures With No Members
- Mixed Declarations and Code

- C++ Style Comments
- Dollar Signs in Identifier Names
- An Inline Function is As Fast As a Macro
- Pragas Accepted by GCC
 - `#pragma redefine_extname`
 - `#pragma pack`
 - `#pragma weak`
- Thread-Local Storage

D.1.1 Attributes

The attributes supported by this system are as follows. For more detail about attributes, refer to the GCC website.

For the attributes supported by Clang Mode, see "<https://releases.llvm.org/7.0.0/tools/clang/docs/AttributeReference.html>".

- `aarch64_vector_pcs` (available for Functions)
- `alias`
- `aligned` (available for Variables, Types)
- `always_inline`
- `const`
- `constructor` (available for Functions)
- `destructor` (available for Functions)
- `deprecated` (available for Functions, Types)
- `format`
- `format_arg` (available for Functions)
- `malloc`
- `may_alias` (available for Types)
- `mode`
- `no_instrument_function` (available for Functions)
- `noinline`
- `nonnull`
- `nopl` (available for Functions)
- `noreturn`
- `nothrow`
- `packed` (available for Variables, Types)
- `pure`
- `section` (available for Functions, Variables)
- `sentinel` (available for Functions)
- `unused` (available for Functions, Variables, Types)
- `used` (available for Functions, Variables)

- transparent_union
- visibility (available for Functions, Types)
- warn_unused_result
- weak (available for Functions, Variables)
- weakref (available for Functions)

D.1.2 Built-in Functions

The Built-in Functions supported by this system are as follows. For more detail about built-in Functions, refer to the GCC website.

- __atomic_load_n
- __atomic_load
- __atomic_store_n
- __atomic_store
- __atomic_exchange_n
- __atomic_exchange
- __atomic_compare_exchange_n
- __atomic_compare_exchange
- __atomic_add_fetch
- __atomic_sub_fetch
- __atomic_and_fetch
- __atomic_xor_fetch
- __atomic_or_fetch
- __atomic_nand_fetch
- __atomic_fetch_add
- __atomic_fetch_sub
- __atomic_fetch_and
- __atomic_fetch_xor
- __atomic_fetch_or
- __atomic_fetch_nand
- __atomic_test_and_set
- __atomic_clear
- __atomic_thread_fence
- __atomic_signal_fence
- __atomic_always_lock_free
- __atomic_is_lock_free
- __builtin_abort
- __builtin_alloca
- __builtin_bswap16

- __builtin_bswap32
- __builtin_bswap64
- __builtin_calloc
- __builtin_choose_expr
- __builtin_clz
- __builtin_clzl
- __builtin_clzll
- __builtin_constant_p
- __builtin_ctz
- __builtin_ctzl
- __builtin_ctzll
- __builtin_exit
- __builtin_expect
- __builtin_extract_return_addr
- __builtin_ffs
- __builtin_ffsl
- __builtin_ffsll
- __builtin_fpclassify
- __builtin_fprintf
- __builtin_fputc
- __builtin_fputs
- __builtin_fscanf
- __builtin_fwrite
- __builtin_huge_val
- __builtin_huge_valf
- __builtin_huge_vall
- __builtin_index
- __builtin_inf
- __builtin_inff
- __builtin_infl
- __builtin_isfinite
- __builtin_isinf
- __builtin_isinf_sign
- __builtin_isinff
- __builtin_isinfl
- __builtin_isnan
- __builtin_isnanf

- __builtin_isnanl
- __builtin_isnormal
- __builtin_malloc
- __builtin_memchr
- __builtin_memcmp
- __builtin_memcpy
- __builtin_memmove
- __builtin_mempcpy
- __builtin_memset
- __builtin_nan
- __builtin_nanf
- __builtin_nanl
- __builtin_nans
- __builtin_nansf
- __builtin_nansl
- __builtin_offsetof
- __builtin_popcount
- __builtin_popcountl
- __builtin_popcountll
- __builtin_prefetch
- __builtin_printf
- __builtin_putchar
- __builtin_puts
- __builtin_return_address
- __builtin_rindex
- __builtin_scanf
- __builtin_signbit
- __builtin_signbitf
- __builtin_signbitl
- __builtin_sprintf
- __builtin_sscanf
- __builtin_stpcpy
- __builtin_strcat
- __builtin_strchr
- __builtin_strcmp
- __builtin_strcpy
- __builtin_strcspn

- __builtin_strlen
- __builtin_strncat
- __builtin_strncmp
- __builtin_strncpy
- __builtin_strpbrk
- __builtin_strrchr
- __builtin_strspn
- __builtin_strstr
- __builtin_trap
- __builtin_types_compatible_p
- __builtin_vfprintf
- __builtin_vprintf
- __builtin_vsprintf
- __sync_fetch_and_add
- __sync_fetch_and_sub
- __sync_fetch_and_or
- __sync_fetch_and_and
- __sync_fetch_and_xor
- __sync_fetch_and_nand
- __sync_add_and_fetch
- __sync_sub_and_fetch
- __sync_or_and_fetch
- __sync_and_and_fetch
- __sync_xor_and_fetch
- __sync_nand_and_fetch
- __sync_bool_compare_and_swap
- __sync_val_compare_and_swap
- __sync_synchronize
- __sync_lock_test_and_set
- __sync_lock_release

D.2 GNU C Compatible Options

This system partially supports the compatible options (GNU C compatible options).

For more detail about these GNU C compatible options, refer to the GCC website.

`{--print-file-name|-print-file-name)=include`

This option specifies to print the include directory.

`{--print-prog-name|-print-prog-name}={as|ld|objdump|ranlib|ar}`

This option specifies to print the name of programs which are called by compile command.

`--shared`

This option makes the linker create shared objects and not dynamic link executable files.

This option is passed to the linker.

`--version`

This option specifies to emit the version and copyright information of the compiler to the standard output.

`-Wp,-MD,filename`

This option is equivalent to the following options:

`-MD -MF filename`

`-Xlinker option`

This option directs that the *option* is to be passed as arguments to the linker.

`-dM`

This option specifies to output all macro definitions when the `-E` option is effective.

`-f{align-loops[=N]|no-align-loops}`

The `-falign-loops` option specifies to align the loop to the boundary of power-of-two byte.

The `-fno-align-loops` option invalidates the `-falign-loops` option. `-fno-align-loops` is set by default.

`-f{exceptions|no-exceptions}`

The `-fexceptions` option defines the `__EXCEPTIONS` macro. The `-fno-exceptions` option invalidates the `-fexceptions` option. `-fno-exceptions` is set by default.

`-ffast-math`

This option specifies to perform the math functions optimization that do not conform strictly to the specifications/rules of IEEE 754 or ISO. When this option is effective, it may give rise to side effects (calculation error or runtime exceptions) in the execution results.

`-ffp-contract=fast`

This option specifies to perform optimization using Floating-Point Multiply-Add/Subtract instructions.

When this option is set, side effects (calculation errors in rounding error extent) may occur in the execution results and produce unexpected results.

`-f{inline-functions|no-inline-functions}`

The `-finline-functions` option specifies to perform the inline expansion for simple functions defined in the source code instead of function calls. The function which is the subject of inline expansion is determined by the compiler automatically. The `-fno-inline-functions` option is set by default.

`-floop-parallelize-all`

This option specifies to perform automatic parallelization. However, if the effect of parallel execution is not expected, automatic parallelization is not performed.

If the object program compiled with this option is included in the filename list, this option must also be specified when linking.

`-fno-common`

This option specifies that the global variable without the initial value specification is allocated to the data section in the object file.

This option is useful for variables declared with the same name without the `"extern"` specifier in two or more sources, as this can be detected as an error at linking time.

`-f{omit-frame-pointer|no-omit-frame-pointer}`

The `-fomit-frame-pointer` option specifies to suppress the storing of the frame pointer in the function that need not store the frame pointer in the register. The `-fno-omit-frame-pointer` option is set by default.

-fopenmp

The **-fopenmp** option specifies to enable Specification of OpenMP Application Program Interface.

When the **-fopenmp** option is specified, **-mt** is set.

The **-fopenmp** option is needed if an object program compiled with the **-fopenmp** option exists in the command line as input files.

-f{openmp-simd|no-openmp-simd}

The **-fopenmp-simd** specifies that only the OpenMP **simd** construct and the **declare simd** construct are enabled.

-fno-openmp-simd is set by default.

-f{optimize-sibling-calls|no-optimize-sibling-calls}

The **-foptimize-sibling-calls** option specifies to perform optimization of the sibling call. The **-fno-optimize-sibling-calls** option invalidates the **-foptimize-sibling-calls** option. **-fno-optimize-sibling-calls** is set by default.

-f{pic|PIC}

This option specifies to generate position-independent code (PIC).

If the **-fPIC** option is specified, a slower object is generated with a long instruction sequence. However a number of unique global symbols are able to be used in the final shared library that the object will be a part of. If the **-fpic** option is specified, a faster object is generated with a short instruction sequence. However few of unique global symbols are able to be used in the final shared library that the object will be a part of. The number of the unique external symbols that can be referred for these cases is the total about all libraries united at the same time.

If the **-fPIC** and **-fpic** options are specified, only the later one is effective.

This option is effective at compilation time only.

-f{pie|PIE}

This option specifies to generate position-independent code.

If the **-fPIE** option is specified, a slower object is generated with a long instruction sequence. However a number of unique global symbols are able to be used in the final shared library that the object will be a part of. If the **-fpie** option is specified, a faster object is generated with a short instruction sequence. However few of unique global symbols are able to be used in the final shared library that the object will be a part of. The number of the unique external symbols that can be referred for these cases is the total about all libraries united at the same time.

If **-fPIE** and **-fpie** options are specified, only the later one is effective.

This option is effective at compilation time only.

The difference between these options and the **-f{pic|PIC}** options is that the objects can be linked with the **-pie** linker option to generate a position independent executable.

-f{plt|no-plt}

The **-fplt** option specifies to use Procedure Linkage Table (PLT) for accessing a global symbol in the position-independent code (PIC).

The **-fno-plt** option invalidates the **-fplt** option. **-fplt** is set by default.

-fprofile-dir=path

This option specifies storage location directory of the **.gdcda** file which is necessary for the use of the code coverage. For *path*, the storage location directory name is specified by the relative path or the absolute path.

-f{schedule-insns|no-schedule-insns}

The **-fschedule-insns** option specifies to perform the optimization of an instruction scheduling before the register is allocated. The **-fno-schedule-insns** option invalidates the **-fschedule-insns** option. **-fno-schedule-insns** is set by default.

-f{schedule-insns2|no-schedule-insns2}

The **-fschedule-insns2** option specifies to perform the optimization of an instruction scheduling after the register is allocated. The **-fno-schedule-insns2** option invalidates the **-fschedule-insns2** option. **-fno-schedule-insns2** is set by default.

-f{signed-char|unsigned-char}

The **-fsigned-char** option specifies that **char** declaration is assumed to be **signed char** declaration. The **-funsigned-char** option invalidates the **-fsigned-char** option. The **-funsigned-char** option is applied by default.

-f{strict-aliasing|no-strict-aliasing}

The `-fstrict-aliasing` option specifies that in case of indirect memory access through a pointer, when the accessing types are different, no memory alias is assumed. The `-fno-strict-aliasing` option invalidates the `-fstrict-aliasing` option. The `-fno-strict-aliasing` is set by default.

-funroll-loops

This option specifies to apply loop unrolling.

-f{unsafe-math-optimizations|no-unsafe-math-optimizations}

The `-funsafe-math-optimizations` option specifies to use flush-to-zero mode. The default is `-fno-unsafe-math-optimizations`.

When `-funsafe-math-optimizations` option is specified, side effects may occur in the execution results.

These options must be set at linking.

-g{dwarf|dwarf-4}

This option specifies to add DWARF4 debugging information to the object file.

This option is equivalent to the `-g` option.

-idirafter *dir*

This option adds the *dir* directory to the end of the standard search paths to search the headers.

If multiple directories are specified in multiple `-idirafter` options, the directories are searched in the specified order.

For details about the order of the search of the header, see the `-I` option in [Section "2.2.2.1 General Options for Compiler"](#).

If a header is specified with an absolute path name, only the specified absolute path name is searched.

If the specified directory does not exist, this option is invalidated.

-include *file*

This option specifies to include the *file* at the top of the source file.

If multiple files are specified in multiple `-include` options, the files are included in the specified order.

-isystem *dir*

This option adds the *dir* directory to the search paths to search the standard headers.

If multiple directories are specified in multiple `-isystem` options, the directories are searched in the specified order.

For details about the order of the search of the header, see the `-I` option in [Section "2.2.2.1 General Options for Compiler"](#).

If a header is specified with an absolute path name, only the specified absolute path name is searched.

If the specified directory does not exist, this option is invalidated.

-march=*arch*[+*features*]

This option specifies the architecture. In *arch*, specifies one of `armv8-a`, `armv8.1-a`, `armv8.2-a`, or `armv8.3-a`.

In *features*, specifies `sve` or `nosve`. The default for *features* is `sve`.

If `-march` option is omitted, `-march=armv8.3-a+sve` is set by default.

-mcmmodel={small|large}

This option specifies the possible largest size of the text area and the static data area in an executable program or a shared object. `-mcmmodel=small` is set by default.

-mcmmodel=small

The total size of the text area and the static data area is limited to 4GB at linking. This option creates an efficient object program.

-mcmmodel=large

Only the size of the text area is limited to 4GB at linking. This option is used when the static data area is large and an error occurs at linking.

-mcpu=*cpu*

This option specifies the processor. In *cpu*, specifies one of a64fx, thunderx2t99, or generic. -mcpu=a64fx is set by default.

-m{pc-relative-literal-loads|no-pc-relative-literal-loads}

The -mpc-relative-literal-loads option specifies to access the literal pool by one instruction assuming that the code area in the function is within 1MB.

The -mno-pc-relative-literal-loads option invalidates the -mpc-relative-literal-loads. -mno-pc-relative-literal-loads is set by default.

-mtls-size={12|24|32|48}

This option specifies the size of an offset necessary for the access to Thread-Local Storage. Units are bits.

-mtune=*cpu*

This option specifies the processor which optimizations targets. In *cpu*, specifies one of a64fx, thunderx2t99, or generic. -mtune=a64fx is set by default.

-nostartfiles

This option specifies not to use standard system startup files at linking.

-nostdinc

This option specifies not to use the standard directory to search headers.

-nostdlib

This option specifies not to use standard system startup files and standard libraries at linking.

-pthread

This option specifies to create thread safe objects by using the POSIX thread library.

-rdynamic

This option specifies to the linker to add all symbols to the dynamic symbol table.

This option passes the -export-dynamic option to the linker.

-undef

This option specifies to suppress the system specific and GNU C specific predefined macros.

-v

This option specifies to display the executing command which compile command called as compiler, assembler and linker.

-w

This option suppresses the output of warning messages.

-x *language* -

Specifies the type of input file. Specify "c" or "assembler-with-cpp" for *language*. If "c" is specified, it is treated as a C source file. If "assembler-with-cpp" is specified, it is treated as an assembler source file that requires preprocessing.

This system supports reading input files only from standard input. Therefore, specify "-" after *language* to represent standard input. Operation is not guaranteed when a file name is specified.

This option cannot be specified simultaneously with the options -Nlst[={p|t}], -Nlst_out=*file*, -Nsta, and -Nsrc.

This option cannot be specified multiple times in the command line.

Option example:

```
$ cat main.ddd | fccpx -x c -
```

Appendix E Data and Memory Regions

This appendix describes attribute of the data and memory regions in this system.

E.1 Data Size and Alignment

Data types, sizes, and alignments of basic data system are shown below.

Data type ^[a]	Size(byte)	Alignment (byte)
char	1	1
short int	2	2
int	4	4
long int	8	8
long long int		
float	4	4
double	8	8
long double	16	16
float _Complex	8	4
double _Complex	16	8
long double _Complex	32	16
struct	-	^[b]
union		

[a] Common to signed and unsigned.

[b] The largest value among the maximum alignment of the member variables and 8 byte.

E.2 Memory Regions

The main usages of the memory regions are shown below.

Memory regions	Usage
Text	Memory regions where instructions are arranged.
Static data(.data)	Memory regions where static data with initialization is stored.
Static data(.bss)	Memory regions where static data without initialization is stored.
Process stack	Stack regions for the process and the main thread.
Thread stack	Stack regions for the child process.
Dynamically allocated memory region	Memory regions allocated when requiring memory to be allocated dynamically, or memory regions allocated as thread heap region or thread stack.
Shared memory	Memory regions shared among processes.

E.3 Data Allocation

Variables in a program are arranged to different memory regions depending on each type of the variables and existence of initial values.



Example

```

int e[2];
int f[2] = {3,3};

int main(){
    int      a[2];
    int      b[2] = {1,1};
    static float c[2];
    static float d[2] = {2.0,2.0};
    double    *g;

    g = (double *)malloc(sizeof(double)*2);
    ...
}

```

In the above example, each array is allocated as follows.

Array name	Region used
a	Allocated to the process stack region, because these are local arrays.
b	
c	Allocated to the static data (.bss) region, because it is a static array and not initialized.
d	Allocated to the static data (.data) region, because it is a static array and initialized.
e	Allocated to the static data (.bss) region, because it is a global array and not initialized.
f	Allocated to the static data (.data) region, because it is a global array and initialized
g	Allocated to the dynamically allocated memory region.

E.4 Data Allocation to Stack Region

In this system, the order of data allocation to the stack region can be controlled by the compiler option.

The compiler option that controls the order of data allocation is shown as follows.

-N{reordered_variable_stack|noreordered_variable_stack}

The **-Nreordered_variable_stack** option directs the compiler the order in which to allocate the automatic variables to the stack area.

If the **-Nreordered_variable_stack** option is set, the allocation order of automatic variables is determined in the following order.

1. Descending order of their alignments
2. Descending order of data sizes if the alignments are equal
3. The order of appearance of the declaration statements in the source program if both the alignments and data sizes are equal

The stack area of the entire program can be reduced by allocating the automatic variables in descending order of their alignments.

When the **-Nnoreordered_variable_stack** option is specified, automatic variables are allocated in order of the appearance of the declaration statements in the source program. **-Nnoreordered_variable_stack** is set by default.

The order of allocation is not guaranteed when the **-Nnoline**, **-g0**, or **-Kocl** option is set.



Example

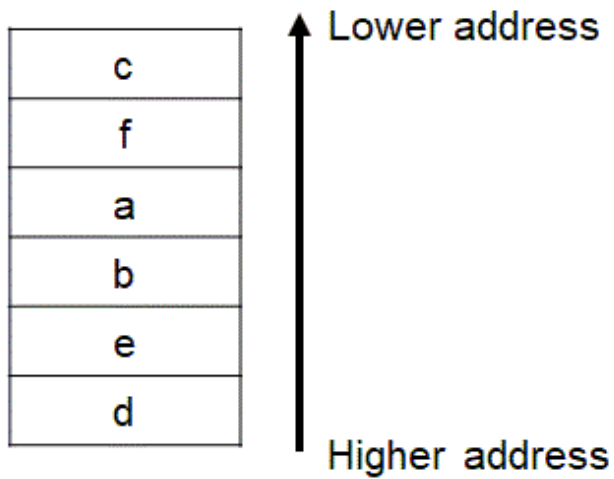
- Source Program

```
int    a;  
double b;  
char   c;  
double d[2],e[2];  
short int f;  
a = 1;  
b = 2.0;  
c = '3';  
d[0] = 4.0;  
e[0] = 5.0;  
f = 6;
```

- When specifying the order of data allocation for the example source program

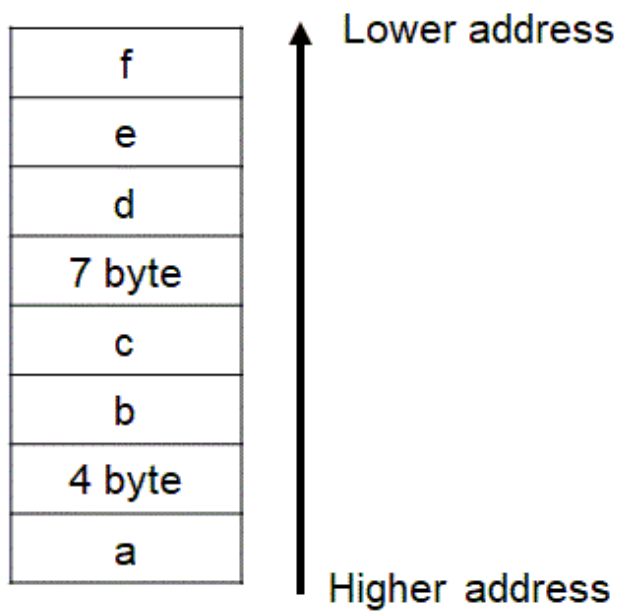
Automatic variables are stored in the stack as shown below. The stack region can be decreased.

Figure E.1 When the `-Nreordered_variable_stack` option is valid



- When not specifying the order of data allocation for the example source program Automatic variables are stored in the stack as shown below.

Figure E.2 When the `-Nnoreordered_variable_stack` option is valid



Appendix F Code Coverage

This appendix explains the code coverage.

The code coverage is the function to measure the execution frequency of the executable statement at execution, and to examine the code coverage rate of the program.

The code coverage uses the coverage tool based on LLVM compiler infrastructure (hereafter called llvm-cov), which is an open source software.

Refer to an online manual of the llvm-cov command by the man command for the use of the llvm-cov command.

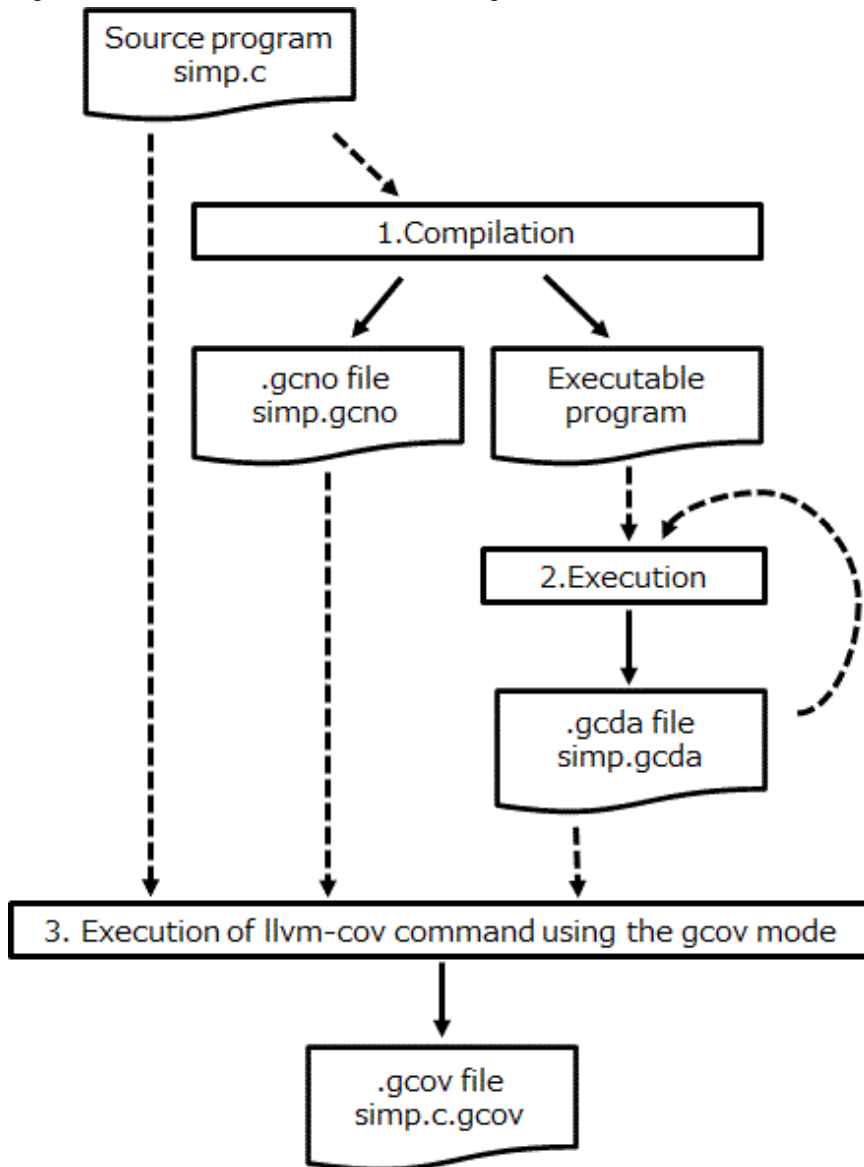
F.1 How to Use Code Coverage

The code coverage is used the following procedures.

1. Compilation
2. Execution
3. Execution of llvm-cov command using the gcov mode

A following figure shows the procedure of the code coverage.

Figure F.1 Procedure of the code coverage



F.2 Necessary Files to Use Code Coverage

This section explains the necessary files to use the code coverage.

F.2.1 The ".gcno" File

The .gcno file is a binary file which contains the information obtained at compilation.

File generation step	The file is generated at compilation
File name	The file name is the same as the source program with file extension .gcno.
File generated location	The .gcno file is generated in the directory where the compilation is performed.

An example of generating .gcno file is shown as follows.



Example

```
$ fcc -Ncoverage ../simp.c
$ ls
a.out simp.gcno
```

When the assembler source file and the object file are generated with specifying the -o option, the -o option is also applied to the .gcno file. An example of generating files with specifying the -o option is shown as follows.



Example

```
$ fcc -Ncoverage ../simp.c -c -o www/yyy.o
$ ls www/
yyy.gcno yyy.o
```

F.2.2 The ".gcda" File

The .gcda file is a binary file which contains the information obtained at execution.

File generation step	The file is generated at execution. The execution frequency is accumulated when the .gcda file with the same name already exists.
File name	The file name is the same as the source program with file extension .gcda.
File generated location	The .gcda file is generated in the directory where the compilation is performed.

An example of generating .gcda file is shown as follows.



Example

```
$ fcc -Ncoverage ../simp.c
$ ls
a.out simp.gcno
$ ./a.out
$ ls
a.out simp.gcda simp.gcno
```

When the assembler source file and the object file are generated with specifying the -o option, the -o option is also applied to the .gcda file. An example of generating files with specifying the -o option is shown as follows.



Example

```
$ fcc -Ncoverage ../simp.c -c -o www/yyy.o
$ ls www/
yyy.gcno yyy.o
$ fcc -Ncoverage www/yyy.o
$ ./a.out
$ ls www/
yyy.gcda yyy.gcno yyy.o
```

The storage location directory of the .gcda file can be changed by specifying the -Nprofile_dir=*dir_name* option.

An example of generating files with specifying the -Nprofile_dir=*dir_name* option is shown as follows.



Example

```
$ gcc -Ncoverage ../simp.c -c -o www/yyy.o -Nprofile_dir=/tmp
$ ls www/
yyy.gcno yyy.o
$ gcc -Ncoverage www/yyy.o
$ ./a.out
$ ls www/
yyy.gcno yyy.o
$ ls /tmp/www/
yyy.gcda
```

F.3 Notes on Code Coverage

This section explains notes of the code coverage.

- The execution performance may decrease because the instructions that measure the execution frequency are added in the object program.
- The storage directory of the .gcda file, which is generated at execution, is determined at compilation.
 - The location of the .gcda file storage directory does not change even if the executable program is moved after compilation.
 - Use the `-Nprofile_dir=dir_name` option at compilation to change the location of the .gcda file storage directory.
- The code coverage function is not available for programs that need the `-Kcmodel=large` or `-mcmodel=large` option.
- The execution frequency may be measured incorrectly in the following cases:
 - Two or more executable statements are included in one line.
 - The `exit(3)` function is called.
 - The `setjmp` or `longjmp` function is called.
 - An exception is caught.
 - The `#line` directive is included in the source program.
 - Optimizations of the compiler are applied.

Appendix G Fujitsu Extended Functions

This appendix describes the Fujitsu Extended Functions supported in this system.

G.1 How to Use Fujitsu Extended Functions

G.1.1 Header

Fujitsu Extended Functions need a header shown below to be included in the source file.

- fjcex.h

G.1.2 Supported Functions

The system supports Fujitsu Extended Functions below.

Table G.1 List of Fujitsu Extended Functions

Function Prototype	Description
double __gettod(void)	The __gettod() function returns the current high-resolution real time, expressed as micro-seconds since the some arbitrary time in the past. Typically, the time is expressed since the most recent system booting.

Appendix H Runtime Information Output Function

This appendix describes the Runtime Information Output Function.

This function outputs information for speedup and performance confirmation of the program, by specifying the compiler option and setting the environment variable, when the program is executed.

H.1 Usage of Runtime Information Output Function

This section describes the usage of Runtime Information Output Function.

H.1.1 Range of Obtainable Information

This section describes the range of information that can be fetched using this function.

Table H.1 Range of obtainable information

Compiler Options	Runtime environment variables	Range of obtainable information
-Nrt_tune	FLIB_RTINFO	from the start of the program to the end
-Nrt_tune_func		each user-defined function
-Nrt_tune_loop		each loop

When the -Nrt_tune compile option and FLIB_RTINFO runtime environment variable is specified, this function is enabled and information can be fetched from the start of the program to the end.

Also, if the -Nrt_tune_func compile option or -Nrt_tune_loop compile option is specified, additional information concerning each function, each loop can be fetched.

Refer to "2.2 Compiler Options" for further details.

If the MPI Programs, information on the rank 0 is output. If the COARRAY Programs, information on the image 1 is output.

Each User-defined Function

When the -Nrt_tune_func compile option is specified, information for each user-defined function is output.

When information is fetched for each user-defined function, the information below is fetched in the cases shown below.

- If a function is called from within a function:
- Information for the called function is not included in the calling source function information.
- If the same function is called from multiple locations :
- The total values are used as the information for that function.
- Only user-defined function s are output. Information concerning system calls and library functions is not fetched.

Each Loop

When the -Nrt_tune_loop compile option is specified, information for each loop is output.

The following information is output, as information for each loop.

- Information for each loop operates in serial processing.
- Information for each loop operates in parallel region by OpenMP.

The cost of the serial loop is output as information for serial loop. If the loop exists in parallel region by OpenMP, it is output as each loop in parallel region by OpenMP. As the information of loop in parallel region by OpenMP, information of the master thread is output.

When information is fetched for each loop, the information below is fetched in the cases shown below.

- If a function is called from within a loop :

- Information for the called function is included in the calling source loop information.
- If a loop is called from within a loop :
- Information for the called loop is included in the calling source loop information.

H.1.2 Runtime Environment Variables

This section describes the runtime environment variables using this function.

The English small letter and the English capital letter specified for the value of the environment variable at execution time are distinguished. For environment variables that do not require operands, the specified operand value is ignored.

Table H.2 Environment variables that control the information output

Runtime environment variables		Explanation
Variable name	Operand	
FLIB_RTINFO	(None)	Outputs runtime information.
FLIB_RTINFO_NOFUNC	(None)	Even if the compiler option -Nrt_tune_func is effective, the output of information for each function is suppressed.
FLIB_RTINFO_NOLOOP	(None)	Even if the compiler option -Nrt_tune_loop is effective, the output of information for each loop is suppressed.
FLIB_RTINFO_CSV	<i>Filename</i>	Outputs the fetched information to a CSV file.
	-	Operand is optional. Any file name can be specified as operand of the environment variable. If operand is omitted, flib_rtinfo.csv is assumed.

H.2 Output of Runtime Information Output Function

This section describes the output of Runtime Information Output Function.

H.2.1 Output Information

This section describes the information output by this function.

Table H.3 Output information

Compiler Options	Runtime environment variables	Output information
-Nrt_tune	FLIB_RTINFO	The following information from the start of the program to the end is output <ul style="list-style-type: none"> - Elapsed time - User CPU time - System CPU time
-Nrt_tune_func		In addition to the -Nrt_tune output, following Information of each function is output <ul style="list-style-type: none"> - Function cost - Percentage of total cost accounted for by the function cost - Function cost per called once - Function start line number - Function end line number - Call frequency of function

Compiler Options	Runtime environment variables	Output information
		- Function name
-Nrt_tune_loop		<p>In addition to the -Nrt_tune output, following Information of each loop in serial and each loop that in parallel region by OpenMP is output</p> <ul style="list-style-type: none"> - Loop cost - Percentage of total cost accounted for by the loop cost - Loop cost per called once - Loop start line number - Loop end line number - Call frequency of Loop - Loop nest level - Generated function name

H.2.2 Output Format

Runtime information can be output by the text format or CSV format.

Text format

When FLIB_RTINFO_CSV environment variable is not specified, information is output to the standard output by text format.

Refer to "[H.2.3 Output Example](#)" for details of output format of information.

CSV format

When the FLIB_RTINFO_CSV environment variable is specified, information is output in CSV format. If a filename is specified in the operands, the fetched information is output to the specified file. If the specified file already exists, the information overwrites the existing file. If the file fails to open, runtime message jwe1653i-w is output and the information is output to the standard output.

When runtime information is output by CSV, data line follows the following compositions. When there is no output information, "-" is output.

Table H.4 Composition of CSV format

Column position	Output information
First	<p>Identifier that shows type</p> <p>[COST] : Information for entire program</p> <p>[COST_ROUTINE] : Information for each function</p> <p>[COST_LOOP_SERIAL] : Information for each loop that operates in serial processing</p> <p>[COST_LOOP_PRL] : Information for each loop that in parallel region by OpenMP</p>
Second and subsequent	It is depending on output information.

Refer to "[H.2.3 Output Example](#)" for details of output format of information.

H.2.3 Output Example

Output example of Runtime Information Output Function is shown below.

Output example of runtime information

```

++=====
++

```

EXECUTION PERFORMANCE INFORMATION										
++=====++										
++										
+(1)-----+(2)-----+(3)-----+										
Elapsed	User	System								
(sec)	(sec)	(sec)								
+-----+-----+-----+										
2.5691	2.5636	0.0020								
+-----+-----+-----+										
Routine (4)										
+(5)-----+(6)---+(7)-----+(8)----+(9)----+(10)---+(11)-----+										
Cost(sec)	%	Once(sec)	Start	End	Count		Routine name			
+-----+-----+-----+-----+-----+-----+-----+										
1.9691	84.41	0.0010	71	86	2000	sub2				(12)
0.3638	15.59	0.0364	38	68	10	sub1				
+-----+-----+-----+-----+-----+-----+-----+										
2.3329	100.00	-	-	-	-	-				(13)
+-----+-----+-----+-----+-----+-----+-----+										
Serial Loop (14)										
+-----+-----+-----+-----+-----+-----+(15)+-----+										
Cost(sec)	%	Once(sec)	Start	End	Count	Nest		Routine name		
+-----+-----+-----+-----+-----+-----+-----+										
2.1979	45.25	2.1979	25	27	1	1	main			(16)
2.1658	44.59	0.0011	77	78	2000	2	(sub2)			
+-----+-----+-----+-----+-----+-----+-----+										
4.8568	100.00	-	-	-	-	-	-			(17)
+-----+-----+-----+-----+-----+-----+-----+										
Parallel Loop (18)										
+-----+-----+-----+-----+-----+-----+-----+										
Cost(sec)	%	Once(sec)	Start	End	Count	Nest		Routine name		
+-----+-----+-----+-----+-----+-----+-----+										
0.2462	50.04	0.0246	58	64	10	2	(sub1._OMP_1)			
0.2458	49.96	0.0002	59	60	1000	3	(sub1._OMP_1)			
+-----+-----+-----+-----+-----+-----+-----+										
0.4919	100.00	-	-	-	-	-	-			
+-----+-----+-----+-----+-----+-----+-----+										

- (1) Elapsed time
- (2) User CPU time
- (3) System CPU time
- (4) Information for each function that operates in serial processing (-Nrt_tune_func is specified)
- (5) Cost (sec)
- (6) Percentage of the entire cost
- (7) Cost called once
- (8) Start line number
- (9) End line number
- (10) Call frequency
- (11) Function name
- (12) Information for each function
- (13) Total value for each function
- (14) Information for each loop that operates in serial processing (-Nrt_tune_loop is specified)

(15) Nest level

(16) Information for each loop

(17) Total value for each loop

(18) Information for each loop that in parallel region by OpenMP (-Nrt_tune_loop is specified)

Note: When information is fetched for each loop, the generated function name is output as the function name. The function name is enclosed with parentheses at the loop that is nested.

Output example of runtime information (environmental variable FLIB_RTINFO_CSV is specified)

```
=====
EXECUTION PERFORMANCE INFORMATION
=====
Type,Elapsed(sec),User(sec),System(sec)
[COST],2.5525,2.5716,0.0020

Routine
Type,Cost(sec),%,Once(sec),Start,End,Count,Routine name
[COST_ROUTINE],1.9597,84.39,0.0010,71,86,2000,"sub2"
[COST_ROUTINE],0.3626,15.61,0.0363,38,68,10,"sub1"
[COST_ROUTINE],2.3222,100.00,-,-,-,-,-

Serial Loop
Type,Cost(sec),%,Once(sec),Start,End,Count,Nest,Routine name
[COST_LOOP_SERIAL],2.1828,45.18,2.1828,25,27,1,1,"main"
[COST_LOOP_SERIAL],2.1596,44.70,0.0011,77,78,2000,2,"(sub2)"
[COST_LOOP_SERIAL],0.2276,100.00,-,-,-,-,-

Parallel Loop
Type,Cost(sec),%,Once(sec),Start,End,Count,Nest,Routine name
[COST_LOOP_PRL],0.2476,50.06,0.0248,58,64,10,2,"(sub1._OMP_1)"
[COST_LOOP_PRL],0.2470,49.94,0.0002,59,60,1000,3,"(sub1._OMP_1)"
[COST_LOOP_PRL],2.3222,100.00,-,-,-,-,-
```

H.3 Notes on Runtime Information Output Function

This section describes the notes on Runtime Information Output Function.

- If the information is fetched each function or loop by specifying -Nrt_tune_func or -Nrt_tune_loop, fetched information is increases, and execute time may increase. If the information cannot be obtained, the diagnostic message jwe1655i-i may be output. If this diagnostic message is output during error processing, the order in which the diagnostic messages are output may be incorrect.
- When there are goto statement, setjmp()/longjmp() during a program or an exceptions is checked, information may be not correctly fetched.
- There is a possibility of receiving the following influences by optimization of the compiler.
 - When fetched the information for each loop, it may be different which loop is worked as a target of the measurement.
 - Line number may not accord with line number of the source program.
 - The nest relations of the loop may not accord with a nest of the source program. By the following optimization, I may not recognize loop information definitely. By the following optimization, it may not be correctly recognized loop information.
 - inline expansion for functions
 - loop unrolling
 - loop blocking
 - loop interchange
 - loop fusion
 - loop striping

- loop splitting
- loop peeling
- loop unswitching
- change the function to a multi-operation function, use SIMD instructions

Appendix I Using High-Speed Facility on Job Operation Software

On the job of Job Operation Software, the high-speed facility (the inter-core hardware barrier and the Sector cache) can be used.

This appendix provides information on the compilation and execution processes for using the high-speed facility.

See manuals of Job Operation Software for details about how to execute and submit a job.

I.1 Inter-Core Hardware Barrier

The inter-core hardware barrier (hereafter called hardware barrier) is a hardware mechanism which facilitates high speed synchronization among threads in a CPU Chip, and raises the execution performance. This section describes compiling and executing programs to take advantage of hardware barriers using the LLVM OpenMP library.

I.1.1 Compilation

When generating a program using the hardware barrier as the thread barrier, specify `-Kparallel` or `-Kopenmp` option.

I.1.2 Execution

`FLIB_BARRIER`

The barrier-kind of OpenMP and Automatic Parallelization can be controlled using the environment variable `FLIB_BARRIER`.

Allowable values and their meanings are as follows:

`HARD`

The hardware barrier is used.

When the hardware barrier cannot be used, the diagnostic message (jwe1050i-w) is output, and the processing is continued using the software barrier.

When the hardware barrier is used, the part or everything of the following OpenMP functions, cannot be used. When these OpenMP functions are used, the diagnostic message (jwe1051i-w) is output, and these functions are ignored, the processing is continued using the software barrier.

- The controlling number of threads

The number of threads will be determined using the following priority.

1. The value specified in the environment variable `OMP_NUM_THREADS`
2. The limit on the number of CPUs (The number of CPUs assigned to the process by the job)

When the value specified in the environment variable `OMP_NUM_THREADS` exceeds the limit on the number of CPUs, the number of threads is the limit on the number of CPUs.

The number of threads specified with the `num_threads` clause or the `omp_set_num_threads` function must be the same as defined by the execution environment.

Irrespective of the environment variable `OMP_DYNAMIC` and the `omp_set_dynamic` function, the number of threads cannot exceed the limit on the number of CPUs.

- The controlling thread affinity

The thread affinity is set using CPUs assigned to the process by the job.

Irrespective of the environment variable `OMP_PROC_BIND` and the `proc_bind` clause of `parallel` construct, OpenMP thread affinity is ignored.

When the thread affinity is set using other ways, the operation is not guaranteed.

- Nested parallelism.

Irrespective of the environment variable `OMP_NESTED` and the `omp_set_nested` function, nested `parallel` regions are always serialized.

- Task

Always, the undeferred task is generated.

- Cancellation

Irrespective of the environment variable `OMP_CANCELLATION`, the `cancel` construct and cancellation points are effectively ignored.

SOFT

The software barrier is used. (default)



Note

- When the number of threads is 1, the hardware barrier cannot be used.
- The hardware barrier in FX system is used only inside the NUMA node. When one process is allocated to multiple NUMA nodes, hardware barrier is used inside NUMA node and software barrier is used between NUMA nodes. Therefore, it is recommended to allocate one process to one NUMA node, in case of giving priority to the performance of threads barrier.
- On the following cases, the high speed runtime library cannot determine whether one process can reserve a Barrier resource or not. Therefore behavior of hardware barrier is indeterminate, program may be ended with diagnostic-message (jwe1044i-u or jwe1045i-u), or the program may terminate abnormally.

- When Process is generated by functions other than Fujitsu MPI

Example:

- FORTRAN `FORK/SYSTEM/SH` service function is used.
- C `fork` system call/system function is used.



Example

Example 1: A program that uses the hardware barrier is executed on a thread parallel processing job.

```
$ cat job.sh
#!/bin/sh
export FLIB_BARRIER=HARD
./a.out

$ pjsub -L "node=1" job.sh
```

Example 2: A program that uses the hardware barrier is executed on a hybrid (process and thread) parallel processing job (One job is executed in one dimension: The number of virtual nodes is set to the number of processes.).

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=12"
#PJM --mpi "shape=12"
#PJM --mpi "rank-map-bynode"
export FLIB_BARRIER=HARD
mpiexec -n 12 a.out

$ pjsub job.sh
```

I.2 Sector Cache

See Section "[3.5 Software Control of Sector Cache](#)" for details on how to control the Sector cache.

This section explains only notes on execution.

Notes on execution

The Sector cache for the primary and secondary level caches are available in the following execution environments.

The execution environments	The Sector cache for the first level cache	The Sector cache for the second level cache
Only one process runs on a NUMA node	Available	Available
Multiple processes run on a NUMA node	Available (*1)	Not available

*1) Not available if the environment variable `FLIB_L1_SCCR_CNTL` is assigned to the value `FALSE`. See Section "[3.5.2.2 Software Control with Environment Variables and Optimization Control Line](#)" for details.

On the following cases, the high speed runtime library cannot determine whether one process can reserve a NUMA node or not. Therefore behavior of Sector cache is indeterminate. Performance of program may be reduced, program may be ended with diagnostic-message (jwe1048i-u), or the program may terminate abnormally.

- Process is generated by functions other than Fujitsu MPI.

Example:

- FORTRAN FORK/SYSTEM/SH service function is used.
- C fork system call/system function is used.
- Threads are generated by functions other than Fujitsu Automatic Parallelization/OpenMP.

Example:

- Threads are controlled by pthread functions.

To invalidate the Sector cache control function unconditionally regardless of the execution environment, "FALSE" is set for the environment variable `FLIB_SCCR_CNTL`.

See Section "[3.5.2.2 Software Control with Environment Variables and Optimization Control Line](#)" for details.

Appendix J Fujitsu OpenMP Library

This appendix describes how parallel processing is performed for a C program using Fujitsu OpenMP Library in Trad Mode.

See "[Chapter 4 Multiprocessing](#)" when you use LLVM OpenMP Library.

Note

If all of the following conditions are met, this system uses Fujitsu OpenMP Library.

- `-Nfjomplib` option is effective at linking
- Trad Mode compiler is used at both compiling and linking

Information

The table below shows a connectable combination of library for multiprocessing (LLVM OpenMP Library and Fujitsu OpenMP Library) and connectable object file.

	Fortran object file	C/C++ object file	
		Trad Mode (-Nnoclang option)	Clang Mode (-Nclang option)
LLVM OpenMP Library (-Nlibomp option)	Available	Available	Available
Fujitsu OpenMP Library (-Nfjomplib option)	Available	Available	Not available

J.1 Overview of Multiprocessing

See Section "[4.1 Overview of Multiprocessing](#)".

J.2 Automatic Parallelization

This section describes automatic parallelization.

J.2.1 Compilation (Automatic Parallelization)

Following options are used to invoke automatic parallelization.

- Compile time

```
-Kparallel
```

- Link time

```
-Kparallel -Nfjomplib
```

Compiler Option for Automatic Parallelization

The compiler option for automatic parallelization is:

```
-K{parallel|parallel_strong|visimpact}  
[,array_private,dynamic_iteration,instance=N,loop_part_parallel,ocl,optmsg=2,parallel_fp_precision,p  
arallel_iteration=N,reduction,region_extension] -Nfjomplib
```


See Section ["2.2 Compiler Options"](#) for details on how to use each option.

J.2.2 Execution Process (Automatic Parallelization)

When a multiprocessing program is executed, the number of threads to be used may be specified by the environment variable `PARALLEL` or `OMP_NUM_THREADS`. The size of the thread stack may be specified by the environment variable `THREAD_STACK_SIZE` or `OMP_STACKSIZE`. The method of synchronizing threads can be changed using the environment variable `FLIB_SPINWAIT` or `OMP_WAIT_POLICY`.

Except these environment variable settings, the procedure for execution is same as for a serial-processing program.

Environment Variable `PARALLEL`

The number of threads to be used may be specified by the environment variable `PARALLEL`. Specify a number from 1 to 2147483647.

For more information on the number of threads, see ["Number of Threads"](#) in Section ["J.2.2 Execution Process \(Automatic Parallelization\)"](#).

Number of Threads

The number of threads is determined with the following priority.

1. The value of the environment variable `PARALLEL`
2. The value of the environment variable `OMP_NUM_THREADS`
3. The number of CPUs that can be used with jobs
4. 1 thread



Note

Value of `PARALLEL`

If the `-Kopenmp` option is specified at the linkage phase of the compilation and the environment variable `FLIB_FASTOMP` is set as `TRUE`, the value of the environment variable `OMP_NUM_THREADS` and `PARALLEL` must be the same. Otherwise, a message (`jwe1042i-w`) is output and the smaller value of `OMP_NUM_THREADS` and `PARALLEL` is adopted.

Finally, the actual number of threads is the minimum number of threads determined by the rule above and the limit on the number of CPUs.

See Section ["J.4.1 Management of CPU Resources"](#) for details of the number of CPU that can be used with jobs.

The limit on the number of CPUs is determined by the following rule.

For jobs	The number of CPUs that can be used with jobs
Not for job	The number of CPUs in the system

See ["Environment Variable for OpenMP Specifications"](#) in Section ["J.3.2 Execution Process \(OpenMP Parallelization\)"](#) for the `OMP_NUM_THREADS` environment variable.

Environment Variable `THREAD_STACK_SIZE` and `OMP_STACKSIZE`

`THREAD_STACK_SIZE`

Specify the stack size (in Kbytes) for each thread using the environment variable `THREAD_STACK_SIZE`. Specify an integer value between from 1 to 18446744073709551615.

When the environment variable `OMP_STACKSIZE` is also specified, the greater value of the two is used as the stack area size for each thread. Refer to ["Stack Size on Execution"](#) in Section ["J.2.2 Execution Process \(Automatic Parallelization\)"](#) for information on stack area.

OMP_STACKSIZE

The stack area size for each thread can be specified using the environment variable OMP_STACKSIZE in byte, Kbytes, Mbytes, or Gbytes.

When the environment variable THREAD_STACK_SIZE is also specified, the greater value of the two is used as the stack area size. Refer to "[Stack Size on Execution](#)" in Section "[J.2.2 Execution Process \(Automatic Parallelization\)](#)" for information on stack area.

Stack Size on Execution

Local variables in the range of a parallelized loop are allocated on the stack region. If there is not enough space for the stack area to allocate local variables, the stack area needs to be extended to an appropriate size.

When the stack area size for each thread needs to be specified, specify the environment variable THREAD_STACK_SIZE or OMP_STACKSIZE.

When the environment variable is not specified, the stack size of each thread is the same as the stack size of the process.

However, if the maximum stack size for the process is larger than the value calculated by the expression "min(the size of available memory in this system, the size of a virtual memory) / (the number of threads)", the system will determine the stack area size and allocate it as follows:

```
the stack size for each thread (byte) =  
(min(the size of available memory in this system, the size of a virtual memory) / (the number of  
threads)) / 5
```

However, the assumed value does not guarantee that the system works correctly. It is recommended that an appropriate value for maximum process stack size be specified manually. The maximum process stack area size can be specified using the bash built in command "ulimit".

Note that the maximum virtual memory size can also be confirmed using ulimit (bash built-in command).

See "[Number of Threads](#)" in Section "[J.2.2 Execution Process \(Automatic Parallelization\)](#)" for details.



Note

Some types of pthread library do not support setting the size of the stack area because of fixed stack. Static types of pthread library tend to be fixed stack, so ensure that you use a pthread library that is of a shared type.

Synchronization Process

The synchronization process can be controlled with the environment variable FLIB_SPINWAIT or OMP_WAIT_POLICY.

FLIB_SPINWAIT

unlimited	Uses spin wait until all threads are synchronized. The default is unlimited.
0	Uses no spin wait but suspending wait.
<i>ns</i>	Uses spin wait until after <i>n</i> seconds, and then uses suspending wait. <i>n</i> is an integer value that must be positive. The unit "s" is specified following <i>n</i> .
<i>ms</i>	Uses spin wait until after <i>n</i> milliseconds, and then uses suspending wait. <i>n</i> is an integer value that must be positive. The unit "ms" is specified following <i>n</i> .

Spin wait is a synchronization method that allows all threads to consume CPU time, while suspending wait consumes less CPU time. Since spin wait is more efficient in parallel execution, "unlimited" is suitable for reducing elapsed time. On the other hand, "0" is suitable for reducing CPU time.

OMP_WAIT_POLICY

ACTIVE	Uses spin wait until all threads are synchronized. The default is ACTIVE.
--------	--

PASSIVE	Uses no spin wait but suspending wait.
---------	--

Select ACTIVE to give priority to the elapsed time. Select PASSIVE to give priority to the CPU time.

The settings of environment variable FLIB_SPINWAIT will be valid when environment variable FLIB_SPINWAIT is specified.

Restriction of fork(2)

Functions that depend on fork(2) are restricted.

CPU binding for thread

CPU binding for thread differs between job execution and non-job execution.

- When executing program on non-job

The thread is not bound to a CPU. But you can control CPU binding for thread using environment variable FLIB_CPU_AFFINITY.

The environment variable FLIB_CPU_AFFINITY

Threads are bound to CPUs in order of the specified cpuid list.

When the number of specified CPUs is exceeded, it is repeatedly used from the beginning of the list.

The cpuid list shall be separated by comma (',') or space (' ').

The cpuid list can have the next form that has range with increment.

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: cpuid of the beginning of the range. ($0 \leq \text{cpuid1} < \text{CPU_SETSIZE}$)

cpuid2: cpuid of the end of the range. ($0 \leq \text{cpuid2} < \text{CPU_SETSIZE}$)

inc: increment ($1 \leq \text{inc} < \text{CPU_SETSIZE}$)

In addition, it is necessary to be the following.

```
cpuid1 <= cpuid2
```

It becomes equivalent to the case where all CPUs for every increment value *inc* in the range from *cpuid1* to *cpuid2* are specified.

The cpuid can be used the above-mentioned value.

However, cpuid which can actually be assigned becomes only within the limits of CPU affinity of the process at the start of execution.

See CPU_SET(3) about details of CPU_SETSIZE.



Note

If cpuid is outside the CPU affinity of the process at the start of execution, the program outputs error messages. Correct the setting value.



Example

Example 1:

```
$ export FLIB_CPU_AFFINITY="12,14,13,15"
```

The thread is bound to CPU in order of 12, 14, 13, and 15.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 2:

```
$ export FLIB_CPU_AFFINITY="12-19"
```

The thread is bound to CPU in order of 12, 13, 14, 15, 16, 17, 18, and 19.

When the number of threads is nine or more, it is repeatedly used from the beginning of the list.

Example 3:

```
$ export FLIB_CPU_AFFINITY="12-19:2"
```

The thread is bound to CPU in order of 12, 14, 16, and 18.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 4:

```
$ export FLIB_CPU_AFFINITY="12-16:2,13,19"
```

The thread is bound to CPU in order of 12, 14, 16, 13, and 19.

When the number of threads is six or more, it is repeatedly used from the beginning of the list.

.....

- **When executing program on job**

When thread is not bound to CPU, you may control CPU binding with `FLIB_CPU_AFFINITY` as executing on non-job.

When thread is bound to CPU on job, the environment variable `FLIB_CPU_AFFINITY` becomes invalid.

For details of CPU binding, see Section "[J.4.2 CPU Binding](#)".

J.2.3 Example of Compilation and Execution

Example 1:

```
$ fccpx -Kparallel,reduction,ocl -Nfjomplib test1.c
$ ./a.out
```

In this example, reduction parallelization optimization and the optimization control lines (OCL) are in effect during compilation. The number of threads using at the runtime is defined by the execution environment. For details about the number of threads, see "[Number of Threads](#)" in Section "[J.2.2 Execution Process \(Automatic Parallelization\)](#)".

Example 2:

```
$ fccpx -Kparallel -Nfjomplib test2.c
$ PARALLEL=2
$ export PARALLEL
$ ./a.out
$ PARALLEL=4
$ export PARALLEL
$ ./a.out
```

The environment variable `PARALLEL` is set to 2 and the program executes with two threads.

Next, the environment variable `PARALLEL` is set to 4 and the program executes with four threads.

J.2.4 Performance Tuning

See Section "[4.2.4 Performance Tuning](#)".

J.2.5 Automatic Parallelization

See Section "[4.2.5 Feature Details on Automatic Parallelization](#)".

J.2.6 Optimization Control Line

See Section "[4.2.6 Optimization Control Line](#)".

J.2.7 Notes on Automatic Parallelization

See Section "[4.2.7 Notes on Automatic Parallelization](#)".

J.2.8 Linking with Other Multi-Thread Programs

This section describes the restrictions of Linking with other multi-thread programs.

OpenMP Program on this System

An object program of an OpenMP program created by this system can be linked with an object program created by this system.

Other Multi-thread Programs

An object program created by this system cannot be linked with an object program created by another multi-thread system.

However, the program may be linked with a pthread program when using the environment variable `FLIB_PTHREAD`.

Environment Variable `FLIB_PTHREAD`

You can control the linking with a pthread program, using the environment variable `FLIB_PTHREAD`.

Valid values are as follows. Default value is 0.

Value	Explanation
0 (Default)	<p>The program can be linked with a pthread program which uses the pthread thread as a control thread.</p> <p>However, there are the following restrictions.</p> <ul style="list-style-type: none">- The pthread thread must use suspending wait.- The pthread program must not be compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option. (*)- The routine which is compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option must not be used in the pthread program. (*)- The Fortran routine must not be used in the pthread program. (*)
1	<p>The program can be linked with a pthread program which executes parallel processing.</p> <p>And, the program can be linked with a pthread program which uses the pthread thread as a control thread.</p> <p>However, there are the following restrictions.</p> <ul style="list-style-type: none">- A pthread parallel processing, and an OpenMP or automatic parallel processing must be executed in order. (*) An OpenMP or automatic parallel processing must not be executed in a pthread parallel processing. And, a pthread parallel processing must not be executed in OpenMP or automatic parallel processing.- The pthread thread must use suspending wait.- The pthread program must not be compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option. (*)- The routine which is compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option must not be used in the pthread program. (*)- The Fortran routine must not be used in the pthread program. (*)

Value	Explanation
	<ul style="list-style-type: none"> - Fujitsu's math libraries must not be used in the pthread program. (The pthread program must not be compiled with -SSL2BLAMP option.) (*) - You must execute OpenMP parallel processing with two or more threads, before creating pthread threads. And, this number of OpenMP threads cannot be changed later. <p>And, when this function is used, the following functions are effective.</p> <ul style="list-style-type: none"> - FLIB_SPINWAIT=0 - FLIB_CPUBIND=off <p>Operation is not guaranteed when a value which is different in the above-mentioned environment variable is specified.</p>

*) Operation is not guaranteed when the restriction function is used.

J.3 Parallelization by OpenMP Specification

This section details parallelization as described in the OpenMP Specification. For more details about the OpenMP specification, refer to the OpenMP Architecture Review Board's website.

This system supports the following specifications.

Supported Specifications
OpenMP 3.1
Part of OpenMP 4.0 ^[a]
Part of OpenMP 4.5 ^[a]

[a] The following functions can be used:

- `simd` construct
- `declare simd` construct

J.3.1 Compilation (OpenMP Parallelization)

For compiling source programs and linking, specify the following options to the compile command.

The specified option is different by whether to perform parallelization by the OpenMP specification.

Parallelization by the OpenMP specification	Specified option	
	Compile time	Link time
When using the parallelization and the SIMD Extensions	-Kopenmp	-Kopenmp -Nfjomplib
When using the SIMD Extensions only, not using the parallelization	-Kopenmp_simd	-

-: There is no specified option.

Compiler Option for OpenMP C Program

This section describes the compilation options used to compile OpenMP programs.

-Kopenmp

This option enables OpenMP directives and compiles the source programs.

The -Kopenmp option should also be specified at the linkage phase of an object file, when the object file has previously been compiled with this option.



Example

```
$ fccpx a.c -Kopenmp -c
$ fccpx a.o -Kopenmp -Nfjomplib
```

-Kopenmp_simd

-Kopenmp_simd option enables only the `simd` construct and the `declare simd` construct of OpenMP and compiles the source programs.

-Nfjomplib

This option specifies to use Fujitsu OpenMP Library.

-Nfjomplib option is required at linking.

Obtaining of Optimization Information

See Section "[4.3.1.2 Obtaining of Optimization Information](#)".

Restriction of OpenMP programs

See Section "[4.3.1.3 Restriction of OpenMP programs](#)".

J.3.2 Execution Process (OpenMP Parallelization)

The execution process of an OpenMP C program is the same as the procedure for a serial-processing program.

Environment Variable at Execution

Environment variable `FLIB_FASTOMP`

The user can control the high-speed runtime library using the environment variable `FLIB_FASTOMP`. Allowable values and their meanings are as follows. The default value is `TRUE`, so that the user can use the high-speed runtime library.

- `TRUE`

The high-speed runtime library is adopted instead of the normal runtime library (default).

- `FALSE`

The normal runtime library is adopted.

The high-speed runtime library achieves a fast execution time by partly limiting the OpenMP specification.

Details of the high-speed runtime library are shown below:

- It is well optimized on the assumption that nested parallelism is inhibited and every thread is connected to a CPU respectively.
- The hardware barrier can be used in the Job Operation Software environment.

To use the high-speed runtime library, OpenMP programs are restricted as follows:

- The number of threads specified with the `num_threads` clause or the `omp_set_num_threads` function must be the same as defined by the execution environment.

To check the number of threads defined by the execution environment, see "[Notes during Execution](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" (or see Section "[J.4 Using High-Speed Facility on Job Operation Software](#)" for the Job Operation Software environment).

When the high-speed runtime library is used, control of parallel execution with OpenMP environment variables and the OpenMP API is restricted as follows:

- Irrespective of the environment variable `OMP_NESTED` and the `omp_set_nested` function, nested `parallel` regions are always serialized.

- Irrespective of the environment variable `OMP_DYNAMIC` and the `omp_set_dynamic` function, the number of threads cannot exceed the limit on the number of CPUs.

The limit on the number of CPUs is described in "Notes during Execution" in Section "J.3.2 Execution Process (OpenMP Parallelization)" (or "J.4 Using High-Speed Facility on Job Operation Software" for the Job Operation Software environment).

Environment variable `FLIB_SPINWAIT` and `OMP_WAIT_POLICY`

The environment variables `FLIB_SPINWAIT` and `OMP_WAIT_POLICY` control the synchronization process.

`FLIB_SPINWAIT`

unlimited	Uses spin wait until all threads are synchronized. The default is unlimited.
0	Uses no spin wait but suspending wait.
<i>n</i> s	Uses spin wait until after <i>n</i> seconds, and then uses suspending wait. <i>n</i> is an integer value that must be positive. The unit "s" is specified following <i>n</i> .
<i>n</i> ms	Uses spin wait until after <i>n</i> milliseconds, and then uses suspending wait. <i>n</i> is an integer value that must be positive. The unit "ms" is specified following <i>n</i> .

Spin wait is a synchronization mode that allows all threads to consume CPU time, while suspending wait consumes less CPU time. Since spin wait is more efficient in parallel execution, "unlimited" is suitable for reducing elapsed time. On the other hand, "0" is suitable for reducing CPU time.

`OMP_WAIT_POLICY`

ACTIVE	Uses spin wait until all threads are synchronized. The default is ACTIVE.
PASSIVE	Uses no spin wait but suspending wait.

Select ACTIVE to give priority to the elapsed time. Select PASSIVE to give priority to the CPU time.

The settings of environment variable `FLIB_SPINWAIT` will be valid when environment variable `FLIB_SPINWAIT` is specified.

Environment variable `OMP_PROC_BIND`

A user can control the CPU binding for thread using the environment variable `OMP_PROC_BIND`.

Valid values and their meanings are as follows:

Value	Meaning
TRUE	The threads are bound to CPUs in order of cpuid list. However, cpuid which can actually be assigned becomes only within the limits of CPU affinity of the process at the start of execution.
FALSE	The thread is not bound to a CPU

The environment variable `FLIB_CPUBIND` or `FLIB_CPU_AFFINITY` is given to priority more than the environment variable `OMP_PROC_BIND`.

However, the environment variable `OMP_PROC_BIND` is effective in the following case.

- The environment variable `FLIB_CPUBIND=off` is specified, and the environment variable `FLIB_CPU_AFFINITY` is not specified.

See "[FLIB_CPUBIND](#)" in Section "J.4.2 CPU Binding" for the environment variable `FLIB_CPUBIND`.

See "Notes during Execution" in Section "J.3.2 Execution Process (OpenMP Parallelization)" for the environment variable `FLIB_CPU_AFFINITY`.

Environment variable `THREAD_STACK_SIZE`

The size of the thread stack can be set for the environment variable `THREAD_STACK_SIZE` in Kbytes.

When the environment variable `OMP_STACKSIZE` is specified, the greater value is used as the stack area size for each thread.

See "[Notes during Execution](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" for details.

Environment variable `OMP_STACKSIZE`

The size of the thread stack can be set for the environment variable `OMP_STACKSIZE` in Kbytes, Mbytes, or Gbytes.

When the environment variable `THREAD_STACK_SIZE` is specified, the greater value is used as the stack area size.

See "[Notes during Execution](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" for details.

Environment Variable for OpenMP Specifications

The user can set the following environment variables, which are in accordance with the OpenMP Specifications. For the details, refer to the OpenMP specifications.

Environment variable `OMP_SCHEDULE`

Sets the schedule type and chunk size for directives that have the schedule type at runtime.

Environment variable `OMP_NUM_THREADS`

Sets the number of threads to use during execution.

Environment variable `OMP_DYNAMIC`

Enables or disables dynamic thread adjustment.

Environment variable `OMP_PROC_BIND`

Specifies whether threads are bound to CPUs.

Environment variable `OMP_NESTED`

Enables or disables nested parallelism.

Environment variable `OMP_STACKSIZE`

Specifies the stack size for each thread.

Environment variable `OMP_WAIT_POLICY`

Specifies the behavior of the standby thread.

Environment variable `OMP_MAX_ACTIVE_LEVELS`

Specifies the maximum number of nested active parallel regions.

Environment variable `OMP_THREAD_LIMIT`

Specifies the maximum number of threads performed on an Open MP program.

Notes during Execution

When executing programs created on this system, note the following.

Variable allocation at execution

Programs created by this system allocate local variables and private variables in a function to the stack region.

If there is not enough space for the stack area to allocate local variables and private variables in the function, the stack area needs to be extended to an appropriate size.

When the stack area size for each thread needs to be specified, specify the environment variable `THREAD_STACK_SIZE` or `OMP_STACKSIZE`.

When the environment variable is not specified, the stack size of each thread is the same as the stack size of the process. However, if the maximum stack size for the process is larger than the value calculated by the expression " $\text{min}(\text{the size of available memory in this system, the size of a virtual memory}) / (\text{the number of threads})$ ", the system will determine the stack area size and allocate it as follows:

```
the stack size for each thread (byte) =
(min(the size of available memory in this system, the size of a virtual memory) / (the number of
threads)) / 5
```

However, the assumed value does not guarantee that the system works correctly. It is recommended that an appropriate value for maximum process stack size be specified manually. The maximum process stack area size can be specified using the bash built in command "ulimit".

Note that the maximum virtual memory size can also be confirmed using ulimit (bash built-in command).

To make the size of the stack area of each thread the same value, the function to change the number of threads dynamically is not considered. Therefore, the same number of threads for "FLIB_FASTOMP=TRUE" is used in this expression.

When the number of threads is dynamically changed, you should set the number of maximum threads to OMP_NUM_THREADS.

For more information, see ["Environment Variable for OpenMP Specifications"](#) in Section ["J.3.2 Execution Process \(OpenMP Parallelization\)"](#) for the environment variable THREAD_STACK_SIZE and OMP_STACKSIZE.

Limits on the Number of CPUs

One of the following values is defined as the limit on the number of CPUs.

For jobs	The number of CPUs that can be used with jobs
Not for job	The number of CPUs in the system

Number of Threads

When the high-speed runtime library is used, a common number of threads is used for parallel execution of both the OpenMP program and automatic parallelization, otherwise the numbers of threads are determined independently of each other. Use of the high-speed runtime library is controlled by setting the environment variable FLIB_FASTOMP (described in ["Environment Variable for OpenMP Specifications"](#) in Section ["J.3.2 Execution Process \(OpenMP Parallelization\)"](#)).

When the high-speed runtime library is used (FLIB_FASTOMP is TRUE):

The number of threads is determined with the following priority:

1. The value of environment variable OMP_NUM_THREADS
2. The value of environment variable PARALLEL
3. The number of CPUs that can be used with jobs
4. 1 thread

The number of threads may not exceed the limit on the number of CPUs. If this occurs, the number of threads becomes equal to the limit on the number of CPUs.

The numbers of threads specified in the program with the num_threads clauses of the parallel directives and the omp_set_num_threads function must adhere to this limitation. If a different number of threads is specified in the program, the execution is stopped with the following message:

```
jwe1041i-s The number of thread cannot be changed when the environment variable FLIB_FASTOMP is
TRUE.
```

When the high-speed runtime library is not used (FLIB_FASTOMP is FALSE):

The number of threads used for OpenMP code is determined with the following priority (while the one for automatic parallelization is described in Section ["J.2 Automatic Parallelization"](#)):

1. The value specified with the num_threads clause of the "parallel" directive
2. The value specified with the omp_set_num_threads function
3. The value of the environment variable OMP_NUM_THREADS
4. The value of the environment variable PARALLEL
5. The number of CPUs that can be used with jobs

6. 1 thread

When the dynamic thread adjustment feature is enabled, the actual number of threads is the minimum number of threads determined by the rule above and the limits on the number of CPUs.

When the dynamic thread adjustment feature is disabled, multiple threads can be executed on the same processor via time-sharing. Such parallel execution is not effective because the overhead of synchronization among threads is very high. Therefore, it is recommended to set the number of threads to be less than the limit on the number of CPUs.

For the execution processing on the job, see Section "[J.4.1 Management of CPU Resources](#)".

Restriction of `fork(2)`

Functions that depend on `fork(2)` are restricted.

CPU binding for thread

CPU binding for thread differs between job execution and non-job execution.

When executing program on non-job

The thread is not bound to a CPU. But you can control CPU binding for thread using environment variable `FLIB_CPU_AFFINITY`.

Environment variable `FLIB_CPU_AFFINITY`

Threads are bound to CPUs in order of the specified cpuid list.

When the number of specified CPUs is exceeded, it is repeatedly used from the beginning of the list.

The cpuid list shall be separated by comma (',') or space (' ') .

The cpuid list can have the next form that has range with increment.

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: cpuid of the beginning of the range. ($0 \leq \text{cpuid1} < \text{CPU_SETSIZE}$)
cpuid2: cpuid of the end of the range. ($0 \leq \text{cpuid2} < \text{CPU_SETSIZE}$)
inc: increment ($1 \leq \text{inc} < \text{CPU_SETSIZE}$)

In addition, it is necessary to be the following.

```
cpuid1 ≤ cpuid2
```

It becomes equivalent to the case where all CPUs for every increment value *inc* in the range from *cpuid1* to *cpuid2* are specified.

The cpuid can be used the above-mentioned value.

However, cpuid which can actually be assigned becomes only within the limits of CPU affinity of the process at the start of execution.

See `CPU_SET(3)` about details of `CPU_SETSIZE`.



Note

If cpuid is outside the CPU affinity of the process at the start of execution, the program outputs error messages. Correct the setting value.



Example

Example 1:

```
$ export FLIB_CPU_AFFINITY="12,14,13,15"
```

The thread is bound to CPU in order of 12, 14, 13, and 15.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 2:

```
$ export FLIB_CPU_AFFINITY="12-19"
```

The thread is bound to CPU in order of 12, 13, 14, 15, 16, 17, 18, and 19.

When the number of threads is nine or more, it is repeatedly used from the beginning of the list.

Example 3:

```
$ export FLIB_CPU_AFFINITY="12-19:2"
```

The thread is bound to CPU in order of 12, 14, 16, and 18.

When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 4:

```
$ export FLIB_CPU_AFFINITY="12-16:2,13,19"
```

The thread is bound to CPU in order of 12, 14, 16, 13, and 19.

When the number of threads is six or more, it is repeatedly used from the beginning of the list.

.....

When executing program on job

When thread is not bound to CPU, you may control CPU binding with `FLIB_CPU_AFFINITY` as executing on non-job.

When thread is bound to CPU on job, the environment variable `FLIB_CPU_AFFINITY` becomes invalid.

For details of CPU binding, see Section "[J.4.2 CPU Binding](#)".

Output from More than One Threads

When errors are detected from more than one thread within a program, traceback maps are output in thread units. Therefore, in the `parallel` region, the information is displayed at the same number as threads.

J.3.3 Implementation-Dependent Specifications

The OpenMP specification includes the following processor-dependent specification. For details on each item, refer to the OpenMP Specifications.

Memory Model

If a variable is longer than 4 bytes or crosses a 4-byte boundary:

- The value of the variable written from two threads without synchronization may be undefined.
- The value read by a thread from a variable that is written from another thread without synchronization may be undefined.

Internal Control Variables

The initial values for each of the internal control variables are as follows:

Internal Control Variable	Initial Value
<code>nthreads-var</code>	1
<code>dyn-var</code>	TRUE
<code>run-sched-var</code>	static without chunk size
<code>def-sched-var</code>	static without chunk size
<code>bind-var</code>	FALSE
<code>stacksize-var</code>	the stack area size for threads in " Notes during Execution " in Section " J.3.2 Execution Process (OpenMP Parallelization) "

Internal Control Variable	Initial Value
wait-policy-var	ACTIVE
thread-limit-var	2147483647
max-active-levels-var	2147483647

Dynamic Thread Adjustment Features

This system supports the dynamic thread adjustment described in Section "[Notes during Execution](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)".

By default, the dynamic adjustment features are on.

Loop Statement

The type used to calculate the iteration count of a loop that is not nested is `long int`.

When `auto` is set for the internal control variable `run-sched-var`, the effect of the `schedule(runtime)` construct is set to `schedule(static)`.

sections Construct

Assignment to a thread of a structured block in a `sections` construct is performed in the same way as a `dynamic schedule`.

Each thread that encounters the `sections` region or that finishes the execution of a structured block in the `sections` region is moved on the next structured block.

single Construct

A `single` region is executed by the thread that encounters the region first.

simd Construct

The integer type used to compute the iteration count of a collapsed loop is `long int`.

SIMD length is the value to which the compiler decides automatically.

When the `alignment` parameter is not specified in the `aligned` clause, it is considered that the following value was specified for the `alignment` parameter.

When -KSVE option is effective at the compile time	Alignment of the type of the list item
When -KNOSVE option is effective at the compile time	16

declare simd Construct

SIMD length when the `simdlen` clause is not specified is as follows.

- When -KSVE option is effective at the compile time

SIMD length is decided at the run time.

- When -KNOSVE option is effective at the compile time

SIMD length is decided by the size of the minimum type among all input parameters and return value.

SIMD length is as follows by the size of the type.

Size of the type (byte)	SIMD length
1	16, 8
2	8, 4
4	4, 2
8	2
16	

When the *alignment* parameter is not specified in the *aligned* clause, it is considered that the following value was specified for the *alignment* parameter.

When -KSVE option is effective at the compile time	Alignment of the type of the list item
When -KNOSVE option is effective at the compile time	16

Task scheduling points

In untied task regions, task scheduling points are located at the same positions as in the case of tied task regions: only in `task`, `taskwait`, explicit or implicit `barrier` constructs, and at the completion point of the task.

atomic Construct

Two `atomic` regions will be executed independently (not exclusively), when a variable type to be updated is different. When the types match, it may be executed exclusively even if the address is different.

- When updating a logical type, complex type, 1 byte integer type, 2 byte integer type, or quadruple precision (16 byte) real type variable
- When updating array element and index expression is not the same
- The target expression has explicit or implicit type conversion

omp_set_num_threads Function

Calling the `omp_set_num_threads` function has no effect when the argument value is equal to or less than 0. A value that exceeds the number of threads supported by the system must not be specified.

omp_set_schedule Function

There are no schedule types that are implementation-dependent.

omp_set_max_active_levels Function

A call to the `omp_set_max_active_levels` function will be ignored when it is performed from an explicit `parallel` region. It will also be ignored when the argument is an integer that less than 0.

omp_get_max_active_levels Function

The `omp_get_max_active_levels` function can be called from anywhere in the program and it returns the value of the internal control variable `max-active-levels-var`.

Environment Variable OMP_SCHEDULE

When the schedule type specified for `OMP_SCHEDULE` is invalid, the schedule type is ignored, and the default value (`static` without chunk size) is used.

When the schedule type specified for `OMP_SCHEDULE` is `static`, `dynamic`, or `guided`, and the chunk size is not a positive number, the chunk size will be as follows:

Schedule Type	Chunk Size
<code>static</code>	no chunk size
<code>dynamic</code>	1
<code>guided</code>	

Environment Variable OMP_NUM_THREADS

When a value equal to or less than 0 is specified for the list of `OMP_NUM_THREADS`, it works in the same ways as when 1 is specified. A value that exceeds the number of threads supported by the system must not be specified.

Environment Variable OMP_PROC_BIND

When a value other than `TRUE` or `FALSE` is specified for `OMP_PROC_BIND`, the value is ignored, and the default value (`FALSE`) is used.

Environment Variable OMP_DYNAMIC

When a value other than TRUE or FALSE is specified for OMP_DYNAMIC, the value is ignored and the default value (TRUE) is used.

Environment Variable OMP_NESTED

When a value other than TRUE or FALSE is specified for OMP_NESTED, it is ignored and the default value (FALSE) is used.

Environment Variable OMP_STACKSIZE

When the value specified for OMP_STACKSIZE does not meet the defined format, it is ignored and the default value is used.

Environment Variable OMP_WAIT_POLICY

ACTIVE performs spin wait. PASSIVE performs suspend wait.

Environment Variable OMP_MAX_ACTIVE_LEVELS

When the value specified for OMP_MAX_ACTIVE_LEVELS is an integer that is less than 0, it is ignored and the default value (2147483647) is used.

Environment Variable OMP_THREAD_LIMIT

When the value specified for OMP_THREAD_LIMIT is not a positive integer, it is ignored and the default value (2147483647) is used.

J.3.4 Notes on OpenMP Programming

This section provides notes on OpenMP C programming.

Implementation of parallel Region and Explicit task Region

The structured block within a parallel construct or a task construct is compiled as an internal function.

The name of the internal function generated from the parallel construct or the task construct is different depending on the compiler mode. The name of an internal function is shown below.

Type of internal function	Internal function name
Internal function generated from parallel construct	The name " <code>._OMP_id-numberA</code> " is added
Internal function generated from task construct	The name " <code>._TSK_id-numberB</code> " is added

id-numberA: Unique number that identifies parallel construct in a source file.

id-numberB: Unique number that identifies task construct in a source file.

Implementation of threadprivate Variable

See Section "[4.3.4.2 Implementation of threadprivate Variable](#)".

Automatic Parallelization for OpenMP Programs

See Section "[4.3.4.3 Automatic Parallelization for OpenMP Programs](#)".

J.3.5 Linking with Other Multi-Thread Programs

The following restrictions apply when linking with other multi-thread programs.

Automatic Parallelization Program on this System

An object program of an automatic parallelization program created by this system can be linked with an object program created by this system.

Other Multi-thread Programs

An object program created by this system cannot be linked with an object program created by another multi-thread system.

However, the program may be linked with a pthread program when using the environment variable FLIB_PTHREAD.

Environment Variable `FLIB_PTHREAD`

You can control the linking with a pthread program, using the environment variable `FLIB_PTHREAD`.

Valid values are as follows. Default value is 0.

Value	Explanation
0 (Default)	<p>The program can be linked with a pthread program which uses the pthread thread as a control thread.</p> <p>However, there are the following restrictions.</p> <ul style="list-style-type: none">- The pthread thread must use suspending wait.- The pthread program must not be compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option. (*)- The routine which is compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option must not be used in the pthread program. (*)- The Fortran routine must not be used in the pthread program. (*)
1	<p>The program can be linked with a pthread program which executes parallel processing.</p> <p>And, the program can be linked with a pthread program which uses the pthread thread as a control thread.</p> <p>However, there are the following restrictions.</p> <ul style="list-style-type: none">- A pthread parallel processing, and an OpenMP or automatic parallel processing must be executed in order. (*) An OpenMP or automatic parallel processing must not be executed in a pthread parallel processing. And, a pthread parallel processing must not be executed in OpenMP or automatic parallel processing.- The pthread thread must use suspending wait.- The pthread program must not be compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option. (*)- The routine which is compiled with <code>-Kopenmp</code> option or <code>-Kparallel</code> option must not be used in the pthread program. (*)- The Fortran routine must not be used in the pthread program. (*)- Fujitsu's math libraries must not be used in the pthread program. (The pthread program must not be compiled with <code>-SSL2BLAMP</code> option.) (*)- You must execute OpenMP parallel processing with two or more threads, before creating pthread threads. And, this number of OpenMP threads cannot be changed later. <p>And, when this function is used, the following functions are effective.</p> <ul style="list-style-type: none">- <code>FLIB_SPINWAIT=0</code>- <code>FLIB_CPUBIND=off</code> <p>Operation is not guaranteed when a value which is different in the above-mentioned environment variable is specified.</p>

*) Operation is not guaranteed when the restriction function is used.

J.3.6 Debugging OpenMP Programs

See Section "[Chapter 8 Debugging Functions](#)" for the debug feature in this system.

The following restrictions are applied to the debugging of the OpenMP programs:

- A function generated by the `declare simd` construct cannot be debugged using the debugger.

J.4 Using High-Speed Facility on Job Operation Software

On the job of Job Operation Software, the high-speed facility (the inter-core hardware barrier and the Sector cache) can be used.

This section provides information on the compilation and execution processes for using the high-speed facility on FX system.

J.4.1 Management of CPU Resources

The CPUs resources that can be used are strictly managed on the job of the Job Operation Software.

- The Number of CPUs that can be used on the job

This is the number of CPUs that can be used for a process on the job.

See "[Number of CPUs that can be used on the job](#)" in Section "[J.4.1 Management of CPU Resources](#)" for details.

- Limits on Number of CPUs

The number of CPUs that can be used on the job is defined as the limit on the number of CPUs.

The number of threads when the program is executed is depends on the limit values of the number of CPUs. See "[Number of Threads](#)" in Section "[J.2.2 Execution Process \(Automatic Parallelization\)](#)" for details on the number of the threads for automatic parallelization. See "Number of Threads" in "[Notes during Execution](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" for details on the number of threads for OpenMP.

Number of CPUs that can be used on the job

The limit on the number of CPUs for one process is determined by considering CPU resources and the number of processes in the virtual node on the job. This value is defined as the number of CPUs that can be used on the job. Details are shown as follows:

"The number of CPUs that can be used on job" = $cpunum_on_node / procnum_on_node$

cpunum_on_node: The number of CPUs that can be used on one virtual node.
1- the number of all CPUs on one virtual node.

procnum_on_node: The number of processes on one virtual node.
For thread parallel processing jobs, it is 1.
For hybrid (process and thread) parallel processing jobs, it is 1- the number of all CPUs on one virtual node.

It is rounded down when it cannot be divided.

When MPI program is executed with VCOORD file that is set the number of CPUs to a process, the value is the number of CPUs that can be used on the job. See the "MPI User's Guide" for details on MPI.

FLIB_USE_ALLCPU

"The number of CPUs that can be used on the job" can be controlled using the environment variable `FLIB_USE_ALLCPU`.

The allowed values their meanings are as follows. The default value is "FALSE".

- TRUE

The number of CPUs that can be used on the job is *cpunum_on_node*. This means all CPUs that can be used on the virtual node are used regardless of the number of processes. Therefore, when the number of processes on the virtual node is two or more, these CPUs resource is shared among the processes.

When the frequency calculated at the same time in each process is low etc., an improvement in execution performance can be expected. In general use, an improvement in execution performance cannot be expected.

You should use the following functions.

- The `-Kopenmp` option is used at compiling and linking time.

- The environment variable `FLIB_FASTOMP=FALSE` is defined at run time.
- The environment variable `FLIB_SPINWAIT=0` is defined at run time.

When the above-mentioned is not used, the performance might be greatly downed like the dead lock. See "[J.3 Parallelization by OpenMP Specification](#)" for details of -Kopenmp option, environment variables `FLIB_FASTOMP` and `FLIB_SPINWAIT`.

When `procnum_on_node` is 1, "TRUE" and "FALSE" have the same effect. When `procnum_on_node` is not 1, "[J.4.3 Inter-Core Hardware Barrier](#)" and "[J.4.4 Sector Cache](#)" cannot be used. See "[Number of CPUs that can be used on the job](#)" in Section "[J.4.1 Management of CPU Resources](#)" for details about `cpunum_on_node` and `procnum_on_node`.

- FALSE

The number of CPUs that can be used is "[Number of CPUs that can be used on the job](#)" in Section "[J.4.1 Management of CPU Resources](#)".

When MPI program is executed with VCOORD file that is set the number of CPUs to a process, the value is ignored. See the "MPI User's Guide" for details on MPI.

FLIB_USE_CPURESOURCE

The environment variable `FLIB_USE_CPURESOURCE` controls the CPU resource manager on the job.

The allowed values their meanings are as follows. The default value is "TRUE".

- TRUE

The CPU resources on the job are managed.

- FALSE

The CPU resources on the job are not managed. Therefore the number of CPUs is not managed, and you cannot use the following functions.

- "[J.4.2 CPU Binding](#)"
- "[J.4.3 Inter-Core Hardware Barrier](#)"
- "[J.4.4 Sector Cache](#)"

J.4.2 CPU Binding

The thread is bound to one CPU when the program using the automatic parallelization or OpenMP is executed on the job. CPUs resources can be strictly managed as described in Section "[J.4.1 Management of CPU Resources](#)". Therefore an improvement on the performance can be expected using CPU binding.

See "[J.2 Automatic Parallelization](#)" and "[J.3 Parallelization by OpenMP Specification](#)" for details.

FLIB_CPUBIND

The environment variable `FLIB_CPUBIND` controls the CPU binding for threads.

The following value can be set to `FLIB_CPUBIND`. The default value is "chip_pack".

chip_pack	All threads are bound to the same CPU Chip as much as possible.
chip_unpack	Each thread is bound to a different CPU Chip as much as possible.
off	The thread is not bound to the CPU. If you control CPU binding by yourself, this must be set.

When only one CPU Chip can be used, "chip_pack" and "chip_unpack" have the same effect.

When only one CPU on a CPU Chip can be used, "chip_pack" and "chip_unpack" have the same effect.

J.4.3 Inter-Core Hardware Barrier

FX system has the inter-core hardware barrier. On the job, it can be used as the thread barrier. The inter-core hardware barrier is a hardware mechanism which facilitates high speed synchronization among threads in a CPU Chip, and raises the execution performance.

See manuals of Job Operation Software for details on Job Operation Software about how to execute and submit a job.

Compilation

When generating a program using the inter-core hardware barrier as the thread barrier, specify `-Kparallel` or `-Kopenmp` option.

The hardware barrier is automatically used during execution in an environment in which it can be used. By contrast, the software barrier is used during execution in an environment that cannot use the hardware barrier.

Execution

The inter-core hardware barrier can be used only on the job. The inter-core hardware barrier is a hardware mechanism for synchronization among threads in a CPU Chip.

`FLIB_CNTL_BARRIER_ERR`

When the inter-core hardware barrier cannot be used, the following message is output, and the processing is continued using the software barrier.

```
jwe1050i-w The hardware barrier couldn't be used and continues processing using the software barrier.
```

This diagnostic message (jwe1050i-w) error can be controlled using the environment variable `FLIB_CNTL_BARRIER_ERR`.

The allowed values and their meanings are as follows. Default is "TRUE".

- TRUE

The diagnostic message (jwe1050i-w) error is detected.

When the inter-core hardware barrier cannot be used, this message is output, and the processing is continued using the software barrier.

- FALSE

The diagnostic message (jwe1050i-w) error is not detected.

When the inter-core hardware barrier cannot be used, the processing is continued using the software barrier.

`FLIB_NOHARDBARRIER`

When the environment variable `FLIB_NOHARDBARRIER` is set, the inter-core hardware barrier is not used and the software barrier is used. Always outputs the diagnostic message jwe1050i-w.

See "[FLIB_CNTL_BARRIER_ERR](#)" in Section "[J.4.3 Inter-Core Hardware Barrier](#)" for details.



Note

- When the inter-core hardware barrier is used by a program made using the compile option `-Kopenmp`, the environment variable `FLIB_FASTOMP` shall not be FALSE. See Section "[J.3 Parallelization by OpenMP Specification](#)" for the meaning and use of the environment variable `FLIB_FASTOMP`.
- When the inter-core hardware barrier is used, CPU binding is needed. Therefore, when "off" is set for the environment variable `FLIB_CPUBIND`, the inter-core hardware barrier cannot be used. See Section "[J.4.2 CPU Binding](#)" for details of CPU binding.
- When the number of threads is 1, the inter-core hardware barrier cannot be used. See "[Number of Threads](#)" in Section "[J.2.2 Execution Process \(Automatic Parallelization\)](#)" or "[Notes during Execution](#)" in Section "[J.3.2 Execution Process \(OpenMP Parallelization\)](#)" for the number of threads.
- When the number of CPUs to a process is three or less, an inter-core hardware barrier may not be able to use. The number of CPUs to a process should be set to four or more.
- When the job is a Swap Target Job and a Node-sharing Job, and the number of CPUs to a process is three or less, an inter-core hardware barrier cannot be used. The number of CPUs to a process should be set to four or more.
- The inter-core hardware barrier in FX system is used only inside the NUMA node. When one process is allocated to multiple NUMA nodes, inter-core hardware barrier is used inside NUMA node and software barrier is used between NUMA nodes. Therefore, it is recommended to allocate one process to one NUMA node, in case of giving priority to the performance of threads barrier.

- On the following cases, the high speed runtime library cannot determine whether one process can reserve a barrier resource or not. Therefore behavior of hardware barrier is indeterminate, program may be ended with diagnostic-message (jwe1044i-u or jwe1045i-u), or the program may terminate abnormally.

- When Process is generated by functions other than Fujitsu MPI

Example:

- FORTRAN FORK/SYSTEM/SH service function is used
- C fork system call/system function is used



Example

Example 1: A program that uses the inter-core hardware barrier is executed on a thread parallel processing job.

```
$ cat job.sh
#!/bin/sh
./a.out

$ pjsub -L "node=1" job.sh
```

Example 2: A program that uses the inter-core hardware barrier is executed on a hybrid (process and thread) parallel processing job (One job is executed in one dimension: The number of virtual nodes is set to the number of processes).

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=12"
#PJM --mpi "shape=12"
#PJM --mpi "rank-map-bynode"
mpiexec -n 12 a.out

$ pjsub job.sh
```

J.4.4 Sector Cache

FX system has Sector cache. On the job, the Sector cache can be controlled.

See Section "3.5 Software Control of Sector Cache" for details on how to control the Sector cache.

This section explains only notes on execution.

Notes on execution

The Sector cache for the primary and secondary level caches are available in the following execution environments.

The execution environments	The Sector cache for the first level cache	The Sector cache for the second level cache
Only one process runs on a NUMA node	Available	Available
Multiple processes run on a NUMA node	Available (*1)	Not available

*1) Not available if the environment variable FLIB_L1_SCCR_CNTL is assigned to the value FALSE. See Section "3.5.2.2 Software Control with Environment Variables and Optimization Control Line" for details.

On the following cases, the high speed runtime library cannot determine whether one process can reserve a NUMA node or not. Therefore behavior of Sector cache is indeterminate. Performance of program may be reduced, program may be ended with diagnostic-message (jwe1048i-u) , or the program may terminate abnormally.

- Process is generated by functions other than Fujitsu MPI.

Example:

- FORTRAN FORK/SYSTEM/SH service function is used.
- C fork system call/system function is used.
- Threads are generated by functions other than Fujitsu Automatic Parallelization/OpenMP.

Example:

- Threads are controlled by pthread functions.

To invalidate the Sector cache control function unconditionally regardless of the execution environment, "FALSE" is set for the environment variable FLIB_SCCR_CNTL.

See Section "[3.5.2.2 Software Control with Environment Variables and Optimization Control Line](#)" for details.