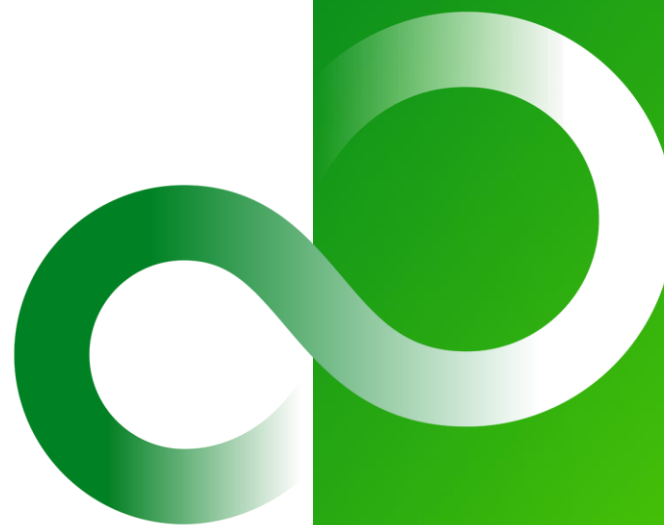


# Programming Guide (Fortran)

Mar. 2023

v1.6

FUJITSU LIMITED



- This document describes how to program and tune applications for A64FX processors in Fortran.
- Refer to the following in conjunction with this document.
  - Fortran User's Guide
  - C User's Guide
  - C++ User's Guide
  - Profiler User's Guide
  - Programming Guide(Processors)
  - Programming Guide(Programming common part)
  - Programming Guide(Tuning)
- The following abbreviation is used in this document:
  - A64FX Logic Specifications
  - A64FX ® Microarchitecture Manual
  - ARM® Architecture Reference Manual (ARMv8 , ARMv8.1 , ARMv8.2 , ARMv8.3)
  - ARM® Architecture Reference Manual Supplement The Scalable Vector Extension
- Trademarks
  - Linux® is a trademark or registered trademark of Linus Torvalds in the United States and other countries.
  - Red Hat is a trademark or registered trademark of Red Hat Inc. in the United States and other countries.
  - ARM is a trademark or registered trademark of ARM Ltd. in the United States and other countries.
  - Proper names such as the product name mentioned are trademark or registered trademark of each company.
  - Trademark symbols such as ® and ™ may be omitted from system names and product names in this document.

## ■ Major Optimization Settings

### ■ Recommended Option

- [-Kfast](#)

### ■ Optimization Level

- [-O](#)

### ■ SIMDization

- [-Ksimd](#)  
[Directive: SIMD](#)
- [-Ksimd\\_packed\\_promotion](#)
- [-Ksimd\\_reg\\_size={128|256|512}](#)
- [-Ksimd\\_reg\\_size=agnostic](#)
- [-Ksimd\\_use\\_multiple\\_structures](#)  
[Directive: USE MULTIPLE STRUCTURES](#)
- [-Ksimd\\_uncounted\\_loop](#)
- [-Kloop\\_part\\_simd](#)
- [Directive: NOVREC/NORECURRENCE](#)

### ■ Software Pipelining

- [-Kswp](#)  
[Directive: SWP](#)
- [-Kswp\\_weak/-Kswp\\_strong](#)  
[Directive: SWP\\_WEAK](#)
- [-Kswp\\_freq\\_rate/-Kswp\\_ireg\\_rate/](#)  
[-Kswp\\_preg\\_rate](#)

- [-Kswp\\_policy](#)  
[Directive: SWP\\_POLICY](#)

### ■ Instruction Scheduling

- [-Ksch\\_pre\\_ra/-Ksch\\_post\\_ra](#)

### ■ Prefetch

- [-Kprefetch\\_sequential](#)  
[Directive: PREFETCH\\_SEQUENTIAL](#)
- [-Kprefetch\\_cache\\_level](#)  
[Directive: PREFETCH\\_CACHE\\_LEVEL](#)
- [-Kprefetch\\_stride](#)  
[Directive: PREFETCH\\_STRIDE](#)
- [-Kprefetch\\_strong/-Kprefetch\\_strong\\_L2](#)  
[Directive: PREFETCH\\_STRONG/](#)  
[Directive: PREFETCH\\_STRONG\\_L2](#)
- [-Kprefetch\\_conditional](#)  
[Directive: PREFETCH\\_CONDITIONAL](#)
- [Directive: PREFETCH\\_INDIRECT](#)
- [Directive:](#)  
[PREFETCH\\_LINE/PREFETCH\\_LINE\\_L2](#)
- [Directive:](#)  
[PREFETCH\\_READ/PREFETCH\\_WRITE](#)

- Loop Expansion
  - -Kunroll  
Directive: UNROLL
  - Directive: FULLUNROLL PRE SIMD
  - -Kunroll and jam  
Directive: UNROLL AND JAM[(n)]  
Directive: UNROLL AND JAM FORCE
  - -Kstriping  
Directive: STRIPING
- Loop Fission
  - -Kloop fission  
Directive: LOOP FISSION TARGET  
Directive: FISSION POINT
  - -Kloop fission threshold  
Directive: LOOP FISSION THRESHOLD
  - -Kloop fission stripmining  
Directive:  
LOOP FISSION STRIPMINING
- Loop Fusion
  - -Kloop fusion  
Directive: LOOP NOFUSION
- Loop Interchange
  - -Kloop interchange  
Directive: LOOP INTERCHANGE
- Loop Unswitching
  - Directive: UNSWITCHING
- Perfect Loop nesting
  - -Kloop perfect nest  
Directive: LOOP PERFECT NEST
- Fast Store
  - -Kzfill
- Clone Optimization
  - Directive: CLONE
- Speculative Execution of Load Instructions
  - -Kpreload  
Directive: PRELOAD
- Stack Allocation
  - -Kauto/-Kautoobjstack/-Ktemparraystack
- Change in Operation Evaluation Method
  - -Keval  
Directive: EVAL
  - -Keval concurrent  
Directive: EVAL CONCURRENT
- Handling of Denormal Numbers
  - -Kfz

- Alignment of Common Block Boundary
  - [-Kalign commons](#)
- Array Declaration Range
  - [-Karray declaration opt](#)  
[Directive: ARRAY DECLARATION OPT](#)
- String Operation Library
  - [-Koptlib string](#)
- Sector Cache Specification
  - [Directive: SCACHE ISOLATE WAY](#)
  - [Directive: SCACHE ISOLATE ASSIGN](#)
- Link Time Optimization
  - [-Klto](#)
- HPC Tag Address Override
  - [-Khpctag](#)
- Data Allocation
  - [-Nreordered variable stack](#)
- Analysis and Inspection
  - [-Ncoverage](#)
- OpenMP
  - [-Kopenmp collapse except innermost](#)
  - [-Nfjomplib/-Nlibomp](#)
- [Adjustment of Optimization](#)
  - [-Kassume=shortloop](#)  
[Directive: ASSUME\(SHORTLOOP\)](#)
  - [-Kassume=memory bandwidth](#)  
[Directive: ASSUME\(MEMORY BANDWIDTH\)](#)
  - [-Kassume=time saving compilation](#)  
[Directive: ASSUME\(TIME SAVING COMPILATION\)](#)
- [Options That Require Attentions to Use \(Side Effects of Optimization\)](#)
  - [Options That May Affect Execution](#)
    - [-Kpreex](#)
    - [-Ksimd=2](#)
    - [-Kpreload](#)
  - [Options That Accompany Calculation Errors](#)
    - [-Keval](#)
    - [-Kfp contract](#)
    - [-Kfp relaxed](#)
    - [-Kilfunc](#)
  - [Options That Suppress Calculation Errors](#)
    - [-Kfp precision](#)
    - [-Kparallel fp precision](#)

## ■ Compilation Information

- [Diagnosis Message/Guidance Message](#)
- [Contents of Compilation Information](#)
- [Lister \(Program List/Optimization Information/Statistical Information\)](#)
- [Notes of Compilation Information](#)

## ■ Executing a Program

- [Execution Commands](#)
- [Thread Parallel Execution](#)
- [Combination of OpenMP Libraries](#)
- [LLVM OpenMP Library and Fujitsu OpenMP Library](#)
  - [LLVM OpenMP Library \(-Nlibomp\)](#)
  - [Fujitsu OpenMP Library \(-Nfjomplib\)](#)

## ■ Notes of Developing Programs

- [Conditions for SIMDizable Loops](#)
- [Conditions for Automatic Parallelizable Loops](#)
- [Fujitsu Extension Object Specification](#)
- [SIMDization of the Redirection Operations Using the bit Operator](#)
- [Large Page](#)
- [Notes on transformational function reference of intrinsic functions](#)

# Contents(5/5): Fortran-Local Information and Performance Tuning Examples

- [Timers Supported by Fortran](#)
  - [Specifications of Timers](#)
  - [Precisions of Timers](#)
- [Relation of Fortran Data Attributes and Optimization](#)
  - [Attribute List](#)
  - [allocatable Attribute](#)
  - [pointer Attribute](#)
  - [contiguous Attribute](#)
  - [intent\(in\) Attribute](#)
  - [intent\(out\) Attribute](#)
  - [intent\(inout\) Attribute](#)
  - [value Attribute](#)
  - [pure Attribute](#)
  - [save Attribute](#)
  - [Other Grammatical Notes](#)
    - [Case where an array not in the program is generated](#)
- [Examples of Performance Tuning by pointer and Argument Interface Improvement](#)
  - [Difference Between pointer and allocatable](#)
  - [Performance Tuning by pointer](#)
  - [Performance Tuning by Argument Interface Improvement](#)

# Major Optimization Settings

- Recommended Option
- Optimization Level
- SIMDization
- Software Pipelining
- Instruction Scheduling
- Prefetch
- Loop Expansion
- Loop Fission
- Loop Fusion
- Loop Interchange
- Loop Unswitching
- Perfect Loop Nesting
- Fast Store
- Clone Optimization
- Speculative Execution of Load Instructions
- Stack Allocation
- Change in Operation Evaluation Method
- Handling of Denormal Numbers
- Alignment of Common Block Boundary
- Array Declaration Range
- String Operation Library
- Sector Cache Specification
- Link Time Optimization
- HPC Tag Address Override
- Data Allocation
- Analysis and Check
- OpenMP
- Adjustment of Optimization



Option Name	Default
-Kfast	-

## ● Function outline

- This option induces the optimization option to create an object program that operates at high-speed on the target machine.

## ● Effect

- This option induces to specify standard optimization options and is expected to improve execution performance.

## ● Points

- The -Kfast option is recommended to be specified as an optimization option when execution performance is pursued.
- This option induces to specify standard optimization options that possibly improve execution performance.

## ● Note

- This option induces an optimization option that may generate operation precision errors. So, it is necessary to suppress options that generate operation precision errors if operation precision matters.

- Optimization options to be induced

Induced Option	Function Outline
-O3	Performs optimization in optimization level 3.
-Keval *	Performs optimization to change the operation evaluation method.
-Kfp_contract *	Performs optimization using the Floating-Point Multiply-Add/Subtract operation instruction.
-Kfp_relaxed *	Performs optimization applying reciprocal approximation operation to single precision floating-point division/double precision floating-point division/SQRT function.
-Kfz *	Uses the flush-to-zero mode (where a denormal operation result or source operand is replaced by 0 with the sign of the original number).
-Kilfunc *	Performs inline expansion of an intrinsic function/operation.
-Kmfunc *	Performs optimization to convert an intrinsic function/operation to a multi-operation function.
-Komitfp	Performs optimization in a way that does not ensure the frame pointer register at procedure calling. Therefore, the traceback information is not ensured.
-Ksimd_packed_promotion	Promotes packed-SIMDization supposing the index calculation of single precision real type/4-byte integer type array elements does not exceed the 4-byte range. A runtime error or execution result error may occur if it exceeds the 4-byte range.

\*: Optimization options where an operation precision error may occur.

Option Name	Default
-O[0 1 2 3]	Without -O specification: -O2 With -O specification only: -O3

## ● Function outline

- This option selects optimization according to the usage.

Optimization Level	Function Outline	Usage
0	Does not perform optimization.	When error check on a program is required
1	Performs basic optimization. This does not perform optimization to improve execution performance such as loop unrolling, software pipelining, and SIMDization. The object size becomes smaller and the execution time becomes shorter than -O0.	To avoid object size being large or to shorten compilation time
2	Performs loop optimization, prefetch instruction generation, SIMD, top alignment adjustment of loops and tail call optimization in addition to -O1. Also, this performs optimization of -O1 iteratively until the room for optimization disappears. Compilation takes longer than -O1.	When performance is pursued as far as the accuracy error is not generated (because -Kfast specification generates accuracy error)
3	Performs unrolling of nested loops, loop fission to promote loop interchange, loop unswitching, and CLONE optimization in addition to -O2. Compilation takes longer than -O2.	When execution performance is pursued

## ● Effect

- Optimization level can be used to select optimization according to the usage.

## ● Points

- When the recommended option -Kfast is specified, option -O3 is induced.
- The -Kfast option is recommended to be specified as the optimization option, but the optimization level can be specified according to the usage.

## ● Notes

- The side effects of optimization such as operation accuracy error may be generated with -O2 or above.
- Compilation takes longer with -O2 or above.

Option Name	Default
-Ksimd[={1 2 auto}] -Knosimd	-O2 or above: -Ksimd -O1 or below: -Knosimd
Optimization Specifier	
SIMD[({ALIGNED   UNALIGNED})] NOSIMD	

## ● Function outline

- This option specifies whether instructions in the loop is SIMDized.

Argument	SIMDization Specification
1	• Issues SIMD instructions.
2	• In addition to the function of -Ksimd=1, issues SIMD instructions to loops that contain IF constructs, etc.
auto (default)	• The compiler judges automatically whether to perform SIMDization.

- To apply this to a specific loop, use the SIMD specifier.

Argument	SIMDization Target
None (default)	• Instructs to perform SIMDization.
ALIGNED	• Equivalent to !OCL SIMD
UNALIGNED	• Equivalent to !OCL SIMD

- The argument of the SIMD specifier can be specified to maintain compatibility with source codes of old products.
- SIMDization may not be performed according to the operation type and the loop structure.

## ● Effect

- Converting two or more operations of the same kind into simultaneously executed SIMD instructions improves the execution performance.

## ● Point

- Specify the `-Ksimd={2|auto}` option to perform SIMDization on a loop of high cost and with any IF constructs that meet any of the following conditions.
  - The true rate of a condition clause in the IF construct is high.
  - The true rate of a condition clause in the IF construct is unknown but the number of executable statements in the THEN/ELSE clauses in the IF construct is smaller than the number of executable statements in the whole loop.

Eliminating branch instructions in the loop promotes software pipelining, and thus improves parallelism in instruction level.

## ● Note

- With the `-Ksimd={2|auto}` option, instructions in the IF construct are executed redundantly. So, execution performance may degrade depending on the true rate of the IF construct. Also, expressions in the IF construct are speculatively executed just like the `-Kpreex` option. So, instructions that should not be executed according to the program logic are executed, resulting in an error.

## ● Example

### ● Incorrect SIMD(ALIGNED) specification

Double precision floating-point store becomes available for SIMDization when it is on a 16-byte boundary. However, when the SIMD(ALIGNED) specifier is specified for double precision floating-point store that is not on a 16-byte boundary, it terminates abnormally with SIGSEGV during execution.

```
REAL(8) A(10)
COMMON //N,A
!OCL SIMD(ALIGNED)
DO I=1,10
  A(I) = ...
END DO
```

Double precision floating-point store may not be allocated on a 16-byte boundary because Array A is an element that is not the first or the last.

```
REAL(8) A(10)
COMMON //A
!OCL SIMD(ALIGNED)
DO I=2,10
  A(I) = ...
END DO
```

Even if Array A is on 16-byte boundary, the second and the subsequent elements are referenced in the loop and they are not on 16-byte boundary.

# -Ksimd\_packed\_promotion (1/2)

Option Name	Default
-Ksimd_packed_promotion -Ksimd_nopacked_promotion	-Ksimd_nopacked_promotion

## ● Function outline

- This option promotes packed-SIMDization, assuming that the index calculation of single precision real type and that 4-byte integer type array elements does not exceed the 4-byte range.
- The -Ksimd\_packed\_promotion option has meaning when the -Ksimd option is valid.
- When the access to an array is nested, the array address to be the index is converted into 8 bytes. So, an 8-byte integer type operation appears on the operation tree connecting to the index and packed-SIMDization is disturbed.

Packed-SIMDization becomes possible when all the following conditions are satisfied.

- Access to an array is nested in a loop.
- All the array data in the loop is 4-byte or less element type.
- The array of indexes to an array is 4 bytes.

## ● Effect

- It is expected to improve execution performance because this option promotes packed-SIMDization.

## ● Notes

- If the index calculation of array elements exceeds the 4-byte range, an invalid area is accessed, resulting in abnormal termination during execution or execution result error.
- If the -Ksimd\_packed\_promotion option is not specified, it becomes 8 SIMD instead of 16 SIMD.

## ● Example

### ● packed-SIMDization

- If C(I) exceeds the 4-byte range, an abnormal termination during execution or an execution result error may be generated.

```
SUBROUTINE TEST(A,B,C,D)
  USE_SIMFUNC
  INTEGER(KIND=4),DIMENSION(1:N) :: A,B
  INTEGER(KIND=4),DIMENSION(1:N) :: C,D
  !OCL SIMD(UNALIGNED)
  DO I=1,N
    A(C(I)) = B(D(I))
  ENDDO
END SUBROUTINE TEST
```

Because 4-byte type is allowed for index, the 4-byte address of C(I) is not converted into 8-byte type but optimization to connect the address directly to the index is implemented to promote packed-SIMDization.



Option Name	Default
-Ksimd_reg_size={128 256 512}	-Ksimd_reg_size=512

## ● Function outline

- This option specifies the size of SVE vector register in unit of bit.
- This option becomes valid when the -KSVE option is valid.
- The generated executable program operates normally only on the CPU architecture where SVE vector register in the size specified by this option.

## ● Effect

- It is expected to improve execution performance because this option promotes optimization.

## ● Notes

- When the specified size differs from the size of vector register on the CPU architecture where the program is executed, an abnormal termination during execution may occur. Even when an abnormal termination during execution does not occur, execution result is not guaranteed.
- When it is obvious that the specified size is smaller than the size of the vector register on the CPU architecture where the program is to be executed, it is necessary to set a valid vector register size by using, for example, the prctl(2) system call.

Option Name	Default
-Ksimd_reg_size=agnostic	-

## ● Function outline

- This option instructs to perform compilation without considering the SVE vector register as a specific size and create an executable program that determines the SVE vector register size at the time of execution.
- This option is valid when the -KSVE option is valid.
- An executable program is executable regardless the size of the SVE vector register on CPU.

## ● Effect

- It is expected to improve execution performance because this option promotes optimization.

## ● Note

- The execution performance may degrade, compared to the case when the  
-Ksimd\_reg\_size={128|256|512} option is specified.

## ● Example

- When the -Ksimd\_reg\_size=agnostic is specified, a similar optimization as -Ksimd\_reg\_size=512 (default) may not be executed. In such a case, execution performance may degrade.

When -Ksimd\_reg\_size=agnostic is specified

```

<<< Loop-information Start >>>
<<< [PARALLELIZATION]
<<<   Standard iteration count: 843
<<< [OPTIMIZATION]
<<<   COLLAPSED
<<<   SIMD(VL: AGNOSTIC; VL: 2 in 128-bit)
<<<   PREFETCH(HARD) Expected by compiler :
<<<     B, A
<<< Loop-information End >>>
6 1 pp 2v DO J=1,N
<<< Loop-information Start >>>
<<< [OPTIMIZATION]
<<<   COLLAPSED
<<< Loop-information End >>>
7 2 p 2 DO I=1,N
8 2 p 2v A(I,J) = A(I,J) + C * B(I,J)
9 2 p 2v ENDDO
10 1 p ENDDO

```

When agnostic is selected, it is judged that software pipelining will not be effective, and software pipelining is suppressed.

When -Ksimd\_reg\_size=512 (default) is specified

```

<<< Loop-information Start >>>
<<< [PARALLELIZATION]
<<<   Standard iteration count: 843
<<< [OPTIMIZATION]
<<<   COLLAPSED
<<<   SIMD(VL: 8)
<<<   SOFTWARE PIPELINING
<<<     (IPC: 3.00, ITR: 192, MVE: 7, POL: S)
<<<   PREFETCH(HARD) Expected by compiler :
<<<     B, A
<<< Loop-information End >>>
6 1 pp 2v DO J=1,N
<<< Loop-information Start >>>
<<< [OPTIMIZATION]
<<<   COLLAPSED
<<< Loop-information End >>>
7 2 p 2 DO I=1,N
8 2 p 2v A(I,J) = A(I,J) + C * B(I,J)
9 2 p 2v ENDDO
10 1 p ENDDO
11 END SUBROUTINE

```

Option Name	Default
-Ksimd_use_multiple_structures -Ksimd_nouse_multiple_structures	-Ksimd_use_multiple_structures
Optimization Specifier	
SIMD_USE_MULTIPLE_STRUCTURES SIMD_NOUSE_MULTIPLE_STRUCTURES	

## ● Function outline

- This option specifies whether to use the Load Multiple Structures instructions and the Store Multiple Structures instructions of SVE at SIMDization.
- This option becomes valid when the `-Ksimd` option and the `-KSVE` option are valid.

## ● Effect

- It is expected to improve execution performance by using the Load Multiple Structures instructions and the Store Multiple Structures instruction for SIMDization-target loading and storing.

## ● Point

- For details, see “Using the Multiple Structures Instruction” in *Programming Guide - Tuning*.

## ● Note

- Performance may degrade depending on data alignment.

## ● Example

```
!OCL SIMD_USE_MULTIPLE_STRUCTURES
DO I=1,N
  A(I) = B(1,I) + B(2,I) + B(3,I)
ENDDO
```

In the target loops, the Load Multiple Structures instructions and the Store Multiple Structures instructions of SVE are used.

Load Multiple Structures instructions of SVE at assembler

```
LD3D {Z0.D, Z1.D, Z2.D}, P0/Z, [X6, 0, MUL VL] // "b"
```

Load instructions of B(1,I), B(2,I), and B(3,I) are implemented in a single instruction (LD3D) of a structure instruction.

Option Name	Default
-Ksimd_uncounted_loop -Ksimd_nouncounted_loop	-Ksimd_nouncounted_loop

## ● Function outline

- This option specifies whether to create an object using SIMD instructions for loops whose iteration counts are unknown (instructions in DO WHILE loops, DO UNTIL loops, and DO loops with statements to end the loops).
- This option becomes valid when -Ksimd option and -KSVE options are valid.

## ● Effect

- It is expected to improve execution performance because an object using SIMD instructions is generated.

## ● Note

- Execution performance may degrade when the iteration count is small.

Option Name	Default
-Kloop_part_simd -Kloop_nopart_simd	-Kloop_nopart_simd

## ● Function outline

- This option specifies whether a loop is split and SIMDized partially when the loop has both the SIMDizable part and unSIMDizable part.
- The target loops are inner-most loops.
- This option becomes valid when the -Ksimd option is valid.

## ● Effect

- It is expected to improve execution performance because an object using SIMD instructions is generated.

## ● Note

- Execution may take longer if there are too many work areas for data passing due to loop fission.

## ● Example

Original source

```
!OCL LOOP_PART_SIMD
DO I=2,N
  A(I) = A(I) - B(I) + LOG(C(I)) ! SIMDizable
  D(I) = D(I-1) + A(I)           ! unSIMDizable
ENDDO
```

Sample source after optimization

```
DO I=2,N
  A(I) = A(I) - B(I) + LOG(C(I)) ! SIMD executed
ENDDO
DO II=2,N
  D(II) = D(II-1) + A(II)        ! non-SIMD executed
ENDDO
```

Only loop I that is split is SIMDized.

## Optimization Specifier

NOVREC[(array1[,array2]...)]

NORECURRENCE [(array1[,array2]...)]

### ● Function outline

- The NOVREC specifier instructs that a loop has no array of recursive operation. It uses SIMD instructions for an array in the loop but may not use SIMD instructions due to the operation type and the loop structure. Also, a pointer array cannot be specified for array.
- The NORECURRENCE specifier instructs that definitions of array elements to be the target of operation in a DO loop are not referenced beyond an iteration. This makes DO loops the following optimization target if the DO loops cannot be optimized due to unknown array definition reference order.
  - Loop slice (automatic parallelization)
  - SIMDization
  - Software pipelining
- array1, array2, ... are array names.

### ● Effect

- It is expected to improve execution performance because an object using SIMD instructions is generated.

### ● Notes

- Execution result with the NOVREC Specifier is not guaranteed if the array used in the loop is recursive data.
- Execution result is not guaranteed if the NORECURRENCE specifier is specified by mistake to an array dependent on the iteration count.



## ● Examples

### ● NOVREC specifier

```
REAL A(20),B(20)
!OCL NOVREC
DO I=2,10
  A(I) = A(I+N) + 1
  B(I) = B(I+M) + 2
ENDDO
```

This specifier instructs that all the arrays in the loop do not become recursive operation. SIMD instructions are used for operations of array A and B.

```
REAL A(20),B(20)
!OCL NOVREC(A)
DO I=2,10
  A(I) = A(I+N) + 1
  B(I) = B(I) + 2
ENDDO
```

The specifier instructs that array A with unknown data dependency is not a recursive operation. SIMD instructions are used for operations of array A and B.

### ● How to select the NOVREC specifier and NORECURRENCE specifier

The specifier can be used without specifying individual array names so as to promote SIMDization.

```
!OCL NOVREC
DO I=1,N
  A(I) = A(I+M)+1
  B(I) = B(I)+1
  :
ENDDO
```

Select the specifier to use depending on the value of variable M.

- Use the NOVREC specifier when the value of M may be 1 or above.
- Use the NORECURRENCE specifier when the value of M is 0 alone.

Option Name	Default
-Kswp -Knoswp	-O2 or above: -Kswp -O1: -Knoswp
Optimization Specifier	
SWP NOSWP	

- Function outline
  - This option performs optimization by software pipelining.
  - When the -Kswp option is specified together with the -Kswp\_weak option or the -Kswp\_strong option, the one specified later becomes valid.
  - This is valid with -O2 or above.
  - Optimization is not performed under the following condition.
    - It is judged the optimization will not produce any effects.
- Effect
  - It is expected to improve execution performance because the order of instructions are changed so that instructions in the loop are executed in parallel as much as possible.
- Point
  - SWP specifier has an effect equivalent to the -Kswp\_strong option.
- Notes
  - The loop needs to be iterated for sufficient times because the shape of the loop is changed by performing instruction scheduling that overlaps executable statements between an arbitrary time of loop iteration and subsequent loop iterations.
  - The size of the object program increases when software pipelining is applied for a loop with a variable as the iteration count.

# -Kswp\_weak/-Kswp\_strong (1/2)

Option Name	Default
-Kswp_weak	-
-Kswp_strong	-

## Optimization Specifier

SWP\_WEAK

### ● Function outline

- The option instructs software pipelining application method.

Option	Software Pipelining Application Method
-Kswp_weak	• Adjusts software pipelining to make overlapping of executable statements in the loop small.
-Kswp_strong	• Relaxes the software pipelining application condition and applies software pipelining forcibly even when the loop body is large.

- When the -Kswp option, the -Kswp\_weak option, and the -Kswp\_strong option are specified together, the one specified last becomes valid.
- Other functions and notes are the same as those for the -Kswp option.

## ● Effect

- It is expected to improve execution performance because executing a loop with software pipelining applied lessens the required iteration of the loop.

Option	Effect
-Kswp_weak	<ul style="list-style-type: none"><li>• Optimization effects are expected when the iteration count of the loop is unknown at the time of compilation and the iteration count of the loop is small.</li></ul>
-Kswp_strong	<ul style="list-style-type: none"><li>• Optimization effects are expected when the iteration count of the loop is large.</li></ul>

## ● Point

- ITR of the route that software pipelining go through varies depending on whether the -Kswp\_weak option is specified.

## ● Note

Option	Note
-Kswp_weak	<ul style="list-style-type: none"><li>• Execution performance may degrade because overlapping of executable statements becomes smaller.</li></ul>
-Kswp_strong	<ul style="list-style-type: none"><li>• Compilation memory and compilation time may increase significantly.</li></ul>

Option Name	Default
-Kswp_freg_rate=n	-Kswp_freg_rate=100
-Kswp_ireg_rate=n	-Kswp_ireg_rate=100
-Kswp_preg_rate=n	-Kswp_preg_rate=100
Optimization Specifier	
SWP_FREG_RATE(n)	
SWP_IREG_RATE(n)	
SWP_PREG_RATE(n)	

- Function outline
  - This option specifies the ratios (percentage) available for floating-point register, SVE, integer register, and predicate register in software pipelining.  
Software pipelining is performed assuming all the registers (32 registers for freg) are available if the argument is 100 and twice as many as the registers of argument 100 (64 registers for freg) are available if the argument is 200.
  - This is valid with -O2 or above.
- Effect
  - Software pipelining application can be adjusted by changing the condition related to the number of registers.
- Point
  - If software pipelining is not applied due to register shortage, specifying an integer larger than 1000 may enable forcible application of software pipelining though spill/fill is generated.
- Note
  - Execution performance may degrade due to change in register backup/restore instructions to memory.

## ● Example

Software pipelining cannot be applied due to floating-point register shortage.

```

7 1      DO J=1,N
8 1      !OCL UNROLL(8)
        <<< Loop-information Start >>>
        <<< [OPTIMIZATION]
        <<< SIMD(VL: 8)
        <<< PREFETCH(HARD) Expected by compiler :
        <<< D, C, A
        <<< Loop-information End >>>
9 2      8v    DO I=1,N
10 2     8v    A(I,J) = B(J,I) + C(I) / D(I,J)
11 2     8v    ENDDO
12 1      ENDDO
    
```

jwd8666o-i "a.f90", line 9: Cannot apply software pipelining due to floating-point register shortage

Available floating-point registers are increased by the SWP\_FREG\_RATE specifier to promote software pipelining.

```

        <<< Loop-information Start >>>
        <<< [OPTIMIZATION]
        <<< PREFETCH(HARD) Expected by compiler :
        <<< D, C, A
        <<< Loop-information End >>>
7 1      DO J=1,N
8 1      !OCL UNROLL(8)
9 1      !OCL SWP_FREG_RATE(120)
        <<< Loop-information Start >>>
        <<< [OPTIMIZATION]
        <<< SIMD(VL: 8)
        <<< SOFTWARE PIPELINING
            (IPC: 2.52, ITR: 192, MVE: 2, POL: S)
        <<< PREFETCH(HARD) Expected by compiler :
        <<< C, D, A
        <<< SPILLS :
        <<< GENERAL   : SPILL 0 FILL 0
        <<< SIMD&FP   : SPILL 0 FILL 0
        <<< SCALABLE  : SPILL 4 FILL 16
        <<< PREDICATE : SPILL 0 FILL 0
        <<< Loop-information End >>>
10 2     8v    DO I=1,N
11 2     8v    A(I,J) = B(J,I) + C(I) / D(I,J)
12 2     8v    ENDDO
13 1      ENDDO
    
```

jwd8204o-i "a.f90", line 10: Applied software pipelining to loop

jwd8205o-i "a.f90", line 10: A loop with software pipelining applied is selected at the time of execution when the loop iteration count is 192 or above.

Option Name	Default
-Kswp_policy={auto small large}	-Kswp_policy=auto

Optimization Specifier
SWP_POLICY({AUTO SMALL LARGE})

## ● Function outline

- This option specifies the criteria to select the instruction scheduling algorithm to be used by software pipelining.
- Software pipelining is performed when the -Kswp option, the -Kswp\_weak option, the -Kswp\_strong options, or a specifier corresponding to each option is valid.

Argument	Algorithm Selection Criteria
auto/AUTO	<ul style="list-style-type: none"> <li>• Selects the instruction scheduling algorithm automatically for each loop.</li> </ul>
small/SMALL	<ul style="list-style-type: none"> <li>• Uses the instruction scheduling algorithm appropriate for small loops (for example, loops that require a small number of registers).</li> </ul>
large/LARGE	<ul style="list-style-type: none"> <li>• Uses the instruction scheduling algorithm appropriate for large loops (for example, loops that require a large number of registers).</li> </ul>

## ● Effect

- It is expected to improve execution performance because software pipelining is applied.

## ● Point

- Try auto/AUTO first because the algorithm is selected automatically.

Option Name	Default
-Ksch_pre_ra -Ksch_nopre_ra	-O1 or above: -Ksch_pre_ra -O0 or below: -Ksch_nopre_ra
-Ksch_post_ra -Ksch_nopost_ra	-O1 or above: -Ksch_post_ra -O0 or below: -Ksch_post_ra

## ● Function outline

- This option performs instruction scheduling before and after register allocation.

## ● Effect

- It is expected to improve execution performance by using strong prefetch.

## ● Point

- Specifying the -Ksch\_post\_ra option may improve execution performance when spill/fill is found at listing.

## ● Notes

- Execution performance may degrade with the -Ksch\_pre\_ra option because instructions to back up/restore the registers in/from memory may increase. Generally, users need not be aware of it because it is processed properly at optimization.
- Instructions to back up/restore the registers in/from memory do not increase with the -Ksch\_post\_ra option.



Option Name	Default
-Kprefetch_sequential[={auto soft}] -Kprefetch_nosequential	-O2 or above: -Kprefetch_sequential -O1 or below: -Kprefetch_nosequential
Optimization Specifier	
PREFETCH_SEQUENTIAL[({AUTO SOFT})] PREFETCH_NOSEQUENTIAL	

## ● Function outline

- This option generates an object that uses prefetch instructions for sequentially accessed array data used in the loop.
- This becomes valid with -O1 or above.

Argument	How to Select Prefetch
auto (default)	• The compiler automatically selects whether to use hardware prefetch or issue a prefetch instruction.
soft	• Does not use hardware prefetch but issues a prefetch instruction.

## ● Effect

- It is expected to improve execution performance by the selected prefetch.

## ● Note

- Execution performance may degrade due to cache efficiency, branch existence, or index complexity of the loop.

Option Name	Default
-Kprefetch_cache_level={1 2 all}	-Kprefetch_cache_level=all
Optimization Specifier	
PREFETCH_CACHE_LEVEL(1 2 all)	

## ● Function outline

- This option instructs to which cache level data is prefetched.

Argument	Cache Level to Prefetch
1	• Instructs to prefetch data to L1 cache.
2	• Instructs to prefetch data to L2 cache.
all (default)	• Instructs to prefetch data to cache in every layer. • More advanced prefetch can be implemented by combining prefetch instructions to each layer.

## ● Effect

- It is expected to improve execution performance by prefetch to the specified cache.

## ● Points

- Try auto first because the automatic prefetch function becomes valid.
- For detailed explanation, see "Prefetch" in *Programming Guide - Integrated Programming Guide*. For usage, see "Data Access Wait Time (Hidden Latency)" in *Programming Guide - Tuning*.

## ● Note

- Execution performance may degrade unless a proper cache level is specified.

# PREFETCH\_STRIDE (1/2)

Option Name	Default
-Kprefetch_stride[={soft hard_auto hard_always}] -Kprefetch_nostride	-Kprefetch_nostride
Optimization Specifier	
PREFETCH_STRIDE[( {SOFT   HARD_AUTO   HARD_ALWAYS} )] PREFETCH_NOSTRIDE	

## ● Function outline

- This option generates an object that uses prefetch instructions for array data accessed with a stride larger than the cache line size used in the loop.
- The -Kprefetch\_stride=soft option and the -Kprefetch\_nostride option become valid with -O1 or above.
- The -Kprefetch\_stride=hard\_auto option and the -Kprefetch\_stride=hard\_always option become valid when the -Khptag option and -O1 or above is specified.
- How to prefetch array data can be specified.

Argument	How to Prefetch
soft (default)	<ul style="list-style-type: none"><li>• Performs prefetch by generating prefetch instructions.</li></ul>
hard_auto	<ul style="list-style-type: none"><li>• Performs prefetch by using hardware stride prefetcher.</li><li>• Specifies stride prefetcher so that only data not on cache is prefetched.</li></ul>
hard_always	<ul style="list-style-type: none"><li>• Performs prefetch by using hardware stride prefetcher.</li><li>• Specifies stride prefetcher so that prefetch is always performed.</li></ul>

## ● Effect

- It is expected to improve execution performance by the specified prefetch.

- Note
  - Execution performance may degrade due to cache efficiency, branch existence, or index complexity of the loop.
- Example

```
!OCL PREFETCH_STRIDE(SOFT)
DO I=1,N,K
  A(I) = B(I)
ENDDO
```

A prefetch instruction is generated in the target loop.

# PREFETCH\_STRONG/PREFETCH\_STRONG\_L2 (1/2)

Option Name	Default
-Kprefetch_strong -Kprefetch_nostrong	-Kprefetch_strong
-Kprefetch_strong_L2 -Kprefetch_nostrong_L2	-Kprefetch_strong_L2

Optimization Specifier
PREFETCH_STRONG PREFETCH_NOSTRONG PREFETCH_STRONG_L2 PREFETCH_NOSTRONG_L2

## ● Function outline

- This option uses strong prefetch for prefetch instructions generated for L1 cache and L2 cache.
- When any of the following options
  - -Kprefetch\_sequential
  - -Kprefetch\_stride
  - -Kprefetch\_indirect

is valid, and:

- When the -Kprefetch\_cache\_level=all option is specified, both the -Kprefetch\_strong option and the -Kprefetch\_strong\_L2 option become valid.
- When the -Kprefetch\_cache\_level=1 option is specified, the -Kprefetch\_strong option becomes valid.
- When the -Kprefetch\_cache\_level=2 option is specified, the -Kprefetch\_strong\_L2 option becomes valid.

- Effect

- It is expected to improve execution performance by using strong prefetch.

- Notes

- There are “Strong property” and “Weak property” for prefetch instruction. With “Strong property,” the hardware tries to complete the prefetch request as much as possible. With “Weak property,” the prefetch request is deleted unless the hardware resource is sufficient.
- The default of L1 cache is “Weak property” on K computer, but it is changed to “Strong property” on Fugaku.

- Example

```
!OCL PREFETCH_STRONG
DO I=1,N
  A(I) = B(I)
ENDDO
```

Prefetch instructions of L1 cache generated in the target loop are strong prefetch.

Option Name	Default
-Kprefetch_conditional -Kprefetch_noconditional	-Kprefetch_noconditional
Optimization Specifier	
PREFETCH_CONDITIONAL PREFETCH_NOCONDITIONAL	

## ● Function outline

- This option generates prefetch instructions for array data used in blocks contained in IF constructs and CASE constructs.
- This option becomes valid when any of the following options is valid.
  - -Kprefetch\_sequential
  - -Kprefetch\_stride
  - -Kprefetch\_indirect

## ● Effect

- It is expected to improve execution performance by prefetch for array data.

## ● Note

- Execution performance may degrade due to cache efficiency, branch existence, or index complexity of the loop.

## Optimization Specifier

PREFETCH\_INDIRECT  
PREFETCH\_NOINDIRECT

### ● Function outline

- This specifier generates prefetch instructions for array data used by the loop through indirect access (list access).

### ● Effect

- It is expected to improve execution performance by prefetch for array data.

### ● Note

- Execution performance may degrade due to cache efficiency, branch existence or index complexity of the loop, or, the cost of prefetch instructions when prefetch target array data is consecutive or an identical value.

### ● Example

```
!OCL PREFETCH_INDIRECT  
DO I=1,N  
  A(INDX(I)) = B(INDX(I))  
ENDDO
```

A prefetch instruction is generated for indirectly accessed (through list access) array data used by the loop through indirect access.



## Optimization Specifier

PREFETCH\_LINE(n)

PREFETCH\_LINE\_L2(n)

### ● Function outline

- Targets data locating “n” cache lines ahead when a prefetch instruction is generated for L1 cache or L2 cache.
- Specify an integer value between 1 and 100 for “n”.

### ● Effect

- It is expected to improve execution performance by prefetch for array data.

### ● Point

- Hardware prefetch prefetches data at a relatively far location such as six cache lines ahead at maximum for L1 cache and 40 cache lines ahead at maximum for L2 cache. So, specifying a close location in this option may produce effects.

### ● Example

```
!OCL PREFETCH_LINE(4)
DO I=1,N
  A(I) = B(I)
ENDDO
```

For prefetch instructions of L1 cache generated in the loop, the target of prefetch is data four cache lines ahead for prefetch.

The size of L1 cache line is 256 bytes. It is the same size as  $256/8=32$  elements when accessed by double precision real type (8 bytes).

In the left example, data of 4 cache lines \* 32 elements = 128 elements ahead is prefetched.

## Optimization Specifier

```
PREFETCH_READ(name[,level={ 1 | 2 }][,strong={ 0 | 1 }])
```

```
PREFETCH_WRITE(name[,level={ 1 | 2 }][,strong={ 0 | 1 }])
```

### ● Function outline

- PREFETCH\_READ generates prefetch instructions for the referenced data, while PREFETCH\_WRITE generates prefetch instructions for the defined data.
- Specify the data (an array element or an array section) referenced/defined in the program for name.  
In addition to specification by elements, vector specification is available.
- Specify “level” to specify to which cache level data is prefetched.  
1 means L1 cache, while 2 means L2 cache. The default is level=1.
- Specify “strong” to specify whether to use strong prefetch.
  - When strong=0 is specified, strong prefetch is not used.
  - When strong=1 is specified, strong prefetch is used.
  - The default is strong=1.

### ● Effect

- It is expected to improve execution performance by prefetch for array data.

### ● Points

- Use the PREFETCH\_WRITE specifier for referenced and defined data.
- When vector specification is used for “name,” the compiler generates a prefetch instruction per cache line dynamically.  
Also, the performance may be better than specification by element because two or more prefetch instructions can be generated at the same time.

## ● Example

### Specification by element

```
DO J=1,N
  DO I=1,ISIZE
    !OCL PREFETCH_WRITE(A(I,J+1),level=1)
    !OCL PREFETCH_READ(B(I,J+1),level=1)
    !OCL PREFETCH_READ(C(I,J+1),level=1)
    A(I,J) = B(I,J) + SCALAR * C(I,J)
  ENDDO
ENDDO
```

Prefetch instructions are generated for A(I,J+1), B(I,J+1), and C(I,J+1).

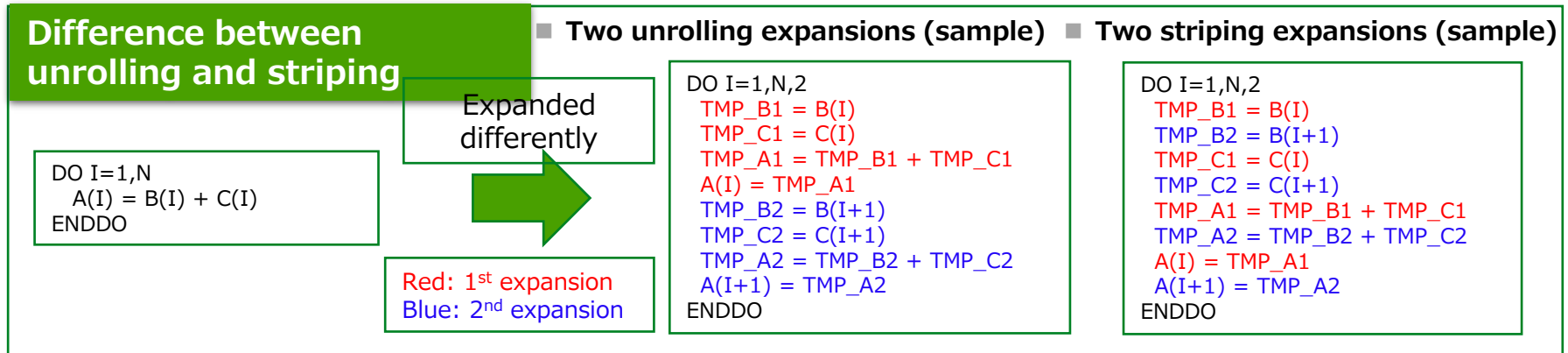
### Specification by vector (recommended)

```
DO J=1,N
!OCL PREFETCH_WRITE(A(1:ISIZE,J+1),level=1)
!OCL PREFETCH_READ(B(1:ISIZE,J+1),level=1)
!OCL PREFETCH_READ(C(1:ISIZE,J+1),level=1)
  DO I=1,ISIZE
    A(I,J) = B(I,J) + SCALAR * C(I,J)
  ENDDO
ENDDO
```

Prefetch instructions are generated for A(1:ISIZE,J+1), B(1:ISIZE,J+1), and C(1:ISIZE,J+1).

Compared to specification by element, performance may be improved because the number of instructions in the inner-most loop can be reduced by vector specification outside of the loop.

- There are two types of loop expansion as shown below.
  - Unrolling (unroll)
  - Striping (striping)



## Points

- Unrolling is recommended in general. Unrolling does not change the order of instructions. So, the expansion count is as small as two or three and striping is effective to change the order of instructions securely.
- When -Kstriping option and -Kunroll option are specified together, the one specified later is valid.

## Note

- Striping uses more registers than unrolling. So, longer stripe length may degrade the execution performance.

Option Name	Default
-Kunroll[=n] -Knounroll	-O2 or above: -Kunroll -O1 or below: -Knounroll
Optimization Specifier	
UNROLL[(n)] UNROLL('full')	

## ● Function outline

- This option expands all the executable statements in a DO loop “n” times in the loop to reduce the iteration count of the loop to once in “n” times.
- If -Ksimd[=level] option is valid and SIMDization of the loop components is performed, this option expands executable statements SIMD-length \* n times to reduce the iteration count of the loop to once in SIMD-length \* “n”.
- The target loops are those that are not entered or not exited.
- When n is omitted to be specified, the compiler automatically determines the best value.
- Use the UNROLL specifier to apply this to a specific DO loop.

## ● Effect

- The iteration count becomes smaller and the overhead due to loop iteration becomes smaller. Also, it is expected to improve the execution performance because executable statements multiply expanded are optimized as a set of the loop.

## ● Points

- Ordinary loops are unrolled by software pipelining that performs instruction scheduling with overlapped executable statements in the loop.  
Loops with redirection operations are unrolled by unrolling and software pipelining.
- Loop unrolling is performed before and after SIMDization.

Before SIMDization	• To apply SIMDization or software pipelining to the outer loop, full unrolling is performed on the inner loop with the small iteration count.
After SIMDization	• To promote optimization such as common expressions, unrolling or full unrolling is performed on the inner loop.

For loops where the -Kunroll[=n] option or the UNROLL specifier is valid, loop unrolling is performed before and after SIMDization. If the iteration count of an inner loop is small, full unrolling is performed by software pipelining.

- Statements with array assignment and array expression (array description) are expanded to a Do loop and become the target of unrolling.
- Instruction scheduling/software pipelining promotion may resolve the operation wait state.

## ● Note

- Statements in a loop multiply as they expand and contract, so the size of the object program increases.

## ● Examples

- Unroll only (does not improve execution performance)

Original source

```
!OCL UNROLL(2)
DO I=1,N
  A(I) = B(I) + C(I)
ENDDO
```

Sample source after optimization

```
DO I=1,N,2
  TMP_B1 = B(I)
  TMP_C1 = C(I)
  TMP_A1 = TMP_B1 + TMP_C1
  A(I) = TMP_A1
  TMP_B2 = B(I+1)
  TMP_C2 = C(I+1)
  TMP_A2 = TMP_B2 + TMP_C2
  A(I+1) = TMP_A2
ENDDO
```

With unrolling only, Gather is output and execution performance is not improved.

- SIMDization + unrolling (improves execution performance)

Original source

```
!OCL UNROLL(2)
DO I=1,N
  A(I) = B(I) + C(I)
ENDDO
```

Sample source after optimization with SIMDization (VL:8) valid

```
DO I=1,N,15
  TMP_B1(1:7) = B(I:I+7)
  TMP_C1(1:7) = C(I:I+7)
  TMP_A1(1:7) = TMP_B1(1:7) + TMP_C1(1:7)
  A(I:I+7) = TMP_A1(1:7)
  TMP_B2(1:7) = B(I+8:I+15)
  TMP_C2(1:7) = C(I+8:I+15)
  TMP_A2(1:7) = TMP_B2(1:7) + TMP_C2(1:7)
  A(I+8:I+15) = TMP_A2(1:7)
ENDDO
```

## Optimization Specifier

FULLUNROLL\_PRE\_SIMD[(n)]

### ● Function outline

- This option controls full unrolling behavior before SIMDization.
- “n” represents an integer between two and 100 that expresses the upper bound of the iteration count of the target loop.
- When the value “n” is omitted, the compiler determines automatically optimum value.
- Only the DO loop or the array description right after this is specified, is the target.

### ● Effect

- The iteration count becomes smaller and the overhead due to loop iteration decreases. Also, it is expected to improve execution performance because executable statements expanded multiply are optimized as a set of the loop.

### ● Point

- Promotion of SIMDization may reduce the operation execution time.

### ● Note

- Optimization is not performed if the iteration count is unknown.



## ■ Example

Original source

```
DO I=1,N
!OCL FULLUNROLL_PRE_SIMD
  DO J=1,16
    A(I,J) = B(I,J) + C(I,J)
  ENDDO
ENDDO
```

Full unrolling before SIMDization  
is applied to the inner loop.

Sample source after optimization

```
DO I=1,N,8
  A(I:I+7,1) = B(I:I+7,1) + C(I:I+7,1)
  A(I:I+7,2) = B(I:I+7,2) + C(I:I+7,2)
  A(I:I+7,3) = B(I:I+7,3) + C(I:I+7,3)
  A(I:I+7,4) = B(I:I+7,4) + C(I:I+7,4)
  A(I:I+7,5) = B(I:I+7,5) + C(I:I+7,5)
  A(I:I+7,6) = B(I:I+7,6) + C(I:I+7,6)
  A(I:I+7,7) = B(I:I+7,7) + C(I:I+7,7)
  A(I:I+7,8) = B(I:I+7,8) + C(I:I+7,8)
  A(I:I+7,9) = B(I:I+7,9) + C(I:I+7,9)
  A(I:I+7,10) = B(I:I+7,10) + C(I:I+7,10)
  A(I:I+7,11) = B(I:I+7,11) + C(I:I+7,11)
  A(I:I+7,12) = B(I:I+7,12) + C(I:I+7,12)
  A(I:I+7,13) = B(I:I+7,13) + C(I:I+7,13)
  A(I:I+7,14) = B(I:I+7,14) + C(I:I+7,14)
  A(I:I+7,15) = B(I:I+7,15) + C(I:I+7,15)
  A(I:I+7,16) = B(I:I+7,16) + C(I:I+7,16)
ENDDO
```

The loop "DO I=1,N" is SIMDized (VL:8)  
after full unrolling of the inner loop.

Option Name	Default
-Kunroll_and_jam[=n] -Knounroll_and_jam	-Knounroll_and_jam
Optimization Specifier	
UNROLL_AND_JAM[(n)]	
UNROLL_AND_JAM_FORCE[(n)]	

## ● Function outline

- This option performs unrolling on an outer loop of a nested loop to expand it “n” times and then fuses the expanded inner loops.
- With the -Kunroll\_and\_jam option or the UNROLL\_AND\_JAM specifier, the optimization is not performed under any of the following condition.
  - The loop is the inner-most loop.
  - It is judged the optimization will not bear any effects (with any addition or subtraction in the second dimension, with reusability, or with any extractable common expression).
  - It is judged data dependency exists beyond an iteration.
- The UNROLL\_AND\_JAM\_FORCE specifier considers no data dependency beyond an iteration and always applies unroll-and-jam.
- To apply this to a specific DO loop, use the UNROLL\_AND\_JAM specifier or the UNROLL\_AND\_JAM\_FORCE specifier.

## ● Effects

- It is expected to improve execution performance because it promotes common expression elimination.
- The memory wait state may be resolved by streamlining L2 cache use.

## ■ Points

- Whether it is effective depends on each loop. So, it is recommended not to use the -Kunroll\_and\_jam[=N] option to apply to the whole program, but to use the UNROLL\_AND\_JAM specifier or the UNROLL\_AND\_JAM\_FORCE specifier to apply to individual loops.
- Use the UNROLL\_AND\_JAM\_FORCE specifier to apply unroll-and-jam forcibly when there is no data dependency beyond an iteration and it is judged that any common expressions can be extracted.

## ■ Notes

- The execution result is not guaranteed if the UNROLL\_AND\_JAM\_FORCE specifier is specified by mistake (there is data dependency beyond an iteration).
- If the iteration count of the inner-most loop is small, cache use efficiency may degrade depending on the increase of data streams and change in the data access order, leading to execution performance degradation.  
For details, see the item, "outer loop unrolling" in *Programming Guide - Tuning*.
- Increase in cache mistake may be improved by correction with prefetch.  
For details, see the item, "outer loop unrolling" in *Programming Guide - Tuning*.

## ■ Example

Original source

```
!OCL UNROLL_AND_JAM(2)
  DO J=1, 128
    DO I=1, 128
      A(I, J) = B(I, J) + B(I, J+1)
      ...
    END DO
  END DO
```

Sample source after optimization

```
DO J=1, 128, 2
  DO I=1, 128
    A(I, J) = B(I, J) + B(I, J+1)
    A(I, J+1) = B(I, J+1) + B(I, J+2)
    ...
  END DO
END DO
```

Option Name	Default
-Kstriping[=n] -Knostriping	-Knostriping
Optimization Specifier	
STRIPING[(n)]	

## ● Function outline

- This option expands all the executable statements in a DO loop for a certain (stripe length “n”) times within the loop and make the iteration count of the loop once in “n” times.
- The target loops are those that are not entered or not exited.
- To apply this to a specific DO loop, use the STRIPING specifier.

## ● Effect

- The iteration count becomes smaller and the overhead due to loop iteration decreases. Also, it is expected to improve execution performance including promotion of instruction scheduling.

## ● Point

- Striping is effective to change the order of instructions without fail when the expansion count is as small as two or three.

## ● Notes

- The order of instructions is changed without fail. However, execution performance may degrade when the stripe length “n” is long due because more registers are used.
- The size of the object program increases because statements in a loop expanded multiply.
- Compilation may take longer.

## ■ Example

Original source

```
!OCL STRIPING (2)
DO I=1,N
  A(I) = B(I) + C(I)
ENDDO
```

Sample source after optimization

```
DO I=1,N,2
  TMP_B1 = B(I)
  TMP_B2 = B(I+1)
  TMP_C1 = C(I)
  TMP_C2 = C(I+1)
  TMP_A1 = TMP_B1 + TMP_C1
  TMP_A2 = TMP_B2 + TMP_C2
  A(I) = TMP_A1
  A(I+1) = TMP_A2
ENDDO
```

Option Name	Default
-Kloop_fission -Kloop_nofission	-O2 or above: -Kloop_fission -O1 or below: -Kloop_nofission

Option Specifier
LOOP_FISSION_TARGET[({CL   LS})] FISSION_POINT[(n)]

## ● Function outline

- This option divides a loop into multiple smaller loops.
- The target loops are those that are not entered or not exited.
- To apply this to a specific DO loop, use the LOOP\_FISSION\_TARGET specifier.

Argument	Loop Fission Algorithm
CL (default)	<ul style="list-style-type: none"> <li>• Clustering algorithm.</li> <li>• Performs loop fission with giving priority to reduction of work array for temporary data transfer associated with loop fission.</li> </ul>
LS	<ul style="list-style-type: none"> <li>• Local search algorithm.</li> <li>• Performs loop fission with giving priority to software pipelining promotion.</li> </ul>

- Use the FISSION\_POINT specifier to split a loop at a specified location of a loop. This option splits nested loops of the depth “n” counting from the inner-most loop. Specify an integer between 1 and 6 for “n”. If “n” is omitted to be specified only the inner-most loops (first division) are divided.

## ● Effect

- It is expected to improve execution performance because loop fission brings loop interchange and parallelization of outer loops is enabled.

## ■ Note

- Local search algorithm takes more compilation time compared to clustering algorithm.

## ■ Example

- -Kloop\_fission option

Original source

```
DO I=1,N
  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)
ENDDO
```

Sample source after optimization

```
DO I=1,N
  E(I) = A1(I)*B1(I) + A2(I)*B2(I) + A3(I)*B3(I) + A4(I)*B4(I) + A5(I)*B5(I)
ENDDO
DO I=1,N
  F(I) = C1(I)*D1(I) + C2(I)*D2(I) + C3(I)*D3(I) + C4(I)*D4(I) + C5(I)*D5(I)
ENDDO
```

## ■ LOOP\_FISSION\_TARGET specifier

CL

```
!OCL LOOP_FISSION_TARGET(CL)
DO I=1,N
  S1 = A(I) + B(I)
  S2 = C(I) + D(I)
  ...
  P(I) = S1 + Q(I)
  X(I) = S2 + Y(I)
  ...
ENDDO
```

This splits loops with giving priority to argument reduction of work array for temporary data transfer associated with loop fission.

LS

```
!OCL LOOP_FISSION_TARGET(LS)
DO I=1,N
  S1 = A(I) + B(I)
  S2 = C(I) + D(I)
  ...
  P(I) = S1 + Q(I)
  X(I) = S2 + Y(I)
  ...
ENDDO
```

This argument splits loops with giving priority to software pipelining promotion.

## ■ FISSION\_POINT specifier

Original source

```
DO I=1,N
  DO J=1,N
    A(I,J) = B(I,J)
!OCL FISSION_POINT(1)
    C(I,J) = D(I,J)
  ENDDO
ENDDO
```

Loop fission is performed on Loop J in the above example.

Sample source after optimization

```
DO I=1,N
  DO J=1,N
    A(I,J) = B(I,J)
  ENDDO
  DO J=1,N
    C(I,J) = D(I,J)
  ENDDO
ENDDO
```



Option Name	Default
-Kloop_fission_threshold=n	-Kloop_fission_threshold=50

Optimization Specifier
LOOP_FISSION_THRESHOLD(n)

## ● Function outline

- This option specifies the threshold to determine the loop granularity (the number of instructions or registers in the loop) after loop fission.
- Specify a value between 1 and 100 for “n”.
- This option is valid when the LOOP\_FISSION\_TARGET specifier is specified in the optimization control line and the -Kloop\_fission option, the -Kocl option, and the -O2 option or above are valid.
- To apply this to a specific DO loop, use the LOOP\_FISSION\_THRESHOLD specifier.

## ● Effect

- It is expected to have effects of promoting software pipelining, improving cache memory use efficiency, and resolving register shortage by arranging the number of splits in loop fission.

## ● Point

- The smaller “n” becomes, the smaller loops after fission and the more the number of splits tend to be.

## ● Note

- Increase in the number of splits results in increase in work areas and cache memory used.

## ● Example

Original source (N= {50|20})

```
!OCL LOOP_FISSION_TARGET
!OCL LOOP_FISSION_THRESHOLD(N)
DO I=1,NN
A1(I) = A1(I) + B1(I)
...
A2(I) = A2(I) + B2(I)
...
A3(I) = A3(I) + B3(I)
...
A4(I) = A4(I) + B4(I)
...
ENDDO
```

Sample source after optimization  
(N=50)

```
DO I=1,NN
A1(I) = A1(I) + B1(I)
...
A2(I) = A2(I) + B2(I)
...
ENDDO
DO I=1,NN
A3(I) = A3(I) + B3(I)
...
A4(I) = A4(I) + B4(I)
...
ENDDO
```

Sample source after optimization  
(N=20)

```
DO I=1,NN
A1(I) = A1(I) + B1(I)
...
ENDDO
DO I=1,NN
A2(I) = A2(I) + B2(I)
...
ENDDO
DO I=1,NN
A3(I) = A3(I) + B3(I)
...
ENDDO
DO I=1,NN
A4(I) = A4(I) + B4(I)
...
ENDDO
```

Option Name	Default
-Kloop_fisson_stripmining[={n L1 L2}] -Kloop_nofisson_stripmining	-Kloop_nofission_stripmining
Optimization Specifier	
LOOP_FISSION_STRIPMINING[({n L1 L2})]	

## ● Function outline

- This option instructs to fractionate loops for smaller iteration counts after automatic loop fission.
- This option becomes valid when the LOOP\_FISSION\_TARGET specifier is specified in the optimization control line and the -Kloop\_fission option, the -Kocl option, and the -O2 option or above are valid.
- There are two ways to specify the strip length: Specifying the length (n) directly, and specifying the length to be the size appropriate for each cache layer ('L1' or 'L2').

Argument	Strip Length
n	<ul style="list-style-type: none"> <li>• Specify the strip length for "n". You can specify a value between 2 and 100000000 for "n".</li> </ul>
L1	<ul style="list-style-type: none"> <li>• Takes cache memory use efficiency into consideration and sets the strip length to fit the size of L1 cache.</li> </ul>
L2	<ul style="list-style-type: none"> <li>• Takes cache memory use efficiency into consideration and sets the strip length to fit the size of L2 cache.</li> </ul>

- When the argument is omitted, the compiler determines the value automatically.
- Strip mining is applied under any of the following conditions.
  - The option or the optimization specifier is valid.
  - The size of the temporary area is unknown at the time of compilation.
  - The size of the temporary area is clear at the time of compilation and 8 Kbytes or above.
  - The -Kstriping option or the STRIPING specifier is invalid.
  - Blocking is not applied.

## ● Effect

- It is expected to improve cache memory use efficiency for data accessed between split loops.

## ● Point

- It is recommended to set the strip length “n” to the length of software pipelining target.  
Adjust the strip length by compiling a source program without the -Kloop\_fisson\_stripmining option and referencing lister information of software pipelining.

## ● Notes

- If the strip length “n” is too long, it may not be stored in L1 cache.
- If the strip length “n” is too short, it may not go to the software pipelining route.
- If the strip length “n” is short, it may disturb cache access continuity and may prevent prefetch from working.

## ● Examples

- -Kloop\_fisson\_stripmining option

Original source

```
REAL A(L),B(L),P(L),Q
!OCL LOOP_FISSON_TARGET
DO I=1,L
  Q = A(I) + B(I)
  ...
  P(I) = P(I) + Q
  ...
ENDDO
```

Sample source  
after automatic loop fission

```
REAL A(L),B(L),P(L),Q
REAL TEMPARRAY_Q(L)
DO I=1,L
  TEMPARRAY_Q(I) = A(I) + B(I)
  ...
ENDDO
DO I=1,L
  P(I) = P(I) + TEMPARRAY_Q(I)
  ...
ENDDO
```

The compiler generates a temporary array  
TEMPARRAY Q.

Sample source after optimization

```
REAL A(L),B(L),P(L),Q
REAL TEMPARRAY_Q(256)
DO II=1,L,256
  DO I=II,MIN(L,II+255)
    TEMPARRAY_Q(I-II) = A(I) + B(I)
    ...
  ENDDO
  DO I=II,MIN(L,II+255)
    P(I) = P(I) + TEMPARRAY_Q(I-II)
    ...
  ENDDO
ENDDO
```

## ● LOOP\_FISSION\_STRIPMINING specifier

### Original source

```
REAL A(L),B(L),P(L),Q
!OCL LOOP_FISSION_TARGET
!OCL LOOP_FISSION_STRIPMINING(256)
DO I=1,L
  Q = A(I) + B(I)
  ...
  P(I) = P(I) + Q
  ...
ENDDO
```

### Sample source after optimization

```
REAL A(L),B(L),P(L),Q
REAL TEMPARRAY_Q(256)
DO II=1,L,256
  DO I=II,MIN(L,II+255)
    TEMPARRAY_Q(I-II) = A(I) + B(I)
    ...
  ENDDO
  DO I= II,MIN(L,II+255)
    P(I) = P(I) + TEMPARRAY_Q(I-II)
    ...
  ENDDO
ENDDO
```

- A loop is generated outside the split loops and strip mining with strip length 256 is performed. At the same time, the compiler generates a temporary array to store the interim result between loops.
- The number of elements in the array TEMPARRAY\_Q is the same as the strip length, 256.

# -Kloop\_fusion (1/2)

Option Name	Default
-Kloop_fusion -Kloop_nofusion	-O2 or above: -Kloop_fusion -O1 or below: -Kloop_nofusion
Optimization Specifier	
LOOP_NOFUSION	

## ● Function outline

- This option instructs to fuse adjacent loops if -O2 or above is specified.
- To suppress loop fusion of a specific DO loop, use the LOOP\_NOFUSION specifier.

## ● Effect

- It is expected to generate an effect of localizing data by loop fusion.

## ● Note

- Excessive fusion causes too large loop bodies, which may prevent software pipelining from working.

## ● Examples

- -Kloop\_fusion option

Original source

```
SUBROUTINE SUB(A, B, C, D, E, N)
REAL*8 A(N), B(N), C(N)
REAL*8 D(N), E(N)
DO I=1,N
  A(I)=B(I)+C(I)
ENDDO
DO I=1,N
  D(I)=A(I)+E(I)
ENDDO
END SUBROUTINE SUB
```

Sample source after optimization

```
SUBROUTINE SUB (A, B, C, D, E, N)
REAL*8 A(N), B(N), C(N)
REAL*8 D(N), E(N)
DO I=1,N
  A(I)=B(I)+C(I)
  D(I)=A(I)+E(I)
ENDDO
END SUBROUTINE SUB
```

- LOOP\_NOFUSION specifier

Original source

```
!OCL LOOP_NOFUSION
DO I=1,N
  A(I)=B(I)+C(I)
ENDDO
DO J=1,N
  D(J)=E(J)+F(J)
ENDDO
DO K=1,N
  G(K)=H(K)+L(K)
ENDDO
```

Sample source after optimization

```
!OCL LOOP_NOFUSION
DO I=1,N
  A(I)=B(I)+C(I)
ENDDO
DO J=1,N
  D(J)=E(J)+F(J)
  G(J)=H(J)+L(J)
ENDDO
```

Loop I and loop J are not fused. Loop J and loop K are fused.

Option Name	Default
-Kloop_interchange -Kloop_nointerchange	-O2 or above: -Kloop_interchange -O1 or below: -Kloop_nointerchange
Optimization Specifier	
LOOP_INTERCHANGE(var1,var2[,var3]...)	

## ● Function outline

- This option instructs to perform loop interchange if -O2 or above is specified.
- To perform loop interchange of a specific DO loop, use the LOOP\_INTERCHANGE specifier. This specifier performs nested DO loop interchange in the specified order (var1, var2, ...). var1, var2, var3, ... are DO variable names.
- Interchange is not performed if it is impossible, for example, if the result changes by interchange.

## ● Effect

- It is expected to generate effect of improving data access efficiency by loop interchange.

## ● Point

- It is deemed desirable to switch loops so that data access in the loop becomes continuous access.  
In particular, make the left side of the expression accessed continuously.



## ● Example

Original source

```
!OCL LOOP_INTERCHANGE(I,J)
  DO I=1,M
    DO J=1,N
      A(I,J) = A(I,J) + B(J,I)
    ENDDO
  ENDDO
```

Sample source after optimization

```
DO J=1,N
  DO I=1,M
    A(I,J) = A(I,J) + B(J,I)
  ENDDO
ENDDO
```

- When the LOOP\_INTERCHANGE specifier is not specified, the loops are put into parallel with DO variable I. When the LOOP\_INTERCHANGE specifier is specified, loops are switched to be put into parallel with DO variable J.

## Optimization Specifier

### UNSWITCHING

#### ● Function outline

- This specifier instructs to perform loop unswitching of the specified IF construct.
- Specify this optimization line right before an IF construct unchanged in the loop.  
If it is specified in other location, it becomes invalid.

#### ● Effect

- It is expected to promote SIMDization and software pipelining because all the branches in the loop disappear.

#### ● Note

- If the loop to be the target of loop unswitching has many executable statements, compilation memory and compilation time may increase significantly.

## ● Examples

Original source

```
DO I=1,N
!OCL UNSWITCHING
  IF (X == 0) THEN
    A(I) = B(I)
  ELSE
    A(I) = C(I)
  ENDIF
ENDDO
```

Loop unswitching of the IF construct is performed.

Sample source after optimization

```
IF (X == 0) THEN
  DO I=1,N
    A(I) = B(I)
  ENDDO
ELSE
  DO I=1,N
    A(I) = C(I)
  ENDDO
ENDIF
```

Original source

```
DO I=1,N
  IF (X == 0) THEN
    A(I) = B(I)
!OCL UNSWITCHING
  ELSE IF (X == 1) THEN
    A(I) = C(I)
  ELSE
    A(I) = D(I)
  ENDIF
ENDDO
```

Loop unswitching is performed only on IF constructs with the UNSWITCHING specifier specified.  
Loop unswitching is not performed on IF constructs without the UNSWITCHING specifier specified even if it is in a loop that contains another IF construct with the UNSWITCHING specifier specified.

Sample source after optimization

```
IF (X == 1) THEN
  DO I=1,N
    IF (X == 0) THEN
      A( I ) = B( I )
    ENDIF
    A( I ) = C( I )
  ENDDO
ELSE
  DO I=1,N
    IF (X == 0) THEN
      A( I ) = B( I )
    ENDIF
    A( I ) = D( I )
  ENDDO
ENDIF
```

Option Name	Default
-Kloop_perfect_nest -Kloop_noperfect_nest	-O3 or above: -Kloop_perfect_nest -O2 or below: -Kloop_noperfect_nest
Optimization Specifier	
LOOP_PERFECT_NEST LOOP_NOPERFECT_NEST	

## ● Function outline

- This option instructs to split an imperfect nested loop into perfect nested loops when “O2 – Kloop\_perfect\_net” or -O3 or above is specified.
- To split a specific imperfect nested loop into perfect nested loops, use the LOOP\_PERFECT\_NEST specifier.

## ● Effect

- It is expected to produce an effect to promote optimization of loop interchange, loop unnesting, etc. by making an imperfect nested loop a perfect nested loop.

## ● Example

Original source

```
!OCL LOOP_PERFECT_NEST
DO J=1,N ! imperfect nested loop
  A(J) = B(J)+1
  DO I=1,N
    C(J,I) = D(J,I)+A(J)
  ENDDO
ENDDO
```

Sample source after optimization

```
DO J=1,N
  A(J) = B(J)+1
ENDDO
DO J=1,N ! perfect nested loop
  DO I=1,N
    C(J,I) = D(J,I)+A(J)
  ENDDO
ENDDO
```

Imperfect nested loops J and I are split into perfect nested loops.

Option Name	Default
-Kzfill[=N] -Knozfill	-Knozfill

## ● Function outline

- This option does not load array data that performs only writing in the loop from the memory but instructs to use an instruction to acquire the cache line for writing on cache.
- By specifying “N”, the data “N” cache lines ahead becomes the optimization target.  
It is recommended not to use the option to apply this to the whole program but use the ZFILL specifier to apply this to individual loops.
- This option becomes valid when -O2 or above is valid. Optimization is not performed on arrays with reference within the same loops, discontinuously accessed arrays, or arrays stored under IF constructs.

## ● Effect

- It is expected to improve execution performance because writing of array data that performs only writing in the loop becomes faster.

## ● Notes

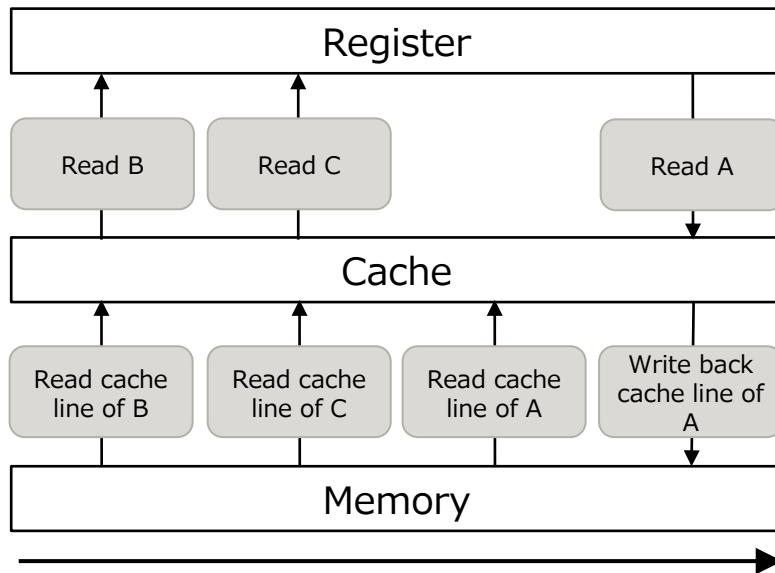
- When the -Kzfill option is applied, the prefetch instruction to L2 cache is not issued.
- The loop is transformed so that the cache line acquired by this optimization is always stored. So, the following optimization cannot be applied.
  - Loop unrolling
  - Loop striping
- Applying this optimization may degrade execution performance under any of the following conditions.
  - The program is not under the influence of the memory bandwidth bottleneck.
  - The iteration count of the loop is small.
  - "N" is used to specify the number of cache lines, and the iteration count is smaller than the number of elements in the cache line.
- If execution performance degrades by specifying the -Kzfill option, do not specify the -Kzfill option.

## ● Example

Original source

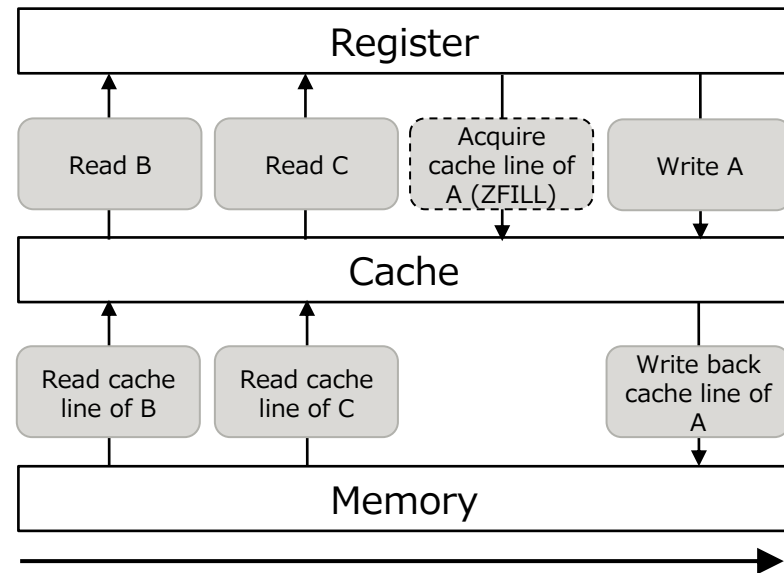
```
DO I=1,N  
  A(I) = B(I) + C(I)  
ENDDO
```

When -Kzfill is not specified



Memory is accessed four times in total.

When -Kzfill is specified



Reading of A from memory is not performed, so memory is accessed three times in total.

## Optimization Specifier

CLONE(var==n1[,n2]...)

### ● Function outline

- This specifier instructs to consider the value of variable `var` as constant in the loop, generate a branch where the equality with the specified variable “`var`” and the value `n1[,n2]` ... is the condition expression, and copy the loop.
- Branches are generated in order of specified values.
- “`Var`” is an integer-type variable. The kind type parameter is 1, 2, 4, or 8. `N1[,n2]` ... is a decimal number or a named constant between -9223372036854775808 and 9223372036854775807.
- This optimization specifier becomes valid when `-O3` or above is valid.

### ● Effect

- It is expected to improve execution performance because this promotes other optimization such as full unrolling.

### ● Notes

- The execution result is not guaranteed when variable “`var`” is updated in the target loop.
- The size of the object program may increase and compilation may take longer because loops are copied.



## ● Example

Original source

```
!OCL CLONE(N==10)
  DO I=1,N
    A(I) = I
  ENDDO
```

Sample source after optimization

```
IF (N==10) THEN
  DO I=1,10
    A(I) = I
  ENDDO
ELSE
  DO I=1,N
    A(I) = I
  ENDDO
ENDIF
```

Option Name	Default
-Kpreload -Knopreload	-Knopreload
Operation Specifier	
PRELOAD	

## ● Function outline

- This option performs speculative execution of load instructions after SIMDization of loops with IF statements (moves load instructions in THEN/ELSE clauses under IF to place it before the IF condition determination).

## ● Effect

- It is expected to improve execution performance because it promotes instruction scheduling.

## ● Point

- While the -Kpreex option and the PREEX specifier move an inequality under an IF statement in a loop to place it outside of the loop, the -Kpreload option and the PRELOAD specifier move load instructions under an IF statement in a loop to place it before the IF statement.

## ● Notes

- An exception that originally should not occur may occur and the execution may abort because a load instruction that should not be executed according to the program logic is executed.
- Whether the abortion is due to speculative execution of load instructions can be checked by the diagnosis message at the time of compilation.

## ■ Example

- Load instructions in B(I) and C(I) are moved to place them before the IF statement.

Original source

```
SUBROUTINE FOO(A,B,C,M,N)
REAL(8),DIMENSION(1:N) :: A,B,C
LOGICAL,DIMENSION(1:N) :: M
DO I=1,N
  IF (M(I) .GT. 0) THEN
    A(I) = B(I) + C(I)
  ENDIF
END SUBROUTINE
```

Sample source after optimization

```
SUBROUTINE FOO(A,B,C,M,N)
REAL(8),DIMENSION(1:N) :: A,B,C
LOGICAL,DIMENSION(1:N) :: M
REAL(8) :: TMP
DO I=1,N
  TMP = B(I) + C(I)
  IF (M(I) .GT. 0) THEN
    A(I) = TMP
  ENDIF
END SUBROUTINE
```

# -Kauto/-Kautoobjstack/-Ktemparraystack (1/2)

Option Name	Default
-Kauto -Knoauto	-Kauto
-Kautoobjstack -Knoautoobjstack	-Kautoobjstack
-Ktemparraystack -Knotemparraystack	-Ktemparraystack

## ● Function outline

- This option allocates each allocation target to the stack.

Option	Allocation Target
auto	<ul style="list-style-type: none"><li>• Local variables except variables with SAVE attribute and variables with default value</li></ul>
autoobjstack	<ul style="list-style-type: none"><li>• Automatic data object</li></ul>
temparraystack	<ul style="list-style-type: none"><li>• Interim result of array operation</li><li>• Masked evaluation result if the iteration count of DO CONCURRENT is a constant</li></ul>

## ● Effect

- It is expected to improve execution performance by allocation to the stack.

## ● Point

- This option did not operate by default on K computer. However, it operates by default on Fugaku and allocation to the stack is performed without the users' awareness.

## ● Notes

- Abnormal termination may occur when the stack runs short.
- To avoid abnormal termination, set a larger bound of the stack, or compile the program with the -Knoauto option, the -Knoautoobjstack option, or the -Knotemparraystack option to reduce the stack area size.
- For thread parallelized programs, specify the required thread stack size with the environment variable OMP\_STACKSIZE or THREAD\_STACK\_SIZE.
- If the -Knoauto option is to be specified at the same time as the -Kopenmp option or the -Kparallel option, it must be specified after the -Kopenmp option or the -Kparallel option.
- If the -Knoauto option is to be specified to an OpenMP program or an automatic parallelized program, the program may not run correctly.

Example of abnormal termination due to stack shortage

```
SUBROUTINE SUB(N)
  REAL(KIND=8),DIMENSION(N) :: A
  A=1.
  PRINT *,A(N)
END SUBROUTINE

CALL SUB(2000000)
END

$ ulimit -s
8192
$ frt a.f90
$ ./a.out
Segmentation exception (core dumped)
```

Normal termination with larger stack area

```
$ ulimit -s unlimited
$ ulimit -s
unlimited
$ frt a.f90
$ ./a.out
1.0000000000000000
```

Option Name	Default
-Keval -Knoeval	-Knoeval
Optimization Specifier	
EVAL NOEVAL	

## ● Function outline

- This option specifies whether to perform optimization to change an operation evaluation method for a source program.

## ● Effect

- It is expected to improve execution performance because changing the operation evaluation method reduces the number of operations (instructions) in a loop and promotes other optimization such as SIMD and automatic parallelization.

## ● Notes

- Calculation errors may be generated due to change in the evaluation order of operations.
- When the -Kfast option is specified, the -Keval option is induced, and calculation errors may be generated. In such a case, specify the -Knoeval option after the -Kfast option.
- When the -Keval option is specified, the -Kfsimple option, the -Kreduction option (when the -Kparallel option is valid), and the -Ksimd\_reduction\_product option (when -Ksimd[={1|2|auto} is valid) are induced. So, pay attention to side effects of those options.

## ● Examples

- Operations of two multiplication and one addition are changed to operations of one multiplication and one addition, reducing one multiplication.

Original source

```
!OCL EVAL
DO I=1,N
  A(I)=A(I)*B(I)+A(I)+C(I)
ENDDO
```

Sample source after optimization

```
DO I=1,N
  A(I)=A(I)*(B(I)+C(I))
ENDDO
```

- Optimization is performed to change the operation evaluation method in the target loop. There is no change in three additions on the source but "A(I)+B(I)" and "C(I)+D(I)" can be operated in parallel. So the execution performance may be improved.

Original source

```
!OCL EVAL
DO I=1,N
  A(I)=A(I)+B(I)+C(I)+D(I)
ENDDO
```

Sample source after optimization

```
DO I=1,N
  A(I)=(A(I)+B(I))+(C(I)+ D(I))
ENDDO
```

- A division is changed to a multiplication of the reciprocal. Execution performance may be improved by changing the operations to a multiplication instruction shorter in latency than a division instruction.

Original source

```
!OCL EVAL
DO I=1, N
  A(I)=B(I)/10
ENDDO
```

Sample source after optimization

```
TMP=1/10
DO I=1,N
  A(I)=B(I)*TMP
ENDDO
```

- Optimization by changing the operation evaluation method promotes parallelization of loops with reduction operations.

Original source

```
!OCL EVAL
DO I=1,10000
  SUM=SUM+A(I)
ENDDO
```

Sample source after optimization

```
(core 1)
SUM1=0
DO I=1,5000
  SUM1=SUM1+A(I)
ENDDO
```

```
(core 2)
SUM2=0
DO I=5001,10000
  SUM2=SUM2+A(I)
ENDDO
```

```
SUM=SUM+SUM1+SUM2
```



Option Name	Default
-Keval_concurrent -Keval_noconcurrent	-Keval_noconcurrent
Optimization Specifier	
EVAL_CONCURRENT EVAL_NOCONCURRENT	

## ● Function outline

- The option performs Tree-Height-Reduction optimization that enhances instruction-level parallelism by reordering operations in loops.
- This option operates when specified with the -O1 or above and the -Keval option.

## ● Effect

- Performance may be improved when it is applied to loops with small iteration counts and many operations unavailable for software pipelining.

## ● Point

- Tree-Height-Reduction optimization reorders operations in loops so that the height of the operation tree becomes as low as possible to enhance instruction-level parallelism.  
An operation tree expresses operation expressions in the form of tree structure according to the priority of operations. Values are placed at leaf parts.  
As for other joints, operators are placed. The higher the priority of an operator is, the upper the operator locates in the layers.

## ● Note

- It is necessary to make the -Keval option valid when this is applied to floating-point operations.

## ● Example

- This option instructs to prioritize instruction-level parallelism in tree-height-reduction optimization.

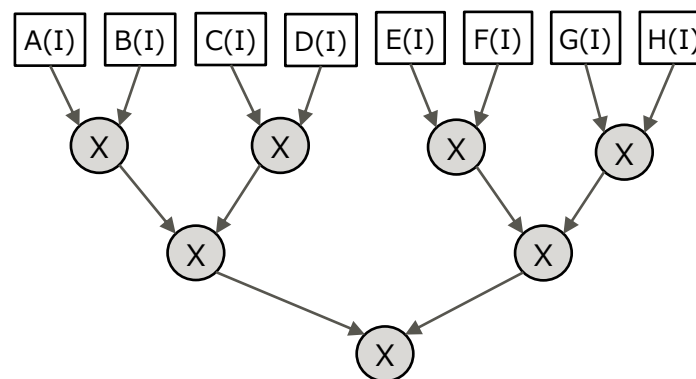
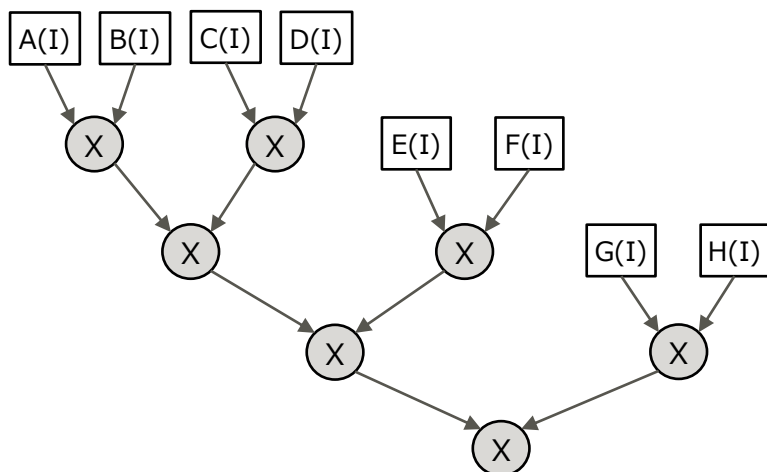
Original source

```
!OCL EVAL_CONCURRENT
DO I=1,N
  X(I)=A(I)*B(I)+C(I)*D(I)+E(I)*F(I)+G(I)*H(I)
ENDDO
```

Sample source after optimization

```
DO I=1,N
  X(I)=(A(I)*B(I)+C(I)*D(I))+(E(I)*F(I)+G(I)*H(I))
ENDDO
```

Operation tree before Tree-Height-Reduction optimization      Operation tree after Tree-Height-Reduction optimization



Option Name	Default
-Kfz -Knofz	-Kfz

## ● Function outline

- This option specifies whether to use flush-to-zero mode.
- In flush-to-zero mode, a denormal operation result or source operand is replaced by 0 with the original sign.

## ● Effect

- Denormal execution result or source operand does not affect execution performance.

## ● Notes

- With the `-Kfz` option, flush-to-zero mode is used. So, execution results and source operands to be a denormal number due to underflow are changed to 0.0 with the original sign.
- Generally, execution performance of operations becomes faster with the `-Kfz` option. With the `-Knofz` option, the execution result is guaranteed because a denormal number is expressed correctly, but execution performance may become slow.

## ● Example

- Specify the `-Kfz` option when flush-to-zero mode is used so that load instructions executed speculatively with the `-Kpreex` option or the `-Ksimd=2` option will not terminate abnormally.

Option Name	Default
-Kalign_commons -Knoalign_commons	-Kalign_commons

## ● Function outline

- This option specifies whether to perform 8-byte boundary alignment for 8-byte integer type, double precision real type, quadruple precision real type, double precision complex type, and quadruple precision complex type data in the processing to allocate variables belonging to the common block to the memory area.
- When this option is used, all the program units must be compiled with the  
-Knoalign\_commons option.

## ● Effect

- It is expected to improve execution performance because effective access to variables belonging to the common block becomes available.

## ● Note

- When 8-byte boundary alignment is performed, the size of the code increases because the compiler automatically inserts free space.

Option Name	Default
-Karray_declaration_opt -Knoarray_declaration_opt	-Knoarray_declaration_opt
Optimization Specifier	
ARRAY_DECLARATION_OPT NOARRAY_DECLARATION_OPT	

## ● Function outline

- In SIMDization of a loop with unknown iteration count, this option considers the number of elements in an array referenced in the loop as the maximum iteration count. When the maximum iteration count is smaller than the SIMD length, it does not create two loop structures of a loop iterating by the SIMD length and the remaining loops but creates only a single structure of the remaining loops.

## ● Effect

- It is expected to improve spill because the register to be used decreases for the decrease of the object.

## ● Note

- No execution result error occurs with the -Karray\_declaration\_opt option and the ARRAY\_DECLARATION\_OPT specifier.  
If an execution result error occurs, the program may be performing a loop iteration more than the array declaration. So, check the iteration count.

## ● Example

```
SUBROUTINE LOOP(A,B,C,N)
  REAL*4 A(10),B(10),C(10)
  !OCL ARRAY_DECLARATION_OPT
  DO I=1,N
    A(I) = B(I) + C(I)
  ENDDO
END
```

The maximum iteration count is judged as 10 because the numbers of elements in Array A(I), B(I), and C(I) are declared as 10 respectively.

Only the loop structure of the remaining loops is created because single precision type SIMD length (16)>10.

Option Name	Default
-Koptlib_string -Knootlib_string	-Knootlib_string

## ● Function outline

- This option links optimization libraries of string operation functions (bcopy, bzero, memchr, memcmp, memccpy, memcpy, memmove, memset, strcat, strcmp, strcpy, strlen, strncmp, strncpy, strncat) statistically.
- Specify this option at compilation and linkage.
- Together with the -Koptlib\_string option, specify the -KSVE option and the -KA64FX option.

## ● Effect

- Cite memcpy as an example. If the copy size is 16 to 128 KBytes, it is expected memcpy is performed 1.5 to 3.0 times faster than usual libc.

## ● Note

- In memcpy, -Kzfill is valid automatically when the copy size per thread exceeds 4 MiB. However, note that the performance degrades at a single thread execution (at non-memory busy). Also, it is expected to improve performance at thread parallel execution (at memory busy) but -Kzfill may not become valid even at memory busy depending on the copy size.

## Optimization Specifier

```
SCACHE_ISOLATE_WAY(L2=n1[,L1=n2])  
END_SCACHE_ISOLATE_WAY
```

### ● Function outline

- This specifier specifies the maximum number of ways for sector 1 in L1 cache and L2 cache.
  - Up to 4 ways for L1 cache and 14 ways for L2 cache can be specified.
- To specify this for a program unit, write the SCACHE\_ISOLATE\_WAY specifier at a description position for the program unit. The applicable range of the specification is the whole program.
- To specify this for a part of a program, specify the range by using the SCACHE\_ISOLATE\_WAY specifier and the END\_SCACHE\_ISOLATE\_WAY specifier.

### ● Effect

- By using the SCACHE\_ISOLATE\_ASSIGN specifier together, control is exercised so that data reusable in the loop will not be evicted from cache. Thus, it is expected to improve execution performance.

### ● Points

- Two or more sectors can be specified for either of L1D cache and L2 cache. The maximum number of sectors is four for L1D and two for L2.
- The maximum number of ways is a target value. The hardware exercises controls so that each sector becomes close to the specified number of ways at line replacement.
- Eviction in sectors is controlled by LRU algorithm (discarding least recently used data first).
- Application can determine the usage of sector 0 and 1. Note that a sequence of instructions are stored in sector 0.



## ● Notes

- In L2 cache, the assistant core uses two ways all the time. Therefore, the ranges specifiable for n1 and n2 are as follows.
  - $0 \leq n1 \leq$  "The maximum number of ways for L2 cache -2"
  - $0 \leq n2 \leq$  "The maximum number of ways for L1 cache"
- In CMG with the assistant core, a part of L2 cache (the amount of two ways = 1 MiB) is used for the assistant core. So, in CMG with the assistant core, the maximum number of ways for L2 cache is 14 and the size is 7 MiB.
- The range specification can be written in a part of a program with program unit specification, but the range specification cannot be nested.
- The name of this specifier in the old specification (K computer, FX100) is the CACHE\_SECTOR\_SIZE specifier. It is changed to the SCACHE\_ISOLATE\_WAY specifier in Fugaku.

## ● Example

- Access by Array b and Array c prevents reusable Array a from being evicted from cache.

Specification in the old specification (K computer, FX100)

```
!OCL CACHE_SECTOR_SIZE(4,10)
!OCL CACHE_SUBSECTOR_ASSIGN(A)
DO J=1,M
  DO I=1,N
    A(I) = A(I) + B(I,J) * C(I,J)
  ENDDO
ENDDO
!OCL END_CACHE_SUBSECTOR
!OCL END_CACHE_SECTOR_SIZE
```

Specification in Fugaku

```
!OCL SCACHE_ISOLATE_WAY(L2=10)
!OCL SCACHE_ISOLATE_ASSIGN(A)
DO J=1,M
  DO I=1,N
    A(I) = A(I) + B(I,J) * C(I,J)
  ENDDO
ENDDO
!OCL END_SCACHE_ISOLATE_ASSIGN
!OCL END_SCACHE_ISOLATE_WAY
```

## Optimization Specifier

```
SCACHE_ISOLATE_ASSIGN(array1[,array2]...)  
END_SCACHE_ISOLATE_ASSIGN
```

### ● Function outline

- This specifier specifies an array to be loaded on sector 1 of the cache.
- To specify this for a program unit, write the SCACHE\_ISOLATE\_ASSIGN specifier in a description position of the program unit. The applicable range of the specification is the whole program.
- To specify this for a part of a program, specify the range by using the SCACHE\_ISOLATE\_ASSIGN specifier and the END\_SCACHE\_ISOLATE\_ASSIGN specifier.
- This is valid only when a numeric type or a logical type array is specified.

### ● Effect

- By using the SCACHE\_ISOLATE\_WAY specifier together, control is exercised so that data reusable in the loop will not be evicted from cache. Thus, it is expected to improve execution performance.

### ● Notes

- The range specification can be written in a part of a program with program unit specification, but the range specification cannot be nested.
- The name of this specifier in the old specification (K computer, FX100) is the CACHE\_SUBSECTOR\_ASSIGN specifier.  
It is changed to the SCACHE\_ISOLATE\_ASSIGN specifier in Fugaku.

## ● Examples

- Access by Array b and Array c prevents reusable Array a from being evicted from cache.

Specification in the old specification (K computer, FX100)

```
!OCL CACHE_SECTOR_SIZE(4,10)
!OCL CACHE_SUBSECTOR_ASSIGN(A)
DO J=1,M
  DO I=1,N
    A(I) = A(I) + B(I,J) * C(I,J)
  ENDDO
ENDDO
!OCL END_CACHE_SUBSECTOR
!OCL END_CACHE_SECTOR_SIZE
```

Specification in Fugaku

```
!OCL SCACHE_ISOLATE_WAY(L2=10)
!OCL SCACHE_ISOLATE_ASSIGN(A)
DO J=1,M
  DO I=1,N
    A(I) = A(I) + B(I,J) * C(I,J)
  ENDDO
ENDDO
!OCL END_SCACHE_ISOLATE_ASSIGN
!OCL END_SCACHE_ISOLATE_WAY
```

Option Name	Default
-Klto -Knolto	-Knolto

## ● Function outline

- This option performs optimization at linkage.
- This has meaning when -O1 or above is valid.

## ● Notes

- This needs to be specified at program compilation and linkage.
- When the -g option or the -Ncoverage option is valid, the -Klto option is invalid.
- When the -Klto option and the -xdir=dir\_name option are specified together, the -xdir=dir\_name option is invalid.

Option Name	Default
-Khpctag -Knohpctag	-Khpctag

## ● Function outline

- This option specifies whether to enable optimization of a compiler using tags by using the HPC tag address override function of A64FX processor.
- The HPC tag address override function is a function to improve performance by using information set in upper eight bits of the address (tag).  
The compiler sets tags at optimization of prefetch and sector cache control.
- To suppress the function, specify the -Knohpctag option.
- This has meaning when the -KA64FX option is valid.

## ● Effect

- The HPC tag address override function enables the sector cache function and the hardware prefetch assist function (hardware prefetch function for stride access, etc.). So, it is expected to improve execution performance.

## ● Notes

- This needs to be specified at program compilation and linkage.
- Tags take the exclusive OR. So, execution performance may degrade when an application overwrites tags because tags are interpreted by mistake as information on prefetch or sector cache control.  
In such a case, the HPC tag address overwrite function needs to be disabled.

Option Name	Default
-Nreordered_variable_stack -Nnoreordered_variable_stack	-Nnoreordered_variable_stack

## ● Function outline

- This option specifies whether AUTOMATIC variables are allocated in the stack area in ascending order of the data size.

## ● Effect

- AUTOMATIC variables in the stack are allocated in ascending order of the data size. Data items with the same size are allocated in ascending order of alignment. Data items with the same data size and alignment are allocated in the order of the declaration statements written in the source program.  
Allocating AUTOMATIC variables in ascending order of the data size can reduce the stack of the whole program.

## ● Note

- If the Nnoline and -g0 options are valid, the allocation order is not guaranteed.

Option Name	Default
-Ncoverage -Nnocoverage	-Nnocoverage

## ● Function outline

- This option specifies whether to create information to use the code coverage function.

## ● Effect

- Information to use the code coverage function is generated.

## ● Point

- -Ncoverage option needs to be specified at program compilation and linkage.

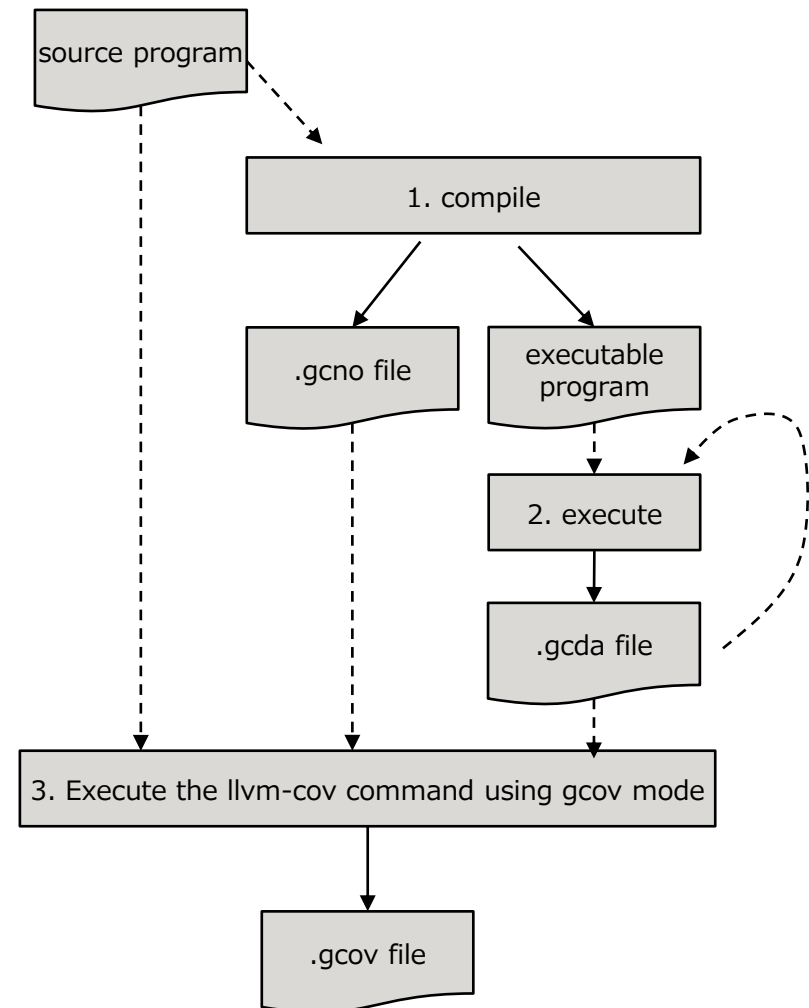
## ● Notes

- Execution performance may degrade because instructions to measure the execution count are added.
- This is not available for programs that require the -Kcmodel=large option.
- The execution count may not be measured correctly under any of the following conditions.
  - Two or more executable statements are written in one line.
  - A STOP statement, an ERROR STOP statement, the EXIT service subroutine, and the SETRCD subroutine are called.
  - An exception is captured.
  - The source program contains a #line instruction.
  - The compiler optimization is applied.

## ● Example

- Use the code coverage function following the procedure below.

1. Compile (-Ncoverage)
2. Execute
3. Execute the llvm-cov command using gcov mode



.gcno file	A binary file with information obtained during compilation
.gcda file	A binary file with information obtained during execution



Option Name	Default
-Kopenmp_collapse_except_innermost -Kopenmp_nocollapse_except_innermost	-Kopenmp_nocollapse_except_innermost

## ● Function outline

- When a loop struct in OpenMP satisfies both of the following conditions, the inner-most loop is excluded from the target of the COLLAPSE specifier.
  - The COLLAPSE specifier covering up to the inner-most loop is specified.
  - The compiler can judge at compilation that execution performance may degrade due to the COLLAPSE specifier covering up to the inner-most loop.
- This has meaning when the -Kopenmp option is valid.

## ● Effect

- SIMDization of the inner-most loop is not disturbed, so degradation of execution performance due to COLLAPSE specifier may be prevented.

## ● Example

```
38 !$OMP PARALLEL
39 !$OMP DO PRIVATE(K,J,I) COLLAPSE(3)
40   DO K=1,L           ! L = 2
41     DO J=1,M         ! M = 512
42       DO I=1,N       ! N = 32
43         A(I,J,K)=B(I,J,K)+C(I,J,K)
44       ENDDO
45     ENDDO
46   ENDDO
47 !$OMP END DO
48 !$OMP END PARALLEL
```

When -Kopenmp\_nocollapse\_except\_innermost is valid

line 39: Execution performance may degrade because the COLLAPSE specifier covering up to the inner-most loop is specified.

When -Kopenmp\_collapse\_except\_innermost is valid

line 39: The inner-most loop is excluded from the target of COLLAPSE.

Option Name	Default
-Nfjomplib	-Nlibomp
-Nlibomp	

## ● Function outline

- This option specifies the library to be used in parallelization processing.
- When the -Nfjomplib option is specified, Fujitsu OpenMP library is used in the parallelization processing.
- When the -Nlibomp option is specified, LLVM OpenMP library is used in the parallelization processing.

OpenMP library	Option	Supported Function
LLVM OpenMP library	-Nlibomp (default)	A part of OpenMP4.5 and 5.0 Hardware barrier (default: software barrier) Sector cache Core bind by default
Fujitsu OpenMP library	-Nfjomplib	OpenMP3.1 Hardware barrier Sector cache

## ● Point

- This needs to be specified at linkage.

## ● Note

- The library selection option is valid even with automatic parallelization (the -Kparallel option).

- Purpose

- To adjust optimization according to the characteristics of the program.

- Points

- Optimization is performed based on the following policy.

Argument	Optimization Policy
shortloop	<ul style="list-style-type: none"><li>• If the iteration count of the inner-most loop is unknown at the time of compilation, it is regarded as large.</li></ul>
memory_bandwidth	<ul style="list-style-type: none"><li>• In the inner-most loop, the memory bandwidth is not regarded as the bottleneck and the bottleneck of the CPU operation is resolved in priority.</li></ul>
time_saving_compilation	<ul style="list-style-type: none"><li>• In programs that consume compilation time, too, making executable programs faster has priority.</li></ul>

- Two or more -Kassume options can be specified together.

Option Name	Default
-Kassume={shortloop noshortloop}	-Kassume=noshortloop

Optimization Specifier
ASSUME({SHORTLOOP NOSHORTLOOP})

## ● Function outline

- This option performs optimization regarding the iteration count of the inner-most loop in the program is small if it is unknown at the time of compilation.
- This option may adjust or suppress optimization such as automatic parallelization, loop unrolling, and software pipelining.

Optimization	Optimization Behavior
Automatic parallelization	• The inner-most loop is suppressed.
Loop blocking	• The inner-most loop blocking is suppressed.
zfill	• Suppressed
prefetch	• The inner-most loop is suppressed.
Loop unrolling	• Suppressed
Loop redirection	• Suppressed due to the suppressed loop unrolling.
Outer loop unrolling	• Suppressed
Software pipelining	• Performs flexible software pipelining.

- This option is valid with -O1 or above.

## ● Effect

- It is expected to improve execution performance because optimization appropriate for a loop with a small iteration count.

# ASSUME(MEMORY\_BANDWIDTH)

Option Name	Default
-Kassume={memory_bandwidth nomemory_bandwidth}	-Kassume=nomemory_bandwidth

Optimization Specifier
ASSUME({MEMORY_BANDWIDTH NOMEMORY_BANDWIDTH})

## ● Function outline

- This option performs optimization assuming that the memory bandwidth becomes a bottleneck in the inner-most loop of the program.
- zfill optimization promotion and optimization such as software pipelining may be adjusted or suppressed.

Optimization	Optimization Behavior
Loop fusion	• Suppressed
Loop fission	• Reduces the number of streams per loop.
zfill	• Executed proactively
prefetch	• Controls the prefetch width dynamically.
Software pipelining	• Performs simple software pipelining.

- This option is valid with -O1 or above.

## ● Effect

- It is expected to improve execution performance by optimization to promote locality of data aiming at reducing memory bandwidth usage and cache-related optimization.

Option Name	Default
-Kassume={time_saving_compilation notime_saving_compilation}	-Kassume=notime_saving_compilation
Optimization Specifier	
ASSUME({TIME_SAVING_COMPILATION NOTIME_SAVING_COMPILATION})	

## ● Function outline

- This option performs optimization to shorten the compilation time.

Optimization	Optimization Behavior
Inline expansion	<ul style="list-style-type: none"> <li>• Controls thresholds and prevents instructions from increasing due to too much inline expansion.</li> </ul>
Loop fusion	<ul style="list-style-type: none"> <li>• Suppressed</li> </ul>
Loop unswitching	<ul style="list-style-type: none"> <li>• Suppressed except OCL.</li> </ul>
Loop unrolling	<ul style="list-style-type: none"> <li>• Narrows down unrolling target loops into cases where unrolling is effective.</li> </ul>

- This option is valid with -O1 or above.

## ● Effect

- It is expected to improve compilation performance because compilation time is saved by controlling specific optimization behavior and suppression.

## ● Point

- This option does not stop a specific function like level control of -O option but restricts and suppresses optimization as the program becomes huger.

# Options That Require Attentions (Side Effects of Optimization)

- Options That May Affect Execution
- Options That May Accompany Calculation Errors
- Options That Suppress Calculation Errors

# Options That May Affect Execution

Abnormal termination may occur during execution because move instructions are performed with speculative executions

Option	Function	Side Effect of Optimization	How to Bypass
-Kpreex	Performs preceding evaluation on invariants. (Moves an invariant in an IF statement in the loop to a location outside the loop.)	Abnormal termination may occur during execution.	This can be bypassed by either of the following. <ul style="list-style-type: none"><li>• Delete the -Kpreex option.</li><li>• Specify the -Knopreex option at a location after the -Kpreex option.</li></ul>
-Ksimd=2	SIMDizes IF constructs. (Executes speculatively an expression in an IF construct.)		This can be bypassed by either of the following. <ul style="list-style-type: none"><li>• Change this to the -Ksimd=1 option.</li><li>• Specify the -Knosimd at a location after -Ksimd=2.</li></ul>
-Kpreload	Executes load instructions speculatively. (Moves a load instruction in an IF statement in the loop to a location before the IF statement.)		This can be bypassed by either of the following. <ul style="list-style-type: none"><li>• Delete the -Kpreload option.</li><li>• Specify -Knopreload option at a location after the -Kpreload option.</li></ul>



# Options That May Accompany Calculation Errors

The following options may cause calculation errors.

Option	Function	Side Effect of Optimization	How to Bypass
-Keval	$x*y + x*z \Rightarrow x*(y+z)$ <p>Operation evaluation method is changed as shown above. When -Kfast is specified, -Keval is induced.</p>	Calculation errors may occur due to change in operation evaluation method.	<ul style="list-style-type: none"> <li>Specify -Knoeval in a location after -Kfast.</li> </ul>
-Kfp_contract	Issues multiply add/subtract floating-point instructions. When -Kfast is specified, -Kfp_contract is induced.	Calculation errors may occur because multiply add/subtract floating-point instructions are issued.	<ul style="list-style-type: none"> <li>Specify -Knofp_contract in a position after -Kfast.</li> </ul>
-Kfp_relaxed	Calculates floating-point division or SQRT function by using reciprocal approximation. When -Kfast is specified, -Kfp_relaxed is induced.	Calculation errors may occur because reciprocal approximation instructions are issued.	<ul style="list-style-type: none"> <li>Specify -Knofp_relaxed in a position after -Kfast.</li> </ul>
-Kilfunc	Performs inline expansion on intrinsic functions. When -Kfast is specified, -Kilfunc is induced.	Calculation errors may occur because inline expansion of intrinsic functions applies an algorithm using the reciprocal approximation operation instruction, the trigonometric function auxiliary instruction, etc. atan2 is calculated under the Fortran95 language specification even when -X03 or -X08 is specified.	<ul style="list-style-type: none"> <li>Specify -Knoilfunc in a position after -Kfast.</li> </ul>

# Options That Suppress Calculation Errors (1/2)

Calculation errors can be suppressed by specifying the following options.

Option	Function
-Kfp_precision	<p>Induces the combination of the following options where floating-point operation does not generate calculation errors.</p> <ul style="list-style-type: none"><li>• -Knoeval (-Knosimple, -Knoredution, and -Ksimd_noredution_product are induced)</li><li>• -Knofp_contract</li><li>• -Knofp_relaxed</li><li>• -Knofz</li><li>• -Knoifunc</li><li>• -Knomfunc</li><li>• -Kparallel_fp_precision(-Kopenmp_ordered_reduction is induced)</li></ul> <p>-Knofp_precision disables only the -Kfp_precision specification.</p> <p>Options are analyzed in the following order. So, even when individual options induced by -Kfp_precision are specified separately, the options specified separately are not affected.</p> <ol style="list-style-type: none"><li>1. Which option of -Kfp_precision or -Knofp_precision is valid is analyzed.</li><li>2. A combination of options at the location where -Kfp_precision is specified if Kfp_precision is valid is expanded.</li></ol> <p>To disable -Kfp_precision set in the profile at compilation, specify -Knofp_precision.</p>

# Options That Suppress Calculation Errors (2/2)

Option	Function
-Kparallel_fp_precision	<p>The compiler suppresses optimization when a calculation error is generated in a floating-point type or complex type operation result due to the difference in the number of thread parallels. Execution performance may degrade because a part of optimization is suppressed.</p> <p>When a REDUCTION clause of OpenMP is specified, the operation evaluation order is changed. So, calculation error may be generated even if -Kparallel_fp_precision is specified.</p> <p>Specify -Kparallel_fp_precision to create an execution program where calculation errors are not generated due to the difference in the number of thread parallels.</p> <p>Specify -Kparallel_nofp_precision if calculation errors due to the difference in the number of thread parallels does not matter.</p>

# Compilation Information

- Diagnosis Message/Guidance Message
- Contents of Compilation Information
- Lister (Program List/Optimization Information/Statistical Information)
- Notes of Compilation Information

## ■ Diagnosis message

A diagnosis message is output under any of the following conditions.

- The operand specified with the compilation command contains an error.
- The program contains an error.
- There is information useful for users.
- There are notes.

Diagnosis message from the compilation command

**frtpx: *message body***

Diagnosis message from the compiler **jwdxxxxz-y *file name* *line number digit position* *message body***

## ■ Guidance message

When the compile option `-Koptmsg=guide` is specified, a guidance message regarding the following optimization (the cause of optimization failure and actions to take) is output.

- SIMDization
- Automatic parallelization
- Software pipelining
- Inline expansion

jwdxxxxz-y diagnosis message

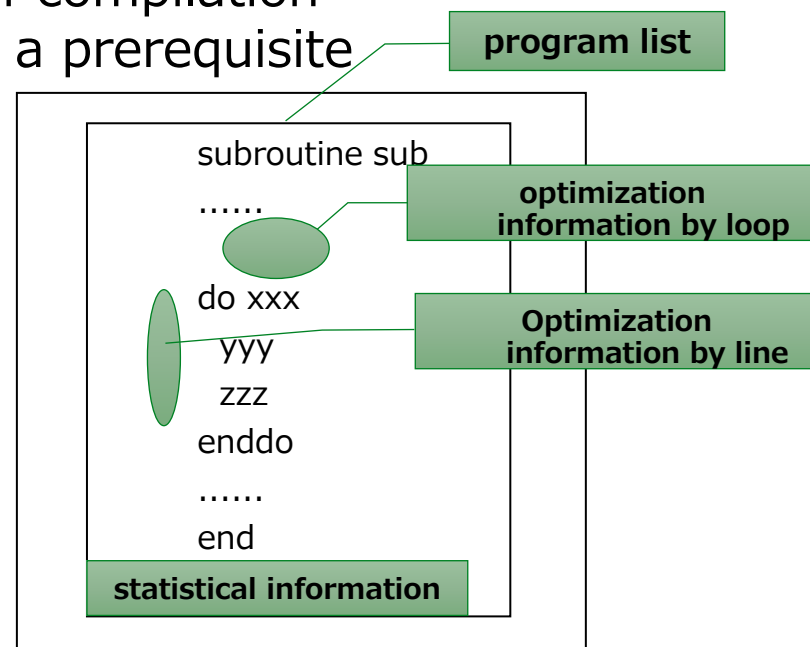
**[Guidance]**

***Guidance message body***

The following describes the contents of compilation information and an output example as a prerequisite knowledge for compiler optimization.

The following information is output as compiler information.

- Program list
- Optimization information by loop
- Optimization information by line
- Error messages



## Compiler option format:

`-Nlst[=lst_arg]    lst_arg: { a | d | i | m | p | t | x }`

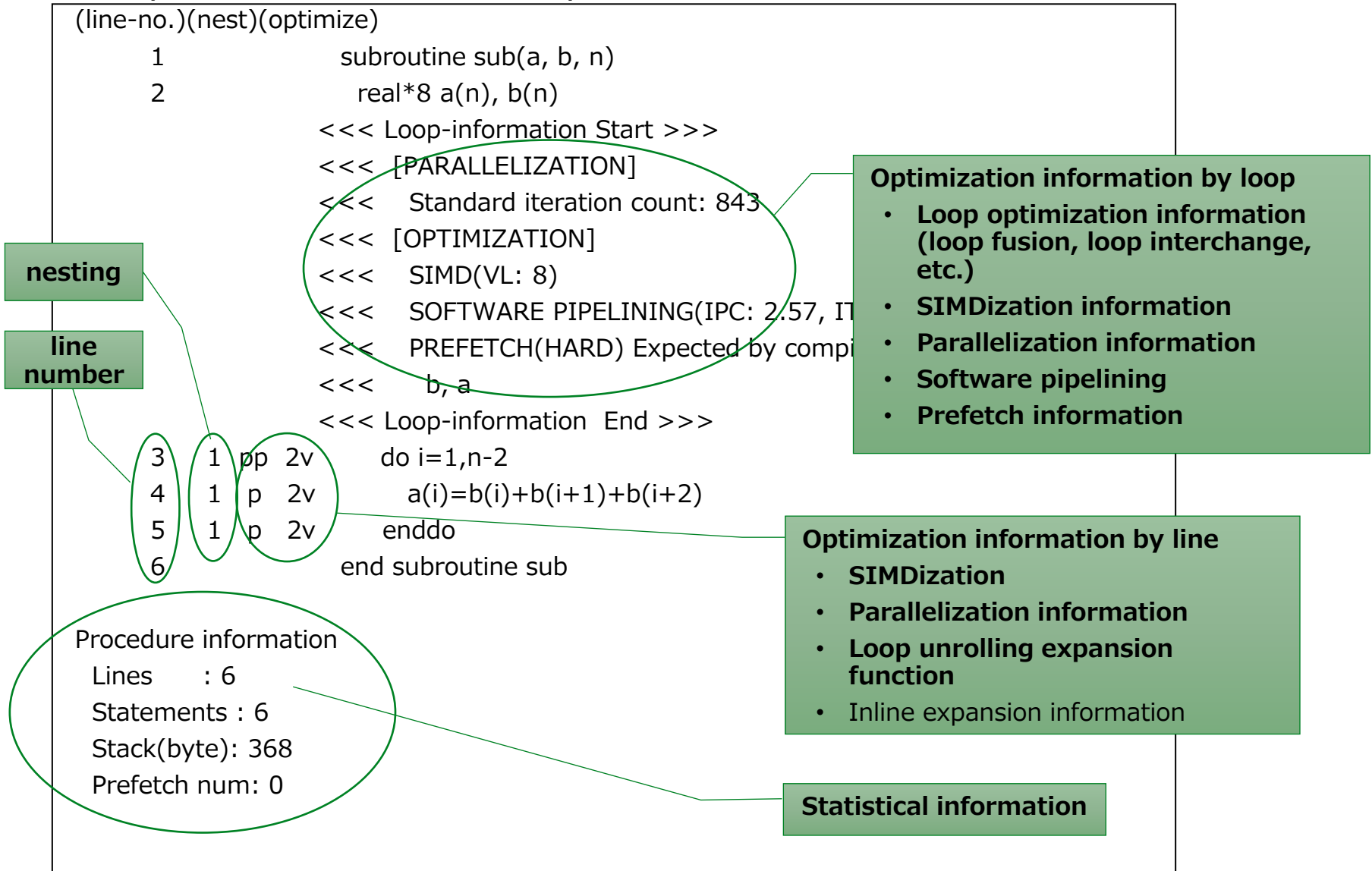
Compilation example

```
$ frtpx -Kfast,parallel -Nlst=t sample.f90
```

-> Outputs sample.lst as compilation information.

# Lister (Program List/Optimization Information/Statistical Information)

## ■ Compilation information output format



- Compilation information (optimization information) output by the compile option `-Nlst=p` or `-Nlst=t` may be output incorrectly as shown below.
  - When inline expansion is performed, optimization behavior may vary by the location of inline expansion.
    - The output may be as follows.
    - Two or more optimization messages are output for a single loop.
    - An optimization message with a meaning opposite to compilation information (optimization information) is output.
    - Compilation information (optimization information) is not output.
    - The prefetch count is output in total for functions with inline expansion performed.
  - Two or more optimizations such as SIMDization, automatic parallelization, loop fusion, and loop fission that is output as detailed optimization are applied to a single loop.
    - The output may be as follows.
    - Optimization information by line number is shifted.
    - An optimization message with a meaning opposite to compilation information (optimization information) is output.
  - Loop unswitching optimization generates both loops where IF construct conditions are met and loops where IF construct conditions are not met. Compilation information and an optimization message on any of those loops are output. Also, optimization information by line number may not be output.
  - When two or more loops are written in a single line, optimization information on any of those loops is output. Prevent loops from being written in a single line as much as possible if you want optimization information on the loops.
  - Detailed optimization information on loops with condition statement and GOTO statement is not output. Optimization information by line numbers may also be shifted.



- The compiler may generate a loop under any of the following conditions.  
Compilation information (optimization information) output by the compilation option `-Nlst=p` or `-Nlst=t` and an optimization message may be output to the generated loop.
  - The actual argument is a pointer array, an assumed-shape array, or an array section and the corresponding dummy argument is neither a pointer nor an assumed-shape array.
  - The compilation option `-Nquickdbg=undef`, `-Nquickdbg=undefnan`, or `-Nsetvalue=array` is valid.
  - A variable name of an array appears in `FIRSTPRIVATE`, `LASTPRIVATE`, `REDUCTION`, `COPYIN`, or `COPYPRIVATE` clause of OpenMP.
  - There is a loop with automatic parallelization performed by the compile option `-Karray_private` or the optimization specifier `ARRAY_PRIVATE`, `FIRST_PRIVATE`, or `LAST_PRIVATE`.
- When link-time optimization is applied, optimization different from the one shown in compilation information may be applied at the time of execution.
- When the source program contains a `#line` directive, compilation information (optimization information) output by the compile option `-Nlst=p` or `-Nlst=t` is output based on the line number specified in the `#line` directive.  
For example, if a DO statement in line 10 in the source program is specified as line 3 by the `#line` directive, compilation information (optimization information) of the DO statement is output for line 3 in the source program.

# Executing a Program

- Execution Commands
- Parallel Thread Execution
- Combination of OpenMP Libraries
- LLVM OpenMP Library and Fujitsu OpenMP Library
  - LLVM OpenMP Library (-Nlibomp)
  - Fujitsu OpenMP Library (-Nfjomplib)

## ■ For sequential execution

The executable program generated at compilation is executed.

```
./[Executable programs]
```

Example: Script for sequential execution

```
#!/bin/sh
#PJM -L "node=1"                # Number of nodes

./a.out
```

## ■ For thread parallel (automatic parallelization, OpenMP) execution

The number of threads to operate in parallel is specified in the environment variable OMP\_NUM\_THREADS and the executable program is executed.

```
OMP_NUM_THREADS={the number of threads} ;export OMP_NUM_THREADS
./[executable program]
```

Example: Script for thread parallel (automatic parallelization or OpenMP) execution

```
#!/bin/sh
#PJM -L "node=1"                # Number of nodes
OMP_NUM_THREADS=16 ;export OMP_NUM_THREADS

./a.out
```

\* The standard output and the standard error output of the job are output to the following files.  
{pjm-script}.{req-id}.out: Standard output  
{pjm-script}.{req-id}.err: Standard error output  
pjm-script: PJM script name, req-id: Request number

- Two parallel thread libraries:
  - LLVM OpenMP library
  - Fujitsu OpenMP
- The following compile options determine which library to use.

Option Format	Function Outline
-Nfjomplib	Uses Fujitsu OpenMP library. Up to OpenMP 3.1 is available.
-Nlibomp	Uses LLVM OpenMP library. Up to OpenMP 4.5 and a part of OpenMP 5.0 are available.

## ■ Notes

- When -Nclang and -Nfjomplib are specified together, a warning is output and LLVM OpenMP library is linked.  
(a process corresponding to -Nlibomp)
- When a function of OpenMP 4.0 or later is used with Fortran and -Nfjomplib is specified, a linkage error occurs.
- If -Njomplib is specified at mixed language linkage of clang mode object and Fortran, a link error occurs.

# Combination of OpenMP Libraries

- The following shows combinations available for OpenMP or automatic parallelization.
  - Available OpenMP specifications are shown.
  - The precondition is that the parallelization option `-Kopenmp` or `-Kparallel` is specified.

		LLVM OpenMP Library (-Nlibomp)	Fujitsu OpenMP Library (-Nfjomplib)
C/C++ compiler	trad mode	OpenMP 3.1 a part of OpenMP 4.5 (Note 1)	OpenMP 3.1 a part of OpenMP 4.5 (Note 4)
	clang mode (-Nclang)	OpenMP 4.5 (Note 2) a part of OpenMP 5.0 (Note 3)	Unavailable
Fortran compiler	-	OpenMP 4.5 a part of OpenMP 5.0 (Note 3)	OpenMP 3.1 a part of OpenMP 4.5 (Note 4)

Note 1: The following functions of OpenMP 4.5 are available.

- `simd` construct
- `declare simd` construct
- `proc_bind` clause of `parallel` construct
- `depend` clause of `task` construct
- `taskgroup` construct

Note 2: The following functions of OpenMP 4.5 are not available.

- `declare simd` construct

Note 3: The following functions of OpenMP 5.0 are available.

- `in_reduction` clause of `task` construct
- `task_reduction` clause of `taskgroup` construct
- `reduction` clause and `in_reduction` clause of `taskloop` construct

Note 4: The following functions of OpenMP 4.5 are available.

- `simd` construct
- `declare simd` construct

# LLVM OpenMP Library and Fujitsu OpenMP Library

- The LLVM OpenMP library is an OpenMP library based on the LLVM OpenMP Runtime Library extended for the A64FX.

OpenMP Library	Option	Supported Functions
LLVM OpenMP library	-Nlibomp (default)	OpenMP 4.5 and Parts of 5.0 Hardware barrier( <u>Default is Software barrier</u> ) Sector cache Bind to core (default)
Fujitsu OpenMP library	-Nfjomplib	OpenMP 3.1 Hardware barrier Sector cache Bind to core (Default when execute on job)

- Selection method
  - Specify in compiler option when linking.
    - -Nlibomp (default) : Use LLVM OpenMP Library
    - -Nfjomplib : Use Fujitsu OpenMP Library

- Difference in specifications

- Thread stack size

Option	Default size	Environment variables for resizing
-Nlibomp	• 8MiB	OMP_STACKSIZE
-Nfjomplib	• Inherit the process stack size. • If the stack size of the process is specified as unlimited. (Memory size / Number of threads) / 5	OMP_STACKSIZE or THREAD_STACK_SIZE

# LLVM OpenMP Library (-Nlibomp) (1/2)

- By default, the **software barrier**/sector cache is available.
- To use the hardware barrier, specify the FLIB\_BARRIER environment variable.
  - **FLIB\_BARRIER=HARD : Use the hardware barrier.**
  - FLIB\_BARRIER=SOFT : Use the software barrier.(**default**)
- Precautions on using the hardware barrier
  - The number of threads (omp\_set\_num\_threads(), num\_threads clause) cannot be controlled.
  - Thread affinity (proc\_bind clause, OMP\_PLACES, OMP\_PROC\_BIND) cannot be controlled.
  - Nesting (omp\_set\_nested(), OMP\_NESTED) cannot be controlled.
  - An undeferred task is always generated for a task construct, and the task will not be parallelized.
  - Cancellation is disabled.
- Supports **part of OpenMP 5.0 in addition to OpenMP 4.5.** Use this library to use the latest OpenMP standard.
- Select the software barrier when using the main functions of OpenMP 4.5/5.0, such as a task or cancellation.

# LLVM OpenMP Library (-Nlibomp) (2/2)

- To bind the initial thread to a specific core, use numactl or taskset. This does not affect the upper limit on the number of threads in the program.

```
#!/bin/sh -ex
:
export FLIB_BARRIER=HARD # Use the hardware barrier
                        #(Use the software barrier when FLIB_BARRIER is SOFT or not set)
numactl -C12 ./a.out     # Fix the initial thread to C12
```

- For MPI programs, you can use [the function of parallelizing memory copy processing in MPI library with threads](#). Specify -Nlibomp and at least one of the -Kparallel and -Kopenmp as options of a compilation/linkage command.
  - mpifrtpx [-Kopenmp -Nlibomp](#) a.f90
- The following environment variables specific to the Fujitsu OpenMP library cannot be used (and will be ignored):

- PARALLEL
- FLIB\_FASTOMP
- THREAD\_STACK\_SIZE
- FLIB\_SPINWAIT
- FLIB\_CPU\_AFFINITY
- FLIB\_NOHARDBARRIER

- FLIB\_HARDBARRIER\_MESSAGE
- FLIB\_CNTL\_BARRIER\_ERR
- FLIB\_PTHREAD
- FLIB\_CPUBIND
- FLIB\_USE\_ALLCPU
- FLIB\_USE\_CPURESOURCE



- Details are omitted since they are the same for existing systems (K computer/FX100).
- By default, the hardware barrier/sector cache is available.
- There is no restriction on generating undeferred tasks while using the hardware barrier.
- OpenMP 3.1 support.
- Specify -Nfjomplib when compiling with the C/C++ compiler in Trad Mode or with the Fortran compiler.
  - Fortran
    - frtpx -Kopenmp -Nfjomplib main.f90
  - C/C++ Trad Mode
    - fccpx -Kopenmp -Nfjomplib main.c
- -Nfjomplib cannot be used with the C/C++ compiler in Clang Mode.
- To bind to core when execute on job. The environment variable FLIB\_CPU\_AFFINITY can be used to control the bind to core when not execute on job.
- The following environment variable specific to the LLVM OpenMP library cannot be used (and will be ignored):
  - FLIB BARRIER

# Notes of Developing Programs

- Conditions for SIMDizable Loops
- Conditions for Automatic Parallelizable Loops
- Fujitsu Extension Object Specification
- SIMDization of the Reduction Operation Using the bit Operator
- Large Page
- Notes on transformational function reference of intrinsic functions

- The iteration count of the loop is determined before the loop is executed.
  - SIMDization may become available by using the `-Ksimd_uncounted_loop` option together.
- There is no branch (such as an IF statement) in the loop.
  - Masked SIMD may become effective depending on, for example, the TRUE rate of the IF statement.
- There is no subroutine call in the loop
  - SIMDization may become available by using inline expansion together.
- There is no overlapped array area between the right side and the left side of an expression in the loop.
- The calculation at the n-th iteration of the loop does not depend on the result of the calculation at the n-1-th iteration.

# Conditions for Automatic-Parallelizable Loops (1/3)

- DO loops (including nested DO loops) and array operating statements (array expression , array assignment)

- The following cases are excluded from the automatic parallelization target.

① Parallel execution is not expected to shorten execution time.

! Do loops with the small iteration count and the small number of operations are  
! out of automated parallelization target.

```
DO I=1, 10
  A(I) = A(I) + B(I)
ENDDO
```

② Operations of types that are out of the loop slice target are included.

Loops with operations of the following types are not automatically parallelized. Same is the case when such operations are included in internal DO loops.

- Character type
- Derived type

③ Reference to external procedures, internal procedures, or module procedures is included.

! Loops including reference to subroutines are out of automatic parallelization  
! target.

```
DO J=1,10
  DO I=1,10000
    A(I,J) = A(I,J) + B(I,J)
    CALL SUB(A)
  ENDDO
```

# Conditions for Automatically Parallelizable Loops (2/3)

- ④ The shape of the DO loop is complicated.

The shapes of Do loops shown below do not become the target of automatic parallelization.

- DO loop with jump from inside of the DO loop to the outside

```
DO J=1,10
  DO I=1,10000
    A(I,J) = A(I,J) + B(I,J)
    IF (A(I,J).LT.0) GOTO 20
  ENDDO
ENDDO
...
20 CONTINUE
```

- DO loop in a complicated structure

```
DO J=1,10000
  IF(N>0) THEN
    ASSIGN 10 TO I
  ELSE IF(N<0) THEN
    ASSIGN 20 TO I
  ELSE
    ASSIGN 30 TO I
  ENDIF
  GOTO I,(10,20,30)
10 A(J)=A(J)+0
20 A(J)=A(J)+1
30 A(J)=A(J)+2
ENDDO
```

# Conditions for Automatic Parallelizable Loops (3/3)

- ⑤ Loops with the input/output statement or an intrinsic subroutine  
DO loops with the input/output statement or an intrinsic subroutine are out of automatic parallelization target.
- ⑥ Loops with a data definition reference order such that it may change with sequential execution  
DO loops with a data definition reference order such that it changes with sequential execution are not targets of automatic parallelization.

```
DO I=2,10000  
  A(I) = A(I-1) + B(I)  
ENDDO
```

## ● Patterns of objects by compiler options

ARMv8 (NEON)		Without Fujitsu extension	-KGENERIC_CPU -KNOSVE
		With Fujitsu extension	-KA64FX -KNOSVE
ARMv8+SVE	Variable Length	Without Fujitsu extension	-KSVE -Ksimd_reg_size=agnostic
		With Fujitsu extension	-KA64FX -Ksimd_reg_size=agnostic
	Fixed Length	Without Fujitsu extension	-KSVE
		With Fujitsu extension	-KA64FX

- -KA64FX
  - instructs to output object files for A64FX processors.
- -KGENERIC\_CPU
  - Instructs to output object files for Arm processors.
- -Ksimd\_reg\_size=agnostic
  - Compiles programs without considering SVE vector registers as a specific size and creates executable programs that determine SVE vector register size at the time of execution.
- -KSVE
  - Instructs to output object files that use SVE, which is an extension of Armv8-A architecture.
- -KNOSVE
  - Instructs to output object files that do not use SVE.

# SIMDization of the Redirection Operation Using the bit Operator

- bit operator to be redirected
  - Execution performance improvement is attempted by making the target of simdization in addition to the redirection operation of ADD, MULT, MAX, and MIN, redirection operation using the bit operator.
  - The following patterns are recognized as the redirection operation and made to be simdization target.
    - $A = \text{IAND}(A, B)$
    - $A = \text{IOR}(A, B)$
    - $A = \text{IEOR}(A, B)$
    - $A = \text{IALL}(B) \text{ (*1)}$
    - $A = \text{IANY}(B) \text{ (*1)}$
    - $A = \text{IPARITY}(B) \text{ (*1)}$
    - $A = \text{ALL}(B) \text{ (*1)}$
    - $A = \text{ANY}(B) \text{ (*1)}$

(\*1) Recognized as the redirection operation when inline expansion is performed
  - If the size of A is smaller than the size of B, it becomes out of the target due to the possible overflow.
  - Signed integer type, unsigned integer type, and Boolean type are the target.



## ■ What is a large page?

■ **Allocating memory (large page) with a larger page size** than a normal page to applications handling data on a larger scale has the following effects:

- Reduces the overhead incurred by the CPU address translation process
- Improves memory access performance

■ In the A64FX system environment, the normal page size is 64 KiB, and the size available as a large page is 2 MiB.

- You can set the following operations by setting environment variables:
  - ✓ Enabling/Disabling the large page allocation operation
  - ✓ Enabling/Disabling the large page allocation operation for the stack area
  - ✓ Selecting the paging method (page allocation trigger) for each memory area
- The various page sizes that can be used with McKernel 32 MiB, 1 GiB, 16 GiB and etc.

## ● Large page specifications

Memory Area	MP10/FX10/ FX100	A64FX			
	Page Size	Page Size			Paging (Default is <u>Underlined</u> )
		Normal Page	Large Page Base	Large Page Base+Stack (Default)	
Text (.text)	8 KiB	64 KiB	64 KiB	64 KiB	-
Static data (.data)	4 MiB (default), 8 KiB, 32 MiB, 256 MiB	64 KiB	2 MiB	2 MiB	Always prepage
Static data (.bss)		64 KiB	2 MiB	2 MiB	demand   <u>prepage</u>
Stack (*1)		64 KiB	64 KiB	2 MiB	<u>demand</u>   prepage
Dynamic memory (*2)		64 KiB	2 MiB	2 MiB	demand   <u>prepage</u>
Shared memory		64 KiB	64 KiB	64 KiB	-

\*1 This covers the process stack/main thread stack/thread stack area.

\*2 This covers the process heap/main thread heap/thread heap/mmap area.

- Environment Variables for Large Page Settings

- Basic settings/Paging method settings

Environment Variable Name	Specified Value (Default is <u>Underlined</u> )	Description
XOS_MMM_L_HPAGE_TYPE	<u>hugetlbfs</u>   none	With this setting, select whether to enable or disable the operation of large page allocation using the large page library. "hugetlbfs" enables large pages using HugeTLBfs. "none" does not enable large pages using the large page library.
XOS_MMM_L_LPG_MODE	<u>base+stack</u>   base	With this setting, select whether to enable or disable the operation of large page allocation for the stack region and thread stack region. "base+stack" enables large pages not only for the static data and Dynamically allocated memory region but also for the stack region and thread stack region. "base" enables large pages only for the static data and dynamic memory storage region. Large pages are not enabled for the stack region and thread stack region.
XOS_MMM_L_PAGING_POLICY	[demand   <u>prepage</u> ]: [demand   <u>prepage</u> ]: [demand   <u>prepage</u> ]	With this setting, select the paging method (page allocation trigger) for each memory region. "demand" represents the demand paging method, and "prepage" represents the prepaging method. This variable specifies the paging method for three memory region delimited by a colon (:). The first specified method applies to the .bss region of static region. (This specified paging method does not cover the .data region of static data. "prepage" is always the value for this region.) The second specified method applies to the stack region and thread stack region. The third specified method applies to Dynamically allocated memory region. If a value not in the Specified Value column is specified, the respective value in "prepage:demand:prepage" is assumed specified.

## ● Settings for tuning (Environment variables specific to large pages)

Environment Variable Name	Specified Value (Default is <u>Underlined</u> )	Description
XOS_MMM_L_ARENA_FREE	<u>1</u>   2	This setting relates to how to handle the heap area released by free(3). Specify "1" to immediately release the memory that can be released. Specify "2" to never release memory but instead pool all memory for reuse.
XOS_MMM_L_ARENA_LOCK_TYPE	0   <u>1</u>	This setting relates to the memory allocation policy. "0" means memory acquisition performance has priority. "1" means memory utilization efficiency has priority.
XOS_MMM_L_MAX_ARENA_NUM	Integer value between <u>1</u> and INT_MAX [decimal number]	Set the number of arenas that can be generated (total for the process heap area and thread heap area). This setting is enabled when XOS_MMM_L_ARENA_LOCK_TYPE=0.
XOS_MMM_L_HEAP_SIZE_MB	Integer value between <u>MALLOC_MMAP_THRESHOLD_x_2</u> and ULONG_MAX <MiB> [decimal number]	Set the size of memory acquired when generating and extending the thread heap area in order to use the thread heap area.
XOS_MMM_L_COLORING	0   <u>1</u>	With this setting, enable or disable cache coloring. Cache coloring mitigates L1 cache conflicts of the processor. "0" does not enable cache coloring. "1" enables cache coloring when the size of memory being acquired by mmap(2) is equal to or greater than MALLOC_MMAP_THRESHOLD_ (default: 128 MiB).
XOS_MMM_L_FORCE_MMAP_THRESHOLD	<u>0</u>   1	Set whether to prioritize mmap(2) when the size of memory being acquired is equal to or greater than MALLOC_MMAP_THRESHOLD_ (default: 128 MiB). "0" does not prioritize mmap(2). First, a search for free memory in the heap area returns any free memory found in the area. Then, mmap(2) acquires memory only when no free memory has been found in the heap area. "1" prioritizes mmap(2). Without a search for free memory in the heap area, mmap(2) acquires memory (in spite of any free memory).

■ For details on environment variables (MALLOC\_MMAP\_THRESHOLD\_, etc.) for glibc , see the user's guide.

- Precautions (Side Effects) Related to Memory Usage

- Static data (.data) area

- Both side effects 1 and 2 always occur.

- Static data (.bss) area

- If the following conditions are met, side effects 1 and 2 occur.

- a. The .dynsym section (dynamic section) has a symbol.
    - b. The symbol has an address within the range of the bss area (bss\_start, bss\_end) of the main program.
    - c. The symbol is global (STB\_GLOBAL) or weak (STB\_WEAK).
    - d. The symbol type is variable (STT\_OBJECT).
    - e. The size of the symbol is larger than 0.

- Side effects

- 1. A large page may use double or triple the memory area.
    - 2. The prepaging method ("prepage") is the active method even when the demand paging method ("demand") is set for the static data (.bss) area in XOS\_MMM\_L\_PAGING\_POLICY.

- Notes on transformational function reference such as SPREAD
  - At run time, the execution cost of an intrinsic transformational function can be high if it is called repeatedly.
  - If the execution cost of an intrinsic function is a problem, avoid it by writing a program that performs the equivalent processing instead of the intrinsic function reference.

# Timers Supported by Fortran

- Timer Specifications
- Timer Precision

# Timer Specifications (1/3)

## Specifications of the major timer routines

No.	Routine Name	Function	Format	What is Measured	Unit Returned by Routine
1	DATE_AND_TIMEb uilt-in subroutine	Obtains the date and time.	CALL DATE_AND_TIME ( [ DATE , TIME , ZONE , VALUES ] ) DATE: Sets the current date in the CCYYMMDD format. (CC: century, YY: year, MM: month, DD: day) TIME: Sets the current time in the hhmmss.sss format. (hh: hour, mm: minute, ss.sss: second) ZONE: Sets the time zone offset from UTC in the shhmm format. (s: sign, hh: hour, mm: minute) VALUES(1): Year VALUES(2): Month VALUES(3): Day VALUES(4): Time zone offset from UTC in minutes VALUES(5): Hour VALUES(6): Minute VALUES(7): Second VALUES(8): Millisecond	Current date/time	
2	GETTIM service subroutine	Obtains the current time.	CALL GETTIM ( hour , minute , second , second1_100 ) hour: Sets the current hour. minute: Sets the current minute. second: Sets the current second. second1_100: Sets the current hundredth of a second.	Current time	
3	FDATE service subroutine	Obtains the current date and time by converting them to ASCII code.	CALL FDATE ( string ) string: Sets the current date and time in the order of day of the week, month, day, time, and year.	Current date and time	
4	ITIME service subroutine	Obtains the current hour, minute, and second.	CALL ITIME ( ia ) ia(1): Sets the current hour. ia(2): Sets the current minute. ia(3): Sets the current second.	Current hour, minute, and second	
5	GETTOD service subroutine	Obtains the current real time. The real time is the time in microseconds elapsed since a specific time point in the past. Usually, it represents the time since system boot.	CALL GETTOD ( g ) g: Sets the elapsed time in microseconds.	Wall clock time	Microsecond



# Timer Specifications (2/3)

## Specifications of the major timer routines

No.	Routine Name	Function	Format	What is Measured	Unit Returned by Routine
6	GMTIME service subroutine	Obtains the specified system clock time information to show the second, minute, hour, day, month, year, day of the week, total number of days since January 1, and observation of daylight saving time according to Greenwich Mean Time.	CALL GMTIME ( time , t ) time: Specifies the system clock time. The following array is set from the system clock time with the specified "time" value according to Greenwich Mean Time: t(1): Second t(2): Minute t(3): Hour t(4): Day t(5): Month t(6): Total number of years since 1990 t(7): Total number of days since Sunday t(8): Total number of days since January 1 t(9): The setting is 0 for standard time and 1 for daylight saving time.	Wall clock time	
7	LTIME service subroutine	Obtains the specified system clock time information to show the second, minute, hour, day, month, year, day of the week, total number of days since January 1, and observation of daylight saving time according to the local time.	CALL LTIME ( time , t ) time: Specifies the system clock time. The following array is set from the system clock time with the specified "time" value according to the local time: t(1): Second t(2): Minute t(3): Hour t(4): Day t(5): Month t(6): Total number of years since 1990 t(7): Total number of days since Sunday t(8): Total number of days since January 1 t(9): The setting is 0 for standard time and 1 for daylight saving time.	Wall clock time	
8	omp_get_wtime routine	Obtains the current real time. The real time is the time in seconds elapsed since a specific time point in the past. Usually, it represents the time since system boot.	y = omp_get_wtime ( ) y: Returns the elapsed time in seconds.	Wall clock time	Second
9	SECNDS service function	Obtains the number of seconds by subtracting the value specified in the first argument from the number of seconds elapsed since midnight in the system clock time.	y = SECNDS ( sec ) y: Returns the number of seconds obtained by subtracting the sec value from the number of seconds elapsed since midnight in the system clock time. sec: Specifies a value in seconds to subtract from the number of seconds elapsed since midnight in the system clock time.	Wall clock time	Second
10	SYSTEM_CLOCK built-in subroutine	Obtains the total time elapsed since midnight. The total time is an elapsed time in seconds. Usually, it represents the time since system boot.	CALL SYSTEM_CLOCK ( [ COUNT , COUNT_RATE , COUNT_MAX ] ) COUNT: Sets the time elapsed from midnight until the present time. COUNT_RATE: Sets the rate of counting (=1000) per second by the processing system. COUNT_MAX: Sets the maximum COUNT value (=86399999).	Wall clock time	Millisecond
11	RTC service function	Obtains the total number of seconds elapsed since 0:00 on January 1, 1970 in UTC.	y = RTC ( ) y: Sets the total number of seconds elapsed since 0:00 on January 1, 1970.	Wall clock time	Second

# Timer Specifications (3/3)

## ● Specifications of the major timer routines

No.	Routine Name	Function	Format	What is Measured	Unit Returned by Routine
12	TIME service function	Obtains the time elapsed in seconds since 00:00:00 GMT (January 1, 1970).	iy = TIME ( ) iy: Returns the elapsed time in seconds since 00:00:00 GMT (January 1, 1970).	Wall clock time	Second
13	TIMEF service function	Returns the elapsed time since the last called TIMEF service function.	y = TIMEF ( ) y: Returns the elapsed time since the last executed TIMEF service function.	Wall clock time	Second
14	TIMER service subroutine	Obtains the time elapsed in hundredths of seconds since midnight.	CALL TIMER( ix ) ix: Sets the time elapsed in hundredths of seconds since midnight.	Wall clock time	Second
15	CLOCK service subroutine	Obtains the CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads.	CALL CLOCK ( g , i1 , i2 ) g: Sets the CPU time in the unit specified in i1. i1: Specifies the unit (second, millisecond, microsecond) for the return value. i2: Specifies the type of the variable specified in g.	CPU time used by current processes and their internal threads	Any of following units depending on specification: - Second - Millisecond - Microsecond
16	CLOCKM service subroutine	Obtains the CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads.	CALL CLOCKM ( i ) i: Sets the CPU time in milliseconds.	CPU time used by current processes and their internal threads	Millisecond
17	CLOCKV service subroutine	Obtains the CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads. The routine is compatible with vector machines.	CALL CLOCKV ( g1 , g2 , i1 , i2 ) g1: Always sets 0. * VU time is set in vector machines. g2: Sets the CPU time in the unit specified in i1. i1: Specifies the unit (second, millisecond, microsecond) for the return value. i2: Specifies the type of the variable specified in g2.	CPU time used by current processes and their internal threads	Any of following units depending on specification: - Second - Millisecond - Microsecond
18	CPU_TIME built-in subroutine	Obtains the CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads.	CALL CPU_TIME ( TIME ) TIME: Sets the CPU time in seconds.	CPU time used by current processes and their internal threads	Second
19	DTIME service function	Obtains the CPU time from the last called DTIME service function. The CPU time represents the time used by the processes executing the program and all their internal threads.	y = DTIME ( tm ) y: Returns the CPU time since the last called DTIME service function. tm(1): Sets the user CPU time in seconds. tm(2): Sets the system CPU time in seconds.	CPU time used by current processes and their internal threads	Second
20	ETIME service function	Obtains the CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads.	y = ETIME ( tm ) y: Returns the CPU time elapsed since the execution start of an executable program. tm(1): Sets the user CPU time in seconds. tm(2): Sets the system CPU time in seconds.	CPU time used by current processes and their internal threads	Second
21	SECOND service function	Obtains the user CPU time elapsed since the execution start of an executable program. The CPU time represents the time used by the processes executing the program and all their internal threads.	y = SECOND ( ) y: Returns the user CPU time in seconds elapsed since the execution start of an executable program.	CPU time used by current processes and their internal threads	Second

# Timer Precision (1/2)

## ● Precision of the major timer routines (Timer overhead)

No.	Routine Name	Precision Resolution	Overhead (μs)	Thread Safe	Implementation	GCC Overhead (μs)	Fujitsu/GCC	Remarks
1	DATE_AND_TIME built-in subroutine	1/1,000,000	77.37 (Improved version) 47.27	✓	gettimeofday localtime	163.13	0.47	
2	GETTIM service subroutine	1/1,000,000	38.37	✓	gettimeofday localtime			
3	FDATE service subroutine	1/1,000,000	18.58	✓	time ctime_r	78.71	0.24	
4	ITIME service subroutine	1/1,000,000	36.07	✓	time localtime	69.27	0.52	
5	GETTOD service subroutine	1/100,000,000	0.02	✓	arm asm instruction time stamp counter gmtime_r localtime_r			Resolution: Value of (1/cntfrq_el0)
6	GMTIME service subroutine		5.72	✓	gmtime_r	58.21	0.10	
7	LTIME service subroutine		10.81	✓	localtime_r	67.59	0.16	
8	omp_get_wtime routine	FJOMP: 1/100,000,000 libomp: 1/1,000,000	1.00	✓	FJOMP: arm asm instruction time stamp counter libomp: gettimeofday localtime	6.05	0.17	Resolution: Value of (1/cntfrq_el0)
9	SECNDS service function	1/1,000,000	49.05	✓	gettimeofday localtime	77.31	0.63	
10	SYSTEM_CLOCK built-in subroutine	1/100,000,000	20.98 (Improved version) 1.42	✓	arm asm instruction time stamp counter	5.23	4.01 (Improved version) 0.27	Resolution: Value of (1/cntfrq_el0)
11	RTC service function	1/1,000,000	5.21	✓	time			

# Timer Precision (2/2)

## ● Precision of the major timer routines (Timer overhead)

No.	Routine Name	Precision Resolution	Overhead (μs)	Thread Safe	Implementation	GCC Overhead (μs)	Fujitsu/GCC	Remarks
12	TIME service function	1/1,000,000	5.28	✓	time	5.03	1.05	
13	TIMEF service function	1/1,000,000	7.26	✓	time			
14	TIMER service subroutine	1/1,000,000	49.85	✓	gettimeofday localtime			
15	CLOCK service subroutine	1/1,000,000	12.82	✓	getrusage			
16	CLOCKM service subroutine	1/1,000,000	13.70	✓	getrusage			
17	CLOCKV service subroutine	1/1,000,000	14.12	✓	getrusage			
18	CPU_TIME built-in subroutine	1/1,000,000	18.09	✓	getrusage	5.39	3.36	
19	DTIME service function	1/1,000,000	14.12	✓	getrusage	10.20	1.38	
20	ETIME service function	1/1,000,000	13.33	✓	getrusage	9.37	1.42	
21	SECOND service function	1/1,000,000	14.27	✓	getrusage	5.91	2.41	

# Relation of Fortran Data Attributes and Optimization

- Attribute List
- allocatable Attribute
- pointer Attribute
- contiguous Attribute
- intent(in) Attribute
- intent(out) Attribute
- intent(inout) Attribute
- value Attribute
- pure Attribute
- save Attribute
- Other Grammatical Notes

# Attribute List (1/3)

No.	Attribute Name	Attribute Outline	Standard	Remarks	Detailed Explanation
1	dimension ( array-spec )			<ul style="list-style-type: none"> <li>Explicit array shape promotes optimization.</li> </ul>	No
	Explicit-shape array	<ul style="list-style-type: none"> <li>Specifies the bounds by an integer expression. Example: dimension( 2:3 ) :: array</li> </ul>	77		
	Assumed-shape array	<ul style="list-style-type: none"> <li>Specifies the shape in the dummy array assumed from the caller.</li> <li>Explicit interface is required (such as interface block) Example: dimension( 2: ) :: array</li> </ul>	90		
	Deferred-shape array	<ul style="list-style-type: none"> <li>Specifies allocatable array and pointer array</li> <li>Explicit interface is required for the function results and dummy arguments. Example: dimension( : ) :: array</li> </ul>	90		
	Assumed-size array	<ul style="list-style-type: none"> <li>Specifies the dummy array to assume the size implicitly. Example: dimension( 2: * ) :: array</li> </ul>	77		
	Implicit-shape array	<ul style="list-style-type: none"> <li>Specifies a named constant array of the same shape as the constant expression. Example: Named constant array of shape[3] integer, parameter, dimension( 2: * ) :: array = [1,2,3]</li> </ul>	08		
2	allocatable	<ul style="list-style-type: none"> <li>Allocates an object by the allocate statement or the intrinsic assignment statement (2003).</li> <li>Can allocate a scalar (2003).</li> </ul>	90 03	<ul style="list-style-type: none"> <li>Objects are not unknown like pointers.</li> <li>But extra instructions are issued, compared to explicit shape.</li> </ul>	Yes
3	pointer	<ul style="list-style-type: none"> <li>Specifies it is a pointer.</li> </ul>	90	<ul style="list-style-type: none"> <li>Data overlap relation is unknown.</li> </ul>	Yes
4	target	<ul style="list-style-type: none"> <li>Specifies an object for which pointer association is available.</li> </ul>	90		No

# Attribute List (2/3)

No.	Attribute Name	Attribute Outline	Standard	Remarks	Detailed Explanation
5	Contiguous: Specifies that the pointer array or the assumed-shape array are contiguous.			<ul style="list-style-type: none"><li>Data contiguity is guaranteed.</li></ul>	Yes
	Pointer array + contiguous	<ul style="list-style-type: none"><li>The pointer association destination is contiguous. The program needs to assure the association destination is contiguous.</li><li>If the dummy argument of a pointer array has the contiguous attribute, the actual argument is a pointer with the contiguous attribute. A fault leads to compilation error.</li></ul>	08		Yes
	Assumed-shape array + contiguous	<ul style="list-style-type: none"><li>The compiler guarantees that the array is contiguous even when the actual argument is not contiguous (The 2008 standard specification changed, implemented on post-K computer compiler).</li></ul>			Yes
6	intent ( in / inout / out): Specifies intents of the dummy argument.			<ul style="list-style-type: none"><li>The intent attribute is used for optimization when the –Kintento option is specified.</li><li>The –Kintento option becomes valid with -O1 or above.</li></ul>	Yes
	in	<ul style="list-style-type: none"><li>Pointer: Does not change the association status of the pointer. The associated value can be changed.</li><li>Allocation: Does not change allocation status/the dummy argument value.</li><li>Other than allocation/pointer: Does not change the dummy argument value.</li></ul>	90		
	inout	<ul style="list-style-type: none"><li>Pointer: Changes the association status and the value of the pointer.</li><li>Other than pointer: References and changes the value of the actual argument to be associated.</li></ul>	90		
	out	<ul style="list-style-type: none"><li>Allocation: Deallocates the field before the procedure is executed.</li><li>Pointer: Makes the association status of the point unstable when the procedure is called.</li><li>Other than the above: Value definition is required prior to reference to the dummy argument during the execution of the procedure.</li></ul>	90		
7	value	<ul style="list-style-type: none"><li>Specified in the dummy argument. This specifies the argument is assumed by the value. <u>Difference from intent(in)</u> The dummy argument of the subprogram can be changed. It does not have an affect on the actual argument.</li><li>The actual argument is assigned to a temporary area and the temporary area is associated with the dummy argument.</li><li>The array can be also specified (the temporary area of the array is generated).</li></ul>	03	<ul style="list-style-type: none"><li>With effect of the optimization promotion</li></ul>	Yes
			08		

# Attribute List (3/3)

No.	Attribute Name	Attribute Outline	Standard	Remarks	Detailed Explanation
8	pure	<ul style="list-style-type: none"> <li>Specified for a procedure without side effects.</li> <li>Specified for the function statement or the subroutine statement.</li> <li>Reference to a procedure in “do concurrent” requires to have the pure attribute.</li> </ul>	03 08	<ul style="list-style-type: none"> <li>With effect of optimization promotion</li> </ul>	Yes
9	save	<ul style="list-style-type: none"> <li>Retains the allocation status, the definition status, and the value even after the return statement or the end statement is executed.</li> <li>The save variable is not thread-safe.</li> </ul>	77		Yes
10	intrinsic	<ul style="list-style-type: none"> <li>Specifies it is an intrinsic procedure.</li> </ul>	77		No
11	optional	<ul style="list-style-type: none"> <li>Specifies a dummy argument is not necessarily associated with an actual argument in reference to a procedure.</li> </ul>	90		No
12	parameter	<ul style="list-style-type: none"> <li>Specifies that it is a named constant.</li> </ul>	77		No
13	private / public	<ul style="list-style-type: none"> <li>Specifies whether entities in a module can be referenced by use association in the module specification part.</li> </ul>	90		No
14	protected	<ul style="list-style-type: none"> <li>Restricts the locations to use entities other than the specified module.</li> </ul>	03		No
15	volatile	<ul style="list-style-type: none"> <li>Specifies that the object is not the target of optimization.</li> </ul>	03	<ul style="list-style-type: none"> <li>Optimization is not performed.</li> </ul>	No
16	asynchronous	<ul style="list-style-type: none"> <li>Specifies to be used at asynchronous input/output.</li> <li>Optimization by the compiler is not performed because it may be being referenced at the asynchronous input/output statement.</li> </ul>	03	<ul style="list-style-type: none"> <li>Optimization is not performed.</li> </ul>	No
17	external	<ul style="list-style-type: none"> <li>Specifies that it is an external procedure, a dummy procedure, or a procedure pointer.</li> </ul>	77		No
18	bind	<ul style="list-style-type: none"> <li>Specifies that it is interoperated with C language.</li> </ul>	03		No
19	codimension [ coarray-spec ]	<ul style="list-style-type: none"> <li>Specifies a coarray, a co-dimension, and co-bounds.</li> </ul>	08		No



```
subroutine foo  
real,dimension(:,:), allocatable :: a  
allocate( a(2,3) )  
end ! Variable a is deallocated.
```

```
integer,allocatable::a(:)  
integer:: b(3)=1  
allocate( a(2) )  
print *,shape( a ) ! The shape of a is [2].  
a = b ! Variable a is deallocated and allocated  
! with the bounds of b, and  
! if the left side is a(:) with an index,  
! the shape remains [2].  
print *,shape( a ) ! The shape of a is [3].
```

```
type x ; integer,allocatable:: a(:) ; end type  
type ( x ) :: y , z  
allocate( y%a(2) , z%a(3) ) ; .....  
y = z ! Variable y%a is deallocated and allocated  
! newly with the bounds (1:3) of z%a.
```

## Grammatical notes

- The initial state is “unallocated.”
- The object of a subprogram without the save attribute is deallocated at the end of the procedure.

## Reallocation by the assignment statement (2003)

- Allocation available by the assignment statement, too.
- When the left side of the assignment statement is a variable with the allocatable attribute and the shape is different from the right side, the variable of the left side is reallocated in the shape of the right side.  
Note: The specification is extended when Fortran 95 is upgraded to Fortran 2003.  
Vendors make this work only when the compile option is specified to maintain execution performance up to Fortran 95.  
The compile option on Fujitsu is -Nalloc\_assign.

## Reallocation by the assignment statement of a derived-type variable

- When the allocatable attribute is specified for a structure component, the components of the left side are deallocated and reallocated with the bounds of the components of the right side.  
Note: Specification of Fortran 2003 and later  
This operates regardless of option specification.

```
real,dimension(:,:),pointer :: a ! data pointer  
procedure(),pointer:: prc ! procedure pointer
```

## Grammatical notes

- There are two kinds of pointers: a data pointer and a procedure pointer.

## Data pointer

- The elements of the pointer array may overlap with those of an array or pointer array with the target attribute.
- The pointer array is not necessarily contiguous. (There may be a gap between elements.)
- Pointer array with the contiguous attribute  
To be explained on the next page.

```
real,dimension(:,:), pointer :: a,b  
real,dimension(2,3), target :: t  
a=> t  
b=> a( :: 2, :)
```

## Procedure pointer

- A procedure pointer is specified in the procedure statement.

```
interface  
  function foo(d)  
    intent(in):: d  
  end function  
end interface  
procedure(foo), pointer :: p  
p=> foo  
print *, p( 1.0 )
```

```
subroutine foo(a)
real,dimension(:,:), contiguous :: a
real,dimension(:,:), pointer, contiguous :: p
```

```
integer,pointer:: pa(:)  ! without
                        ! contiguous attribute
call foo( pa )           ! compilation error
contains
  subroutine foo( p )
    integer,pointer,contiguous:: p(:)
    integer,target :: t(5)=[1,2,3,4,5]
    p=> t(:, 2 )          ! compilation error
    k=2; p=> t(:, k )     ! object not in a
                        ! contiguous area
    print *,p(2)         ! The value is unstable.
  end subroutine
```

## Grammatical notes

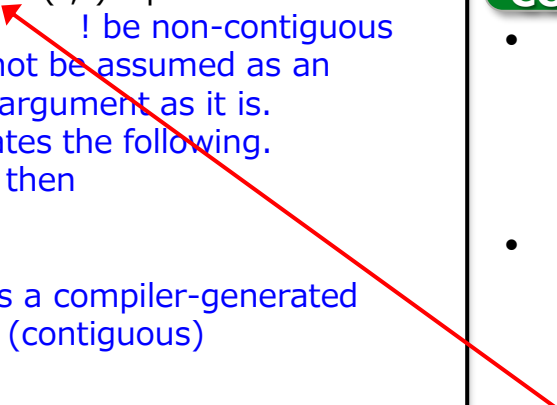
- Can be specified for a pointer array or an assumed-shape array.
- Declares that the associating object is contiguous.
- For “pointer array with the contiguous attribute,” see pointer Attribute.

## Pointer array with the contiguous attribute

- A pointer array with the contiguous attribute is contiguous.
- Therefore, the pointer needs to be associated with the contiguous data.

```
integer:: a(3)
call foo( a( :: 2 ) ) ! non-contiguous
! The compiler generates the following.
! tmp = a( :: 2 ) // tmp is a
! compiler-generated array (contiguous)
! call foo(tmp)
! a( ::2 ) = tmp
contains
subroutine foo( d )
integer, dimension(:,:),contiguous:: d
```

```
subroutine sub(pa , pac)
integer,pointer, dimension(:,:): pa ! Possible to
! be non-contiguous
call foo( pa ) ! pa cannot be assumed as an
! actual argument as it is.
! The compiler generates the following.
! if (pa is contiguous) then
! call foo( pa )
! else
! tmp = pa // tmp is a compiler-generated
! array (contiguous)
! call foo( tmp )
! pa = tmp
! endif
contains
subroutine foo( d )
integer, dimension(:,:),contiguous::d ! Assumed-
: shape array
```



## Assumed-shape array with contiguous attribute (1)

- Declares an assumed-shape array to be contiguous.
- Even when the actual argument is contiguous, the compiler guarantees the array is contiguous. (See the comment on the left.)

## Assumed-shape array with contiguous attribute (2)

- If the actual argument may be not contiguous, whether it is contiguous is judged dynamically and the compiler guarantees that it is contiguous.
- If the actual argument is a pointer array or an assumed-shape array, and if the array is contiguous, the actual argument should be specified with the contiguous attribute, too. If it is with the contiguous attribute, the compiler does not generate the judgement on whether it is contiguous but assumes it directly.

```
subroutine foo( a )  
real, intent(in):: a  
a=a+1.0      ! The value definition  
end          ! results in a compilation error.
```

```
subroutine foo( p )  
real, intent(in), pointer :: p(:, :)  
p=1.0        ! The value can be defined.  
allocate( p(2,3) ) ! Change in association  
end          ! results in a compilation error.
```

```
subroutine foo( a )  
real, intent(in), allocatable :: a(:, :)  
a=1.0        ! The value definition results  
             ! in a compilation error.  
allocate( a(2,3) ) ! Allocation results in a  
end           ! compilation error.
```

## intent(in)

- A value cannot be defined.

## intent(in) of a pointer

- The value in the association destination can be defined. Where the pointer points to cannot be changed but what is pointed can be changed.
- Pointer association cannot be changed.

## Allocatable intent(in)

- Allocation or deallocation cannot be performed. The difference from pointer is whether a value can be defined.
- A value cannot be defined.

```
real:: a
a=1.0
call sub( a ) ! Associated with intent(in) dummy
               ! argument.
print *, a    ! The value is unstable.
contains
  subroutine sub( d )
    real, intent(in):: d
    call def( d ) ! Intents of the argument is implicit.
  end subroutine
end
subroutine def( dd )
  real:: dd
  dd=2.0 ! The value of the dummy argument
end      ! is defined, the actual argument of
          ! intent(in) is updated.
```

## Variable value definition by intent(in) is incorrect

- When the intent(in) variable value is defined, the behavior is not guaranteed.
- When the intent(in) variable is specified with an actual argument and the value is defined at the destination of the procedure call, this problem is hard to be found.

## Check available with the debug option -Ha

A definition error of the intent(in) variable value is detected during execution.

```
subroutine foo( p, a )  
real, intent(out), allocatable :: a(:, :)  
real, intent(out), pointer :: p(:, :)  
allocate( p(2,3), a(2,3) ) ! Association/allocation  
p=1.0 ! before reference  
a=1.0  
end
```

## pointer/allocatable intent(out)

- The pointer is unstable when the procedure is called.
- Allocatable is deallocated when the procedure is called.
- Association or allocation is required before reference.

Check available with the debug option -Hx  
An unassociated pointer error is detected during execution.

```
real :: a  
a=1.0  
call foo( a )  
print *, a ! The result is unstable.  
contains  
subroutine foo( d )  
real, intent(out) :: d  
d=d+1.0 ! The value cannot be referenced  
end subroutine ! before it is defined.  
end
```

## intent(out) other than above

- Unstable when the procedure is called.
- The actual argument must be definable.
- The value must be defined before it is referenced.

Check available with the debug option -Hu  
An undefined variable referencing error is detected during execution.

```
subroutine foo( p, a, d )  
real, intent(inout), allocatable :: a(:, :)  
real, intent(inout), pointer :: p(:, :)  
real, intent(inout) :: d(2,3)  
:  
end
```

```
call foo( 1.0 ) ! The actual argument cannot  
                ! be defined. Compilation error  
contains  
subroutine foo1( d )  
real, intent(inout) :: d  
end subroutine  
end
```

## intent(inout)

- The dummy argument can be referenced or defined.
- Association change, allocation, or deallocation is available.

## Actual argument for intent(inout)

- The actual argument must be definable.
- When the intent attribute is omitted and not defined at the time of execution, the corresponding actual argument is not necessarily definable.
  - \* In the left figure, a compilation error does not occur if the intent(inout) does not exist.



## Grammatical notes

- Indispensable at mutual reference when the argument is assumed with the value in association with C language.
- Change in the dummy argument value and definition status does not affect the actual argument.
- The actual argument is assigned to a temporary area and the temporary area is associated with the dummy argument.
- A procedure that has the dummy argument with the value attribute needs **interface block**.
- The argument is assumed via the stack or a register except the following.
  - ✓ Array (not specifiable in association with C language)  
2008 specification
  - ✓ Character type of a procedure without procedure language binding specifier (bind (c))
  - ✓ Derived type whose component contains a pointer or allocation
  - ✓ Derived type other than above (following ABI)
  - ✓ optional attribute

```
subroutine sub(k) bind(c) ! Called from a C
integer,value :: k      ! program.
k=k+1 ! The value can be assigned, no
print *,'k=',k ! impact on the actual argument.
end
interface
  subroutine foo(k) bind(c) ! Explicitly indicates the
    integer,value :: k      ! procedure of the program.
  end subroutine
end interface
integer::n
n=1
call foo( n ) ! A call of a C program
end
```

## C program

```
void sub(int);
int foo(int k) {
  sub(k) ; /* A call of a Fortran program */
  return 0 ;
}
```

```
module mod
contains
  pure function foo( d )
    real , intent(in):: d
    foo = d + 1.0
  end function
end
subroutine test(a,b)
use mod
real,dimension(1000) :: a,b
do concurrent(i=1:1000)
  a(i) = a(i) + foo( b(i) ) ! The procedure in
enddo                      ! do concurrent is pure.
end subroutine test
```

## Grammatical notes

- A procedure with pure (or elemental) specified in the function statement or the subroutine statement defines there is no side effect.
- Restrictions of a pure procedure
  - ✓ The dummy argument of a function needs to have the intent(in) or value attribute (2008) unless it has the pointer attribute.
  - ✓ The dummy argument of a subroutine needs to have the intent attribute unless it has the pointer attribute.
  - ✓ For local variables, an explicit initial value and the save attribute cannot be specified.
  - ✓ Variables in the common block and variables referenceable by a host association or a use association must not be defined.
  - ✓ All the procedures referenced in pure procedures must be pure procedures.  
etc.

## Relation to optimization

Specifying pure promotes optimization including automatic parallelization.

```
subroutine sub_save()  
  real(8), save :: a  
  a = a + 1.0  
end subroutine sub_save
```

At the time of re-entry, variable a with the save attribute guarantees the value at the time of the return of the last entry.

## Grammatical notes

- Data is saved even after return or end. => to guarantee the value of the variable at re-entry
- The “save” variable is not thread-safe.
- In K computer, “save” was the default because the specification can be traced back to main frames.



## Relation of the save attribute and optimization

If the save attribute is unnecessarily specified, variables that should be dead variables cannot be eliminated and redundant store may remain.

## Making -Kauto the default

```
subroutine sub_flocal()  
  real(8) :: a  
  a = a + 1.0  
end subroutine sub_flocal
```

## Impact of making -Kauto default

- ✓ Variable “a” is stored in a register or the stack.
- ✓ The value of variable “a” declared without the save attribute is undefined at the time of re-entry.
- ✓ The definition of variable “a” becomes unreferenced, and optimization deletes “a = a + 1.0”.

- The case where the compiler generates a temporary array internally
  - Three examples of temporary array generated by the compiler according to the grammar (mentioned later)
    - A non-contiguous actual argument array is associated with an explicit-shape array or an assumed-shape array with the contiguous attribute.
    - The actual array is an array or a dummy array with the value attribute.
    - An array assignment statement has an overlapping array element in the left side and the right side of the expression.
  - Problems in temporary arrays generated by the compiler
    - Overhead of allocate / deallocate to generate a temporary array
      - ✓ The action to generate a temporary array in the stack (-Kauto, -Ktemparraystack) is made to the default, so there is no worry about overhead. (The stack size needs attention.)
    - Overhead of copy-in / copy-back to a temporary array
      - ✓ The compiler performs optimization if possible, but if it cannot, overhead occurs.

# Case where an array that does not exist in the program is generated (1/3)

- A non-contiguous actual argument array is associated with an explicit-shape array or an assumed-shape array with the **contiguous** attribute.

```
subroutine sub(a)
  real(8),dimension(1:1000) :: a
  interface
    subroutine foo(a,m)
      real(8),dimension(1:m) :: a
    end subroutine
  end interface
  call foo(a(1:1000:2),500)
end subroutine
```

Sample compiler output when a non-contiguous data is received in an explicit-shape array

```
tmp(1:500) = a(1:1000:2)
call foo(tmp,500)
a(1:1000:2) = tmp
```

# Case where an array that does not exist in the program is generated (2/3)

- The actual argument is an array or a dummy array with the **VALUE** attribute (Fortran 2008)

```
subroutine sub(x)
  integer(4),dimension(1:80) :: x
  interface
    subroutine foo(a)
      integer(4),dimension(:),value :: a
    end subroutine foo
  end interface
  call foo(x)
end subroutine sub
```

Sample compiler output when the actual argument is an array or a dummy array with the value attribute

```
tmp(1:80) = x(1:80)
call foo(tmp)
```

Grammar: "Change in the value and the definition status of a dummy argument does not affect the actual argument."  
That is, the result of defining a on foo side must not be reflected to x. Therefore, consider "value attribute = reference only" and x cannot be passed as it is.

# Case where an array that does not exist in the program is generated (3/3)

- An array assignment statement may have an overlapping array element between the left side and the right side.

The grammar of the array description indicates assignment to the left side is performed after the right side is evaluated.

! Case to get an incorrect result unless the  
! assignment to the left side is performed  
! after the right side is calculated.

```
subroutine sub(a,b,n,m)
real(8),dimension(1:n) :: a,b
a(2:m+1) = a(1:m) + b(1:m)
end subroutine
```

! Assignment is implemented by using  
! the compiler generated array (temp).  
! Same as when the argument is a pointer, etc.  
tmp(1:m) = a(1:m) + b(1:m)  
a(2:m+1) = tmp(1:m)

There is no problem in loop fusion of the left side loop and the right side loop.

```
subroutine sub(a,b,n,m)
real(8),dimension(1:n) :: a,b
a(1:m) = a(2:m+1) + b(1:m)
end subroutine
```

! The above example can be optimized by  
! inverting the loop (implemented on the  
! compiler). A compiler generated array  
! becomes unnecessary.  
a(m+1:2:-1) = a(m:1:-1) + b(m:1:-1)

# Example of Performance Tuning by pointer and Argument Interface Improvement

- Difference Between pointer and allocatable
- Performance Tuning by pointer
- Performance Tuning by Argument Interface Improvement



# Difference Between pointer and allocatable

```
subroutine test(a,b,n1,n2)
real(kind=8),dimension(:,:), pointer :: a,b
do j=1,n2
  do i=1,n1
    a(i,j) = a(i,j) + b(i,j)
  enddo
enddo
end subroutine test
```



```
subroutine test(a,b,n1,n2)
real(kind=8),dimension(:,:), allocatable :: a,b
do j=1,n2
  do i=1,n1
    a(i,j) = a(i,j) + b(i,j)
  enddo
enddo
end subroutine test
```

Difference in Grammatical Interpretation and Optimization Effect by Correction Difference		pointer	allocatable
Data dependency of definition array and reference array		Unknown	No
Data dependency beyond an iteration		Unknown	No
Access continuity in the first dimension of the inner-most loop		Unknown	Yes
Access continuity of the whole nested loop		Unknown	Unknown
Possibility of optimization effect by correction	Automatic parallelization	No	Yes
	SIMD	No	Yes
	(Effective) SWP	No	Yes

\* allocatable has more address calculations than explicit-shape.

# Performance Tuning by pointer (1/3)

## ■ Effect of contiguous

```
subroutine test(a,b,n1,n2)
real(kind=8),dimension(:,,:), pointer :: a,b
do j=1,n2
  do i=1,n1
    a(i,j) = a(i,j) + b(i,j)
  enddo
enddo
end subroutine test
```



```
subroutine test(a,b,n1,n2)
real(kind=8),dimension(:,,:), pointer,contiguous :: a,b
do j=1,n2
  do i=1,n1
    a(i,j) = a(i,j) + b(i,j)
  enddo
enddo
end subroutine test
```

### Difference in Grammatical Interpretation and Optimization Effect by Correction Difference

		pointer	Left Column + contiguous
Data dependency of definition array and reference array		Unknown	Unknown
Data dependency beyond an iteration		Unknown	Unknown
Access continuity in the first dimension in the inner-most loop		Unknown	Yes
Access continuity of the whole nested loop		Unknown	Unknown
Possibility of optimization effect by correction	Automatic parallelization	No	No
	SIMD	No	No
	(Effective) SWP	No	No

\* Address calculation group in the inner-most loop is improved by specifying contiguous.

- Resolution of data dependency by the norecurrence optimization specifier

```
subroutine test(a,b,n1,n2)
real(kind=8),dimension(:,,:),
pointer,contiguous :: a,b
do j=1,n2
  do i=1,n1
    a(i,j) = a(i,j) + b(i,j)
  enddo
enddo
end subroutine test
```



```
subroutine test(a,b,n1,n2)
real(kind=8),dimension(:,,:),
pointer,contiguous :: a,b
!ocl norecurrence
do j=1,n2
  do i=1,n1
    a(i,j) = a(i,j) + b(i,j)
  enddo
enddo
end subroutine test
```

Difference in Grammatical Interpretation and Optimization Effect by Correction Difference		pointer+contiguos	Left Column +norecurrence
Data dependency of definition array and reference array		Unknown	No
Data dependency beyond an iteration		Unknown	No
Access continuity in the first dimension in the inner-most loop		Yes	Yes
Access continuity of the whole nested loop		Unknown	Unknown
Possibility of optimization effect by correction	Automatic parallelization	No	Yes
	SIMD	No	Yes
	(Effective) SWP	No	Yes

# Performance Tuning by pointer (3/3)

## ■ Resolution of data dependency by do concurrent

```
subroutine test(a,b,n1,n2)
  real(kind=8),dimension(:,:),
  pointer,contiguous :: a,b
  do j=1,n2
    do i=1,n1
      a(i,j) = a(i,j) + b(i,j)
    enddo
  enddo
end subroutine test
```



```
subroutine test(a,b,n1,n2)
  real(kind=8),dimension(:,:),
  pointer,contiguous :: a,b
  do concurrent(j=1:n2)
    do concurrent(i=1:n1)
      a(i,j) = a(i,j) + b(i,j)
    enddo
  enddo
end subroutine test
```

Difference in Grammatical Interpretation and Optimization Effect by Correction Difference		pointer+contiguos	Left Column +do concurrent
Data dependency of definition array and reference array		Unknown	No
Data dependency beyond an iteration		Unknown	No
Access continuity in the first dimension in the inner-most loop		Yes	Yes
Access continuity of the whole nested loop		Unknown	Unknown
Possibility of optimization effect by correction	Automatic parallelization	No	Yes
	SIMD	No	Yes
	(Effective) SWP	No	Yes

# Performance Tuning by Argument Interface Improvement

## ■ Optimization promotion by using the intent attribute

```
subroutine sub()  
  real(kind=8) :: a,b  
  a = 1.0  
  b = 1.0  
  call foo(a,b)  
  print *,a,b  
end subroutine sub
```

Fujitsu compiler uses the intent attribute for optimization when `-Kintentopt` is valid. (default)

Note: Incorrect intent attribute use results in an error.

```
subroutine sub()  
  interface  
    subroutine foo(a,b)  
      real(kind=8),intent(inout) :: a  
      real(kind=8),intent(in) :: b  
    end subroutine foo  
  end interface  
  real(kind=8) :: a,b  
  a = 1.0  
  b = 1.0  
  call foo(a,b) ! b is not updated.  
  print *,a,b ! b can be replaced by 1.0.  
end subroutine sub
```

Difference in Grammatical Interpretation and Optimization Effect by Correction Difference	Without intent	With intent
Update of a in foo	Yes	Yes
Update of b in foo	Yes	No
Optimization by constant propagation of a	-	-
Optimization by constant propagation of b	No	Yes

## ● Revision History

Version	Date	Details
1.0	Aug. 2020	- First published
1.3	Mar. 2021	- Correcting typographical errors and expressions by reviewing articles
1.5	Jul. 2022	<ul style="list-style-type: none"><li>- Changed slide design</li><li>- Changed default value for <code>-Karray_declaration_opt</code></li><li>- Corrected feature descriptions of <code>-O</code></li><li>- Added methods for dealing with stack shortages</li><li>- Added note on <code>-Knotemparraystack</code></li></ul>
1.6	Mar. 2023	- Correcting typographical errors and expressions by reviewing articles

