プログラミングガイド
プログラミング共通編

2023年3月

V1.6

富士通株式会社

はじめに

- 当資料は、A64FX プロセッサを対象としたアプリケーション開発者やチューニング実施者のためのものです。
- 当資料と併せて以下も参照してください。
  - Fortran 使用手引書
  - C言語 使用手引書
  - C++言語 使用手引書
  - プロファイラ 使用手引書
  - プログラミングガイド　プロセッサ編
  - プログラミングガイド　Fortran編
  - プログラミングガイド　チューニング編
  - Technical Computing Suite ジョブ運用ソフトウェアが提供するマニュアル
- 当資料の記載においては、以下の文章を参考にしています。
  - A64FX 論理仕様書
  - A64FX®Microarchitecture Manual
  - ARM® Architecture Reference Manual (ARMv8 , ARMv8.1 , ARMv8.2 , ARMv8.3)
  - ARM® Architecture Reference Manual Supplement The Scalable Vector Extension
- 商標について
  - Linux® は、Linus Torvalds 氏の米国およびそのほかの国における登録商標または商標です。
  - Red Hat は、Red Hat,Inc. の米国およびそのほかの国における登録商標または商標です。
  - ARMは、ARM Ltd.の英国およびその他の国における登録商標です。
  - そのほかの会社名、製品名等の固有名詞は、各社の登録商標または商標です。
  - 本資料に記載されているシステム名、製品名等には、必ずしも商標表示(®、™)を付記していません

プログラミング共通編　目次(1)

FUJITSU

プログラミング共通編　目次(2)

FUJITSU

プログラミング共通編　目次(3)

FUJITSU

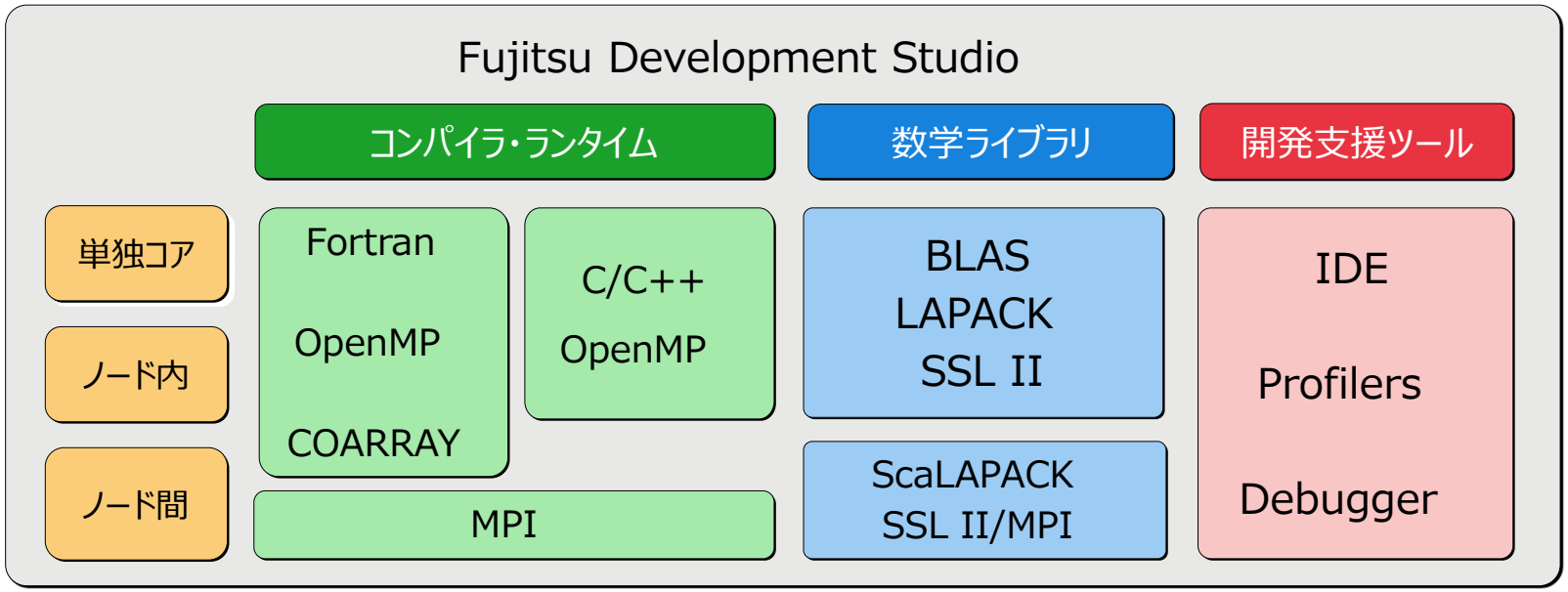
FUJITSU

プログラム開発環境概略

- プログラミング開発環境体系
- 言語仕様
- MPIの高速化
- 数学ライブラリの高速化
- 開発支援ツール

プログラミング開発環境体系

- Fujitsu Development Studio の構成
  - Fujitsu Development Studio は、さまざまな利用形態に対応できるように、PRIMERGY で動作するクロスコンパイラと、A64FX で動作するネイティブコンパイラの2種類のコンパイラを提供します。
  - C/C++コンパイラは、Clang/LLVM と互換性がある clangモードで動作させることができます。clangモードを使うことで、容易にオープンソースのアプリケーションを動作させることができます。



言語仕様

- 最新の標準規格および業界標準仕様への対応
  - A64FX 上で最先端の規格を用いたアプリケーション開発を可能にするため、またオープンソースのアプケリーションを容易に動作させるために、Fujitsu Development Studio は最新の標準規格および業界標準仕様をサポートしています。

サポート標準規格一覧	
言語	サポート規格仕様
Fortran	ISO/IEC 1539-1:2018（Fortran 2018 規格）の一部 ISO/IEC 1539-1:2010（Fortran 2008 規格） ISO/IEC 1539-1:2004、JIS X 3001-1:2009（Fortran 2003 規格） ISO/IEC 1539-1:1997、JIS X 3001-1:1998（Fortran 95 規格） Fortran 90 規格および Fortran 77 規格
C	ISO/IEC 9899:2011（C11 規格） ISO/IEC 9899:1999（C99 規格） ISO/IEC 9899:1990（C89 規格） ※GNUコンパイラの拡張仕様もサポート
C++	ISO/IEC 14882:2017（C++17 規格）の一部 ISO/IEC 14882:2014（C++14 規格） ISO/IEC 14882:2011（C++11 規格） ISO/IEC 14882:2003（C++03 規格） ※GNUコンパイラの拡張仕様もサポート
OpenMP	OpenMP API Version 5.0の一部 OpenMP API Version 4.5
MPI	Message-Passing Interface Standard Version 3.1および4.0の一部

MPI の高速化

FUJITSU

- TofuインターコネクトD を利用した MPI の高速化
  - 従来の4方向同時通信から、6方向同時通信へと、通信機能が大幅に強化された事で、ノード当たりの通信バンド幅が必要な集団通信(Allgather や Bcast)のアルゴリズムが改良されました。

インターコネクト	ネットワークインターフェース数	対象マシン
Tofu	4 (1ノード)	FX10
Tofu2	4 (1ノード)	FX100
TofuインターコネクトD	6 (1ノード)	A64FX

- ネットワーク階層に対応した MPI の高速化
  - CMG(Core Memory Group)を4つ搭載し、それらをリングバスによって接続するNUMA(Non-Uniform Memory Access)アーキテクチャを採用しています。
  - CMG を単位としたプロセス並列化では、TofuインターコネクトD と A64FX 内のリングバスによる階層ネットワーク構造で結合されます。
  - Alltoall のアルゴリズムには、リングバスにおける通信路の競合が最も少なくなるように最適化したアルゴリズムが実装され、高速化を実現しています。

数学ライブラリの高速化 (1/2)

FUJITSU

- コア内の高速化機能による数学ライブラリの高速化
  - Fujitsu Development Studio は、コア内の高速化機能を適用した高速数学ライブラリとして、線型代数分野で著名な BLAS 、LAPACK 、広い分野のアルゴリズムをサポートし、日本国内のR&Dユーザーに幅広く利用されている富士通の数学ライブラリ SSL II 、さらに4倍精度の値を double-double 形式で扱うことで高速化する高速4倍精度基本演算ライブラリを提供します。
  - これらの数学ライブラリをアプリケーションから呼び出すことで、容易にアプリケーションの高速化を実現します。

逐次版ライブラリ（スレッドセーフ）	
BLAS 、LAPACK	米国で開発され Netlib で公開されているデファクトスタンダードな線形計算ライブラリ。 BLAS 約80種、LAPACK 約400種ルーチン。 BLAS の一部で FP16 を独自にサポート。
SSL II	幅広い分野をカバーする、約300種の Fortran ルーチンのライブラリ。
C-SSL II	SSL II への Cインターフェース。
高速4倍精度基本演算ライブラリ	4倍精度の値を double-double 形式で表現し、演算を行うライブラリ。 一部スレッド並列ルーチンを含む。

数学ライブラリの高速化 (2/2)

FUJITSU

- スレッド並列機能による数学ライブラリの高速化
  - Fujitsu Development Studio は、スレッド並列化した高速数学ライブラリとして BLAS 、LAPACK と、SSL II を提供します。
  - これらの数学ライブラリをアプリケーションから呼び出すことで、スレッド並列による高速化を実現します。

逐次版ライブラリ（スレッドセーフ）	
BLAS 、LAPACK	逐次版と同一インターフェース。LAPACK をタスク並列化した PLASMA を含む。 Python（NumPy 、SciPy）からも利用可能。
SSL II スレッド並列機能	重要機能約80ルーチンのスレッド並列版。 混在使用できるように逐次版 SSL II と別インターフェース。
C-SSL II スレッド並列機能	SSL II スレッド並列機能への Cインターフェース。

- MPI 並列機能による数学ライブラリの高速化
  - Fujitsu Development Studio は、MPI 並列化した高速数学ライブラリとして ScaLAPACK と SSL II/MPI を提供します。
  - これらの数学ライブラリをアプリケーションから呼び出すことで、MPI 並列による高速化を実現します。

MPI 並列版ライブラリ	
ScaLAPACK	BLAS 、LAPACK の機能を MPI で並列化したライブラリ。約200種ルーチン。
SSL II/MPI	3次元FFT機能の MPI 並列版。

開発支援ツール

FUJITSU

- アプリケーション開発支援ツール
  - Fujitsu Development Studio は、アプリケーション開発支援ツールとして、統合開発環境、プロファイラ、並列実行デバッガ を提供します。
  - これらのツールを利用することで、効率的にアプリケーションを開発できます。

アプリケーション開発支援ツール	
統合開発環境	開発作業(ソースコード編集、コンパイル、ジョブ投入、ジョブの状態確認、性能情報採取/表示など)の支援 ・ Eclipse ・ Parallel Tools Platform (PTP)
プロファイラ	アプリケーションの性能解析を支援 ・ 基本プロファイラ ・ 詳細プロファイラ ・ CPU 性能解析レポート
並列実行デバッガ	大規模並列処理で発生するトラブル(異常終了やデッドロックなど)の調査を支援 ・ 異常終了調査機能 ・ デッドロック調査機能 ・ コマンドファイルによるデバッグ機能

FUJITSU

コンパイラ推奨オプション

- 翻訳コマンド
- Fortran、C/C++ tradモード
  - 性能重視
  - 精度重視
- C/C++ clangモード

翻訳コマンド

FUJITSU

- 翻訳コマンド(非MPI)

種別	言語	翻訳コマンド	説明
クロスコンパイラ	Fortran	frtpx	ログインノード上で使用します。
	C	fccpx	
	C++	FCCpx	
ネイティブコンパイラ	Fortran	frt	計算ノード上で使用します。
	C	fcc	
	C++	FCC	

- 翻訳コマンド(MPI)

種別	言語	翻訳コマンド	説明
クロスコンパイラ	Fortran	mpifrtpx	ログインノード上で使用します。
	C	mpifccpx	
	C++	mpiFCCpx	
ネイティブコンパイラ	Fortran	mpifrt	計算ノード上で使用します。
	C	mpifcc	
	C++	mpiFCC	

Fortran、C/C++ tradモード（性能重視）

FUJITSU

推奨オプション（性能重視：Fortran、C/C++ tradモード）

```
-Kfast,openmp[,parallel]
```

コンセプト

スレッド並列によるコア活用、SIMD化によるSVE活用、Software Pipeliningによる命令レベルの並列性向上、最適化による演算順序変更、逆数近似演算の利用など、A64FXの性能をフルに引き出すためのオプション指定です。

-Kfastが誘導する翻訳オプション

- Fortran
  -O3 -Keval,fp_contract,fp_relaxed,fz,ilfunc,mfunc,omitfp,simd_packed_promotion

- C/C++
  -O3 -Keval,fast_matmul,fp_contract,fp_relaxed,fz,ilfunc,mfunc,omitfp,simd_packed_promotion

-Kfastが誘導する翻訳オプション

FUJITSU

誘導オプション	言語	意味
-O3	共通	最適化レベル3 SIMDおよびSoftware Pipeliningの最適化に加えて、多重ループのアンローリングなどの最適化が動作。
-Keval	共通	プログラムに対する演算評価方法を変更する最適化が動作
-Kfast_matmul	C/C++	行列積のループを高速なライブラリ呼び出しに変換
-Kfp_contract	共通	Floating-Point Multiply-Add/Subtract演算命令を使用した最適化が動作
-Kfp_relaxed	共通	単精度浮動小数点除算、倍精度浮動小数点除算、およびSQRT関数について、逆数近似演算を行う最適化が動作
-Kfz	共通	flush-to-zeroモードの使用
-Kilfunc	共通	数学関数をインライン展開 関数内の数学関数をインライン展開する。
-Kmfunc	共通	関数をマルチ演算関数に変換する最適化 引数の多重度をSIMD長と同じ値としたマルチ演算関数を使用する。 -Kilfuncできなかった関数が対象
-Komitfp	共通	手続または関数呼び出しにおける最適化 フレームポインタレジスタは保証できなくなる。
-Ksimd_packed_promotion	共通	単精度実数型ならびに4バイト整数型の配列要素のインデックス計算が4バイトの範囲を超えないと仮定して、16SIMD化を促進

Fortran、C/C++ tradモード（精度重視）

FUJITSU

推奨オプション（精度重視：Fortran、C/C++ tradモード）

```
-Kfast,openmp[,parallel],fp_precision
```

コンセプト

-O0と同じ精度にしたいが、最適化はある程度動かしたいという場合に利用します。性能重視の推奨オプションに、精度に影響のあるすべての最適化を抑止する新規オプション-Kfp_precisionを付加するオプション指定です。
性能に大きく影響する複数の最適化を抑止することになります。

-Kfp_precisionが誘導する翻訳オプション

- Fortran
  -Knoeval,nofp_contract,nofp_relaxed,nofz,noilfunc,nomfunc,parallel_fp_precision

- C/C++
  -Knoeval,nofast_matmul,nofp_contract,nofp_relaxed,nofz,noilfunc,
  nomfunc,parallel_fp_precision

-Kfp_precisionが誘導する翻訳オプション

FUJITSU

誘導オプション	言語	意味
-Knoeval	共通	プログラムに対する演算評価方法を変更する最適化を抑止
-Knofast_matmul	C/C++	行列積のループを高速なライブラリ呼び出し(matmul)への変換を抑止
-Knofp_contract	共通	Floating-Point Multiply-Add/Subtract演算命令を使用した最適化を抑止
-Knofp_relaxed	共通	単精度または倍精度の浮動小数点除算またはSQRT関数について、通常の除算命令またはSQRT命令を利用することを指示
-Knofz	共通	flush-to-zeroモードを使用しない
-Knoilfunc	共通	数学関数のインライン展開を抑止
-Knomfunc	共通	関数をマルチ演算関数に変換する最適化を抑止
-Kparallel_fp_precision	共通	スレッド並列数の変化による浮動小数点または複素数型の演算結果に計算誤差を起こす可能性のある最適化を抑止

C/C++ clangモード

FUJITSU

推奨オプション（C/C++ clangモード）

```
-Nclang -Ofast
```

コンセプト

OSS翻訳に適したclangモードの翻訳オプション指定です。

-Ofastが誘導する翻訳オプション

- C/C++
  -O3 -ffj-fast-matmul -ffast-math -ffp-contract=fast -ffj-fp-relaxed
  -ffj-ilfunc -fbuiltin -fomit-frame-pointer -finline-functions

clangモードの特徴

- モダンなCコード、C++コードの実行性能高速化はtradモード以上
- OSSアプリ翻訳の利便性（GCC互換）はtradモード以上
- ACLEおよびFP16はclangモードのみサポート

-Ofastが誘導する翻訳オプション

FUJITSU

誘導オプション	言語	意味
-O3	共通	最適化レベル3 ループ系最適化やインライン展開などの最適化のほか、高度な最適化を実施する。
-ffj-fast-matmul	C/C++	-Kfast_matmul と同義
-ffast-math	共通	-Keval と同義
-ffp-contract=fast	共通	-Kfp_contract と同義
-ffj-fp-relaxed	共通	-Kfp_relaxed と同義
-ffj-ilfunc	共通	-Kilfunc と同義
-fbuiltin	共通	-Klib と同義 標準ライブラリ関数の動作を認識して、最適化を促進させる。
-fomit-frame-pointer	共通	-Komitfp と同義
-finline-functions	共通	-x- と同義 ソースプログラム上で定義された関数をインライン展開の対象とする。

FUJITSU

C/C++ tradモードと clangモード

- 想定利用シーン
- clangモード 外部仕様概要
- マイクロアーキ(ISA含む)対応比較
- コンパイラ機能（一般、ループ最適化）比較
- コンパイラ機能（リスタ最適化情報）比較

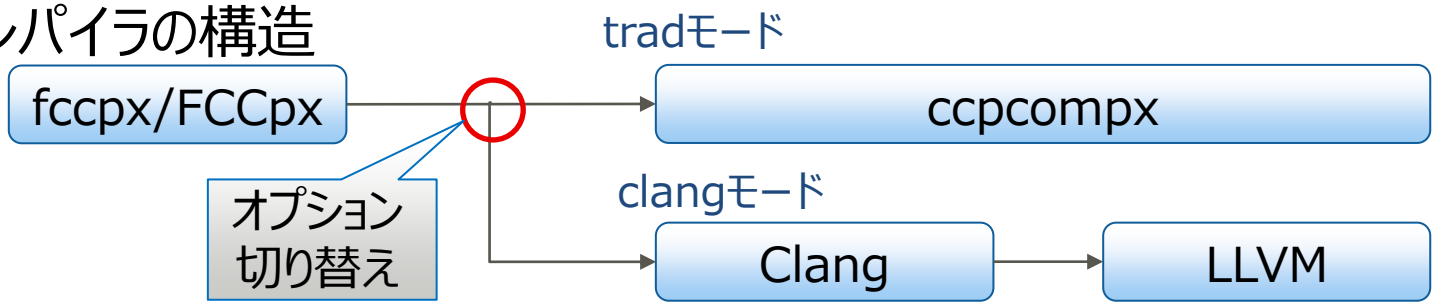
C/C++のtrad/clangモードの選択について

FUJITSU

- 想定している利用シーン（使い分け）

利用シーン	シーンに適したコンパイラ	
	tradモード	clangモード
京,FX100で作成した資源をそのまま利用したい	○	×
HPC向けのチューニングをしたい	○	○ (オプション指定が必要)
OSSを利用したい	×	○
C++17の新しい言語規格を利用したい	○ (一部サポート)	○

- コンパイラの構造



clangモード 外部仕様概要

FUJITSU

- clangモード としての基本方針
  - Clang/LLVM のビューを基本とする
- モードの切り替えオプション
  - -Nclang オプションでコンパイラを切り替える
    例：FCCpx –Nclang test.cpp
- tradモードとの互換を持たせる機能
  - 翻訳時情報出力機能(リスタ機能)
    - Fujitsu 従来コンパイラの出力形式(テキスト形式)はシンプルで可読性が高い
  - ツールの出力情報(ループ関連情報など)
    - ループに対する最適化情報などがチューニングに有効であるため
  - 主要な最適化オプション(-Kオプション)および仕様関連オプション(-Nオプション)
    - 京/FX100の資産を Fujitsu clang モードで翻訳する場合の移植性を配慮

    最適化オプション:

    -Kfast , -Kilfunc , -Keval , -Kfp_contract , -Kfp_relaxed, etc.

    仕様関連オプション:

    -Nhook_time , -Nsrc , -Nlst , -Nline , etc.

マイクロアーキ(ISA含む)対応比較

FUJITSU

項目	tradモード	clangモード	備考
SVE対応	○	○	SIMD化能力は同じではない
fp16サポート	×	○	
可変長SIMD幅指定	○	○	clangモードはデフォルトが可変長SIMD
固定長SIMD幅指定	○	○	tradモードはデフォルトが512bit幅 clangモードについては [clangモード]SVEの固定長SIMD幅指定 を参照
ラージページサポート	○	○	
命令スケジューリングのレイテンシ対応	○	○	
zfill	○	○	
セクタキャッシュ	○	×	
ノード内バリア	○	○	
Prefetch対応	○	△	clangモードは一部未サポート有り
構造体命令活用	○	○	

コンパイラ機能（一般、ループ最適化）比較

FUJITSU

	項目	tradモード	clangモード	備考
一般	OpenMP	○	○	clangモードはOpenMP 4.5をサポート (ただし、declare simd構文およびtaskloop simd構文のlinear指示節は除く)
	自動並列	○	×	
	ACLE	×	○	
	lto	×	○	
ループ最適化	ソフトウェアパイプライニング	○	△	clangモードはオプション指定
	多重ループのループ交換	○	×	
	多重ループの一重化	○	×	
	ループアンローリング	○	○	
	フルアンローリング	○	○	
	ループ分割	△	△	オプション指定
	ループ融合	○	×	
	アンスイッチング	○	○	
	マルチバージョニング	△	△	ともに個別機能のみ

コンパイラ機能（リスタ最適化情報）比較

FUJITSU

	項目	tradモード	clangモード	備考
リスタ最適化表示	SIMD	○	○	
	アンローリング (展開数)	○	○	
	Full unrolling	○	○	
	Software pipelining	○	○	
	Striping/Interleave	○	○	tradモード: STRIPING clangモード: INTERLEAVE
	Unswitching	○	○	
	Loop versioning	○	×	
	Clone	○	○	
	Prefetch (hardware)	○	×	
	Prefetch (software)	○	○	
	Spills	○	○	
	Fused	○	×	
	Collapsed	○	×	
	Interchanged	○	×	
	Fission	○	○	
	Multi-operation function	○	×	
	Pattern matching (matmul)	○	○	
	並列化情報	○	×	
最適化メッセージ		○	△	clangモードはSIMD, アンロール、アンスイッチングは出力

FUJITSU

C/C++ の SVE ACLE 対応について

- C/C++ の SVE ACLE 対応について

C/C++ の SVE ACLE 対応について

FUJITSU

- SIMD組み込み関数として SVE ACLE (Arm C Language Extensions for SVE)が使用可能です。
- C/C++コンパイラの SVE ACLE への対応について、モード(trad/clang)との関係を以下に示します。

○：対応　×：未対応

		SVE	FP16	SVE ACLE (FP16含む)
富士通コンパイラ	trad モード	○	×	×
	clang モード	○	○	○ ※1
他コンパイラ	Arm コンパイラ (19.2)	○	○	○ ※1
	gcc 9.1.0	○	○ ※2	×

※1：Version 00bet1をサポート
https://developer.arm.com/documentation/100987/latest/
(1.1.2 Change historyから差分を確認のこと)
※2：C++ は _Float16 に非対応

FUJITSU

SVEの固定長SIMD幅と可変長SIMD幅

- SVEの固定長SIMD幅と可変長SIMD幅
- [clangモード]SVEの固定長SIMD幅指定

SVEの固定長SIMD幅と可変長SIMD幅

FUJITSU

- SVEのSIMD幅(ベクトルレジスタサイズ)

  SVEは、CPUによってサポートするSIMD幅が異なります(A64FXでは最大512bit)。

  富士通コンパイラではSVEのSIMD幅の差異を考慮し、可変長SIMD幅指定オプションと固定長SIMD幅指定オプションを指定できます。

  - 可変長SIMD幅指定 (tradモード: -Ksimd_reg_size=agnostic, clangモード: -msve-vector-bits=scalable)

    SVEのSIMD幅を特定のサイズとみなさず翻訳を行い、実行時にSIMD幅を決定する実行可能プログラムを作成します。この実行可能プログラムは、CPUに実装されたSVEのSIMD幅によらず実行可能です。C/C++のclangモードのデフォルトです。

  - 固定長SIMD幅指定 (tradモード: -Ksimd_reg_size={128|256|512}, clangモード: -msve-vector-bits=512)

    翻訳時にSIMD幅を固定値とみなすことによって、以下のような最適化の促進が期待できます。FortranとC/C++のtradモードのデフォルトです。C/C++のclangモードでは-ffj-swpオプションと-ffj-zfillオプションから固定長SIMD幅指定オプションが誘導されます。

    - ✓ ソフトウェアパイプライニング
    - ✓ ループインターリーブ
    - ✓ zfill
    - ✓ 数学関数のインライン展開
    - ✓ ループアンローリング

    (など)

[clangモード]SVEの固定長SIMD幅指定(1/2)

FUJITSU

- clangモードで固定長SIMD幅指定でプログラムの性能向上が期待できる例

固定長SIMD幅を指定することで性能向上が期待できるアプリケーションプログラムは、カーネルループでの処理に依存します。一例ですが、以下のようなカーネルループでは固定長SIMD幅指定により各最適化が促進され、性能向上が期待できます。

プログラムの特徴を最も表しているループをカーネルループ と呼んでいます。

```
for(i = 0; i < n; i = i+3) {
  y[i] = x1[i] * x2[i];
}
```

＊ ストライドアクセス
ループインターリーブ、およびソフトウェアパイプライニングが促進されます

```
for (i  =  0; i  <  n; i++) {
  b[i] = 0.9 + a[i] * (0.1 + a[i] *
       (0.2 + a[i] * (0.3 + a[i] *
       (0.4 + a[i] * (0.5 + a[i])))));
  }
}
```

＊ 回転をまたいだ依存がない
ソフトウェアパイプライニングが促進されます

```
double *x, *y;
for(i = 0; i < n; i++) {
  x[i] = sin(y[i]);
}
```

＊ 数学関数を呼び出す
数学関数のインライン展開、および
ソフトウェアパイプライニングが促進されます

[clangモード]SVEの固定長SIMD幅指定(2/2)

FUJITSU

- clangモードにおける固定長SIMD幅指定時の注意点

  clangモードで固定長SIMD幅を指定する際には、以下の点に注意してください。

  - 翻訳時指定値と異なるSIMD幅のCPUで実行した場合の動作は保障されません
  - -fslp-vectorizeオプションは無効となります
  - SIMD組込み関数を使用することはできません
  - -fltoオプションが有効な場合、固定長SIMD幅指定は無効となります
  - 以下のような例では、可変長SIMD幅指定の方が性能向上を期待できます

  ✓ カーネルループ内で構造体の連続した複数メンバや複素数への代入、読み込みがある場合、SVEのload/store multiple structures命令を使用した高速化を期待できます

  ✓ 回転数の少ないループの場合、SIMD幅からはみ出す回転(余りループ)についても、SIMD化が期待できます

FUJITSU

富士通コンパイラと他コンパイラが生成するオブジェクトの互換性について

- オブジェクトの互換性について
- リンク時の注意事項

オブジェクトの互換性について

FUJITSU

- オブジェクトのリンク可否
  - 富士通コンパイラと他コンパイラで生成したオブジェクトのリンク可否
  - 互換性がないオブジェクトをリンクした場合、リンク時にエラー発生

○：リンク可能
×：リンク不可

コンパイラ	言語	富士通コンパイラ (リンク時は富士通コンパイラを使用すること)			
		Fortran	C trad/clangモードの区別をする必要はなし	C++	
				tradモード	clangモード
ARM	Fortran	×	○	×	○
	C	○	○	○※1	○※1
	C++	×	○	×	○
LLVM	Fortran	×	○	×	○
	C	○	○	○※1	○※1
	C++	×	○	×	○
GNU	Fortran	×	○	×	○
	C	○	○	○※1	○※1
	C++	×	○	×	○

※1 富士通コンパイラで作成したオブジェクト同士であっても、C++のclangモードで作成したオブジェクトとC++のtradモードで作成したオブジェクトをリンクした場合には、C++言語使用手引書9.6.1の注意点を参照のこと。

- 注意
  - プロファイラや並列実行デバッガを使用する場合、富士通コンパイラの翻訳コマンドを使用してプログラムの翻訳およびリンクを行ってください。GNUコンパイラなどの他のコンパイラの翻訳コマンドを使用した場合、プロファイラや並列実行デバッガを使用することはできません。

リンク時の注意事項 (1/3)

FUJITSU

- Armコンパイラとのリンク時の注意事項

		富士通コンパイラ ※リンク時は富士通コンパイラを使用すること		
	言語	Fortran	C trad/clangモードの区別をする必要はなし	C++
ARM	Fortran	ー	他コンパイラが独自のライブラリを使うようなプログラムを記述した場合はリンク不可	trad/clangモード共に、他コンパイラが独自のライブラリを使うようなプログラムを記述した場合はリンク不可
	C	スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可)	スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可)	trad/clangモード共に、スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可)
	C++	ー	スレッド並列時では、富士通コンパイラは-Nlibomp オプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可))	clangモードのみ ・スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可) ・STL(C++標準ライブラリは)他社コンパイラと同じものを使用すること - 他社コンパイラで-stdlib=libc++を指定して翻訳した場合は、富士通コンパイラも同様に-stdlib=libc++を指定すること - 他社コンパイラで-stdlib=libstdc++を指定して翻訳した場合は、富士通コンパイラも同様に-stdlib=libstdc++を指定すること

リンク時の注意事項 (2/3)

FUJITSU

- LLVMコンパイラとのリンク時の注意事項

		富士通コンパイラ ※リンク時は富士通コンパイラを使用すること		
	言語	Fortran	C trad/clangモードの区別をする必要はなし	C++
LLVM	Fortran	ー	他コンパイラが独自のライブラリを使うようなプログラムを記述した場合はリンク不可	trad/clangモード共に、他コンパイラが独自のライブラリを使うようなプログラムを記述した場合はリンク不可
	C	スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可)	スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可)	trad/clangモード共に、スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可)
	C++	ー	スレッド並列時では、富士通コンパイラは-Nlibomp オプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可))	clangモードのみ ・スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可) ・STL(C++標準ライブラリは)他社コンパイラと同じものを使用すること - 他社コンパイラで-stdlib=libc++を指定して翻訳した場合は、富士通コンパイラも同様に-stdlib=libc++を指定すること - 他社コンパイラで-stdlib=libstdc++を指定して翻訳した場合は、富士通コンパイラも同様に-stdlib=libstdc++を指定すること

リンク時の注意事項 (3/3)

- GNUコンパイラとのリンク時の注意事項

		富士通コンパイラ ※リンク時は富士通コンパイラを使用すること		
	言語	Fortran	C trad/clangモードの区別をする必要はなし	C++
GNU	Fortran	ー	他コンパイラが独自のライブラリを使うようなプログラムを記述した場合はリンク不可	trad/clangモード共に、他コンパイラが独自のライブラリを使うようなプログラムを記述した場合はリンク不可
	C	スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可)	スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可)	trad/clangモード共に、スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可)
	C++	ー	スレッド並列時では、富士通コンパイラは-Nlibomp オプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可)	clangモードのみ ・スレッド並列時では、富士通コンパイラは-Nlibompオプションを指定すること (-Nfjomplibオプションを使用した場合はリンク不可) ・富士通コンパイラは-stdlib=libstdc++を指定すること

FUJITSU

clangモードがサポートする翻訳時オプションと最適化指示子について

- clangモードの翻訳時オプション体系
- clangモードの最適化指示子体系
- clangモードがサポートするオプション
- clangモードがサポートする最適化指示子

clangモードの翻訳時オプション体系

FUJITSU

- 翻訳時オプションのサポート方針
  - tradモードの -K系列 、-N系列で有用なものをサポート
    - モダンな C コード、OSS アプリの特徴(ループ回転数が少ない等)を考慮し、命令単位の最適化、数学関数向けの最適化のオプションから優先してサポート
  - Clang/LLVM 形式のオプションは、 OSS アプリの翻訳・実行を重視して GCC 互換なものから優先してサポート

- 翻訳時オプションサポートの体系
  - tradモードの -K系列 、-N系列のオプション
    - Clang/LLVM に同等なオプションが有る場合
      - ✓ -K系列 、-N系列のオプションを、Clang/LLVM の同等なオプション(-f 系列)に変換
    - Clang/LLVM に同等なオプションが無い場合
      - ✓ 新機能 & 新オプション(-ffj 系列)を追加し、-K系列 、-N系列のオプションを -ffj 系列に変換
      - ✓ 新機能 & 新オプション(-ffj 系列)は、今後、拡充していく予定
  - Clang/LLVM形式のオプション
    - clangモードのマニュアルに記載している範囲をサポート
      - ✓ 記載されていないオプションについては、使用は可能だが、サポートの対象外

clangモードの最適化指示子体系

FUJITSU

- 最適化指示子のサポート方針
  - Clang/LLVM の最適化指示子に加えて、富士通コンパイラの最適化指示子の有用なものをサポート

- 最適化指示子サポートの体系
  - Clang/LLVM の最適化指示子
    - clangモードのマニュアルに記載している範囲をサポート
      - ✓ 記載されていない最適化指示子については、使用は可能だが、サポートの対象外
  - 富士通独自の最適化指示子
    - #pragma fj の形式で富士通独自の最適化指示子をサポート
      - ✓ tradモードでも #pragma fj の形式をサポートすることで、clangモード・tradモード間の互換性を確保
      - ✓ tradモードとの互換性の向上および機能拡張に応じてサポート範囲を拡充予定

clangモードがサポートするオプション(1/7)

FUJITSU

- 最適化関連 (Clang/LLVMと同じ)

clangモードのオプション	概要	tradモードのオプション
-Ofast	ターゲットマシン上で高速に実行するオブジェクトプログラムを作成	-Kfast
-f{builtin|no-builtin}	標準ライブラリ関数の動作を認識して最適化を促進させるか	-K{lib|nolib}
-f{lto|no-lto}	リンク時最適化を行うか	-K{lto|nolto}
-f{vectorize|no-vectorize}	ループ内の演算に対し、SIMD拡張命令を利用したオブジェクトを生成するか	-K{simd|nosimd}
-f{strict-aliasing|no-strict-aliasing}	言語規格で規定された厳密なaliasing ruleに従って、メモリ領域の重なりを考慮した最適化を行うか	-K{strict_aliasing|nostrict_aliasing}
-f{unroll-loops|no-unroll-loops}	ループアンローリングの最適化を行うか	-K{unroll|nounroll}
-f{omit-frame-pointer|no-omit-frame-pointer}	関数呼出しにおける、フレームポインタレジスタを保証しない最適化を行うか	-K{omitfp|noomitfp}
-f{inline-functions|no-inline-functions}	ソースプログラム上で定義された関数をインライン展開の対象にするか	{-x-|-x0}
-f{fast-math|no-fast-math}	演算の評価方法を変更する最適化を行うか	-K{eval|noeval}
-ffp-contract={fast|on|off}	Floating-Point Multiply-Add/Subtract演算命令を使用した最適化を行うか	-K{fp_contract|nofp_contract}

clangモードがサポートするオプション(2/7)

FUJITSU

- 最適化関連 (Clang/LLVMと同じ2)

clangモードのオプション	概要	tradモードのオプション
-msve-vector-bits={512|scalable}	SVEのベクトルレジスタサイズを指定する(単位はビット)	(なし)
-f{finite-math-only|no-finite-math-only}	引数または演算結果において有限の数値のみであるということを仮定し、浮動小数点演算の最適化を促進させるか否かを指示する	(なし)
-f{reroll-loops|no-reroll-loops}	ループリローリングの最適化を行うか否かを指する	(なし)
-f{signed-char|unsigned-char}	char 型で宣言した変数を singed char 型として扱うことを指示する	(なし)
-f{slp-vectorize|no-slp-vectorize}	スーパーワードレベルの並列化を行うことを指示する	(なし)

clangモードがサポートするオプション(3/7)

FUJITSU

- 最適化関連 (富士通固有、プリフェッチ系以外)

clangモードのオプション	概要	tradモードのオプション
-f{fj-eval-concurrent| fj-no-eval-concurrent}	tree-height-reduction最適化において、浮動小数点演算命令の並列性を優先するか	-K{eval_concurrent| eval_noconcurrent}
-f{fj-fast-matmul| fj-no-fast-matmul}	行列積のループを高速なライブラリ呼び出しに変換するか	-K{fast_matmul| nofast_matmul}
-f{fj-fp-relaxed| fj-no-fp-relaxed}	単精度浮動小数点除算、倍精度浮動小数点除算およびsqrt関数について、逆数近似演算命令とFloating-Point Multiply-Add/Subtract命令を利用した逆数近似演算を行うか	-K{fp_relaxed|nofp_relaxed}
-f{fj-ilfunc[={loop|procedure}]| fj-no-ilfunc}	数学関数をインライン展開するか	-K{ilfunc[={loop|procedure}]| noilfunc}
-f{fj-ocl|fj-no-ocl}	clangモードがサポートする富士通コンパイラ独自の最適化制御行(プラグマディレクティブ)を有効にするか	-K{ocl|noocl}
-f{fj-preex|fj-no-preex}	不変式の先行評価の最適化を行うか	-K{preex|nopreex}

clangモードがサポートするオプション(4/7)

FUJITSU

- 最適化関連 (富士通固有、プリフェッチ系以外2)

clangモードのオプション	概要	tradモードのオプション
-f{fj-fp-precision| fj-no-fp-precision}	浮動小数点演算の計算誤差が生じないようなオプションの組合せを誘導する	-K{fp_precision| nofp_precision}
-f{fj-hpctag|fj-no-hpctag}	A64FXプロセッサのHPCタグアドレスオーバライド機能を利用する	-K{hpctag|nohpctag}
-f{fj-interleave-loop-insns[=N]| fj-no-interleave-loop-insns}	SVEを利用してSIMD化されたループに対して、ループインターリーブの最適化を行う(Nはインターリーブ展開数)	(なし)
-f{fj-loop-fission| fj-no-loop-fission}	ソフトウェアパイプライニングの促進、SIMD化の促進、およびレジスタ不足の解消のために、ループを複数のループに分割する最適化を行う	-K{loop_fission| loop_nofission}
-f{fj-loop-fission-threshold=N}	自動ループ分割における分割後のループの粒度(ループ内の命令数やレジスタ数など)を決める閾値Nを指示する(Nは1から100までの整数値)	-K{loop_fission_threshold=N}
-f{fj-optlib-string| fj-no-optlib-string}	文字列操作関数において、最適化版のライブラリをリンクする	-K{optlib_string| nooptlib_string}
-f{fj-swp|fj-no-swp}	ソフトウェアパイプライニングを行う	-K{swp|noswp}
-f{fj-zfill[=N]| fj-no-zfill}	zfillの最適化を行う	-K{zfill[=N]|nozfill}

clangモードがサポートするオプション(5/7)

FUJITSU

- 最適化関連 (富士通固有、プリフェッチ系)

clangモードのオプション	概要	tradモードのオプション
-ffj-prefetch-cache-level={1|2|all}	どのキャッシュレベルにデータをプリフェッチするか	-Kprefetch_cache_level={1|2|all}
-f{fj-prefetch-conditional|fj-no-prefetch-conditional}	if文やswitch文に含まれるブロックの中で使用される配列データに対してprefetch命令を生成するか	-K{prefetch_conditional|prefetch_noconditional}
-ffj-prefetch-iteration=**N**	prefetch命令を生成する際、ループの**N**回転後に引用されるデータを対象とする (1次キャッシュ用)	-Kprefetch_iteration=**N**
-ffj-prefetch-iteration-L2=**N**	同上 (2次キャッシュ用)	-Kprefetch_iteration_L2=**N**
-ffj-prefetch-line=**N**	prefetch命令を生成する際、**N**キャッシュライン先に該当するデータをプリフェッチの対象とする (1次キャッシュ用)	-Kprefetch_line=**N**
-ffj-prefetch-line-L2=**N**	同上 (2次キャッシュ用)	-Kprefetch_line_L2=**N**
-f{fj-prefetch-sequential[={auto|soft}]|fj-no-prefetchsequential}	ループ内で使用される連続的にアクセスされる配列データに対して、prefetch命令を生成するか	-K{prefetch_sequential[={auto|soft}]|prefetch_nosequential}
-f{fj-prefetch-stride|fj-no-prefetch-stride}	ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対してprefetch命令を生成するか	-K{prefetch_stride|prefetch_nostride}
-f{fj-prefetch-strong|fj-no-prefetch-strong}	prefetch命令を生成する際、strong prefetch命令を生成するか (1次キャッシュ用)	-K{prefetch_strong|prefetch_nostrong}
-f{fj-prefetch-strong-L2|fj-no-prefetch-strong-L2}	同上 (2次キャッシュ用)	-K{prefetch_strong_L2|prefetch_nostrong_L2}
-ffj-no-prefetch	prefetch命令を使用したオブジェクトを生成しない	-Knoprefetch

clangモードがサポートするオプション(6/7)

FUJITSU

- CPU/アーキテクチャ関連、コード生成関連、OpenMP関連

clangモードのオプション	概要	tradモードのオプション
-mcpu=a64fx	A64FXプロセッサ向けのオブジェクトファイルを出力	-KA64FX
-mcpu=generic	Armプロセッサ(汎用)向けのオブジェクトファイルを出力	-KGENERIC_CPU
-march= arm{v8|v8.1|v8.2|v8.3}-a+sve	Armv8.x-Aの命令とSVE命令を利用	-KARCH -KSVE
-march= arm{v8|v8.1|v8.2|v8.3}-a+nosve	Armv8.x-Aの命令を利用 (SVE命令は利用しない)	-KARCH -KNOSVE
-mcmodel={small|large}	実行可能プログラムおよび共有オブジェクトの、コード領域と静的データ領域の最大値	-Kcmodel={small|large}
-f{PIC|pic}	位置独立コード(PIC)を生成	-K{PIC|pic}
-f{fj-largepage|fj-no-largepage}	ラージページ機能を使用するか	-K{largepage|nolargepage}
-f{openmp|no-openmp}	OpenMP仕様の指示文を受け入れるか	-K{openmp|noopenmp}
-f{openmp-simd|no-openmp-simd}	OpenMP仕様のsimd構文、declare reduction構文、およびsimd指示節をもつordered構文のみを有効にするか	-K{openmp_simd|noopenmp_simd}

clangモードがサポートするオプション(7/7)

FUJITSU

- その他のオプション

clangモードのオプション	概要	tradモードのオプション
-f{fj-fjcex| fj-no-fjcex}	clangモードが提供する富士通拡張関数を利用するか	-N{fjcex|nofjcex}
-f{fj-fjprof| fj-no-fjprof}	プロファイラ機能を有効にするか	-N{fjprof|nofjprof}
-f{fj-line|fj-no-line}	プロファイラで提供される実行時間のサンプリング機能に必要な追加情報を生成するか	-N{line|noline}
-f{fj-hook-time| fj-no-hook-time}	一定時間間隔での呼出しによるフック機能を利用するか	-N{hook_time|nohook_time}
-ffj-lst[=p]	翻訳情報として、ソースリストと統計情報をファイルに出力する	-Nlst[=p]
-ffj-lst=t	翻訳情報として、ソースリスト、統計情報、より詳細な最適化情報をファイルに出力する	-Nlst=t
-ffj-lst-out=**file**	翻訳情報を指定されたファイル名**file**に出力する	-Nlst_out=**file**
-ffj-src	ソースリストを標準出力に出力する	-Nsrc
-Rpass=.*	最適化情報を標準エラー出力に出力する	-Koptmsg=2
--version	コンパイラのバージョンと著作権の情報を標準出力に出力	-V

clangモードがサポートする最適化指示子(1/3)

FUJITSU

- Clang/LLVM と同じもの

先頭の「#pragma」は省略

clangモードの最適化指示子	概要	tradモードの最適化指示子
clang fp contract(fast)	FMA命令を使用する	(なし)
clang fp contract(on)	FMA命令を使用する (#pragma STDC FP_CONTRACT ON と等価)	(なし)
clang fp contract(off)	FMA命令を使用しない	(なし)
clang loop unroll(enable)	ループをアンローリングする	loop unroll
clang loop unroll(full)		loop unroll full
clang loop unroll_count(n)		loop unroll(n)
clang loop unroll(disable)	ループをアンローリングしない	loop nounroll
clang loop vectorize(enable)	ループをSIMD化する	loop simd
clang loop vectorize_width(n, scalable)	ループをSIMD化する、SVEを利用したSIMD化においてSIMD長がnの定数倍になる	(なし)
clang loop vectorize(assume_safety)	ループ中で配列の要素またはポインタ変数に対して依存がないことを指示する	(なし)
clang loop vectorize(disable)	ループをSIMD化しない	loop nosimd

clangモードがサポートする最適化指示子(2/3)

FUJITSU

- 富士通固有 (プリフェッチ系以外)

先頭の「#pragma」は省略

clangモードの最適化指示子	概要	tradモードの最適化指示子
fj loop eval_concurrent	tree-height-reduction最適化において、浮動小数点演算命令の並列性を優先する	loop eval_concurrent
fj loop eval_noconcurrent	tree-height-reduction最適化において、浮動小数点演算命令の並列性を抑え、FMA命令の利用を優先する	loop eval_noconcurrent
fj loop preex	不変式の先行評価を行う	loop preex
fj loop nopreex	不変式の先行評価を行わない	loop nopreex
fj loop clone var==n	指定された変数varと値nの等式を条件式とする分岐を生成し、ループを複製する	loop clone var==n
fj loop loop_fission_target [cl]	コンパイラによる自動ループ分割を行う	loop loop_fission_target [cl]
fj loop loop_fission_threshold n	自動ループ分割における分割後のループの粒度を決める閾値nを指示する (n=1～100)	loop loop_fission_threshold n
fj loop swp	ソフトウェアパイプライニングを行う	loop swp
fj loop noswp	ソフトウェアパイプライニングを行わない	loop noswp
fj loop zfill [n]	zfillの最適化を有効にする (n=1～100)	loop zfill [n]
fj loop nozfill	fillの最適化を無効にする	loop nozfill

clangモードがサポートする最適化指示子(3/3)

FUJITSU

- 富士通固有 (プリフェッチ系)

先頭の「#pragma」は省略

clangモードの 最適化指示子	概要	tradモードの 最適化指示子
fj loop prefetch	コンパイラの自動prefetch機能を有効化する	loop prefetch
fj loop noprefetch	コンパイラの自動prefetch機能を無効化する	loop noprefetch
fj loop prefetch_sequential [auto|soft]	連続的にアクセスされる配列データに対してprefetch命令を生成する	loop prefetch_sequential [auto|soft]
fj loop prefetch_nosequential	連続的にアクセスされる配列データに対してprefetch命令を生成しない	loop prefetch_nosequential
fj loop prefetch_stride	ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対してprefetch命令を生成する	loop prefetch_stride
fj loop prefetch_nostride	ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対してprefetch命令を生成しない	loop prefetch_nostride
fj loop prefetch_strong	1次キャッシュに対して生成されるprefetch命令をstrong prefetchとする	loop prefetch_strong
fj loop prefetch_nostrong	1次キャッシュに対して生成されるprefetch命令をstrong prefetchとしない	loop prefetch_nostrong
fj loop prefetch_strong_L2	2次キャッシュに対して生成されるprefetch命令をstrong prefetchとする	loop prefetch_strong_L2
fj loop prefetch_nostrong_L2	2次キャッシュに対して生成されるprefetch命令をstrong prefetchとしない	loop prefetch_nostrong_L2

2つのOpenMPライブラリ

- LLVM OpenMPライブラリと富士通OpenMPライブラリ
  - LLVM OpenMP ライブラリ (-Nlibomp)
  - 富士通 OpenMPライブラリ (-Nfjomplib)

LLVM OpenMPライブラリと富士通OpenMPライブラリ (1/2)

FUJITSU

- 富士通開発のfjomplib(富岳/FX1000/FX700)とOSSのlibompを選択可能です。
- OpenMPの新規仕様サポートを重視し、libompがデフォルトです。
  - 特徴

	概要	ループ並列	サポート仕様
libomp	LLVMで使わている OpenMP用ランタイムライブラリ	◎ ハードバリアをサポート (環境変数 FLIB_BARRIER = HARDの指定が必要)	OpenMP4.5 と 5.0 の一部
fjomplib	富士通開発の OpenMPライブラリ	◎ ハードバリアをサポート	OpenMP3.1 (タスク並列を利用する場合はlibompを推奨)

  - コンパイラとの組み合わせ
    - FortranとC/C++のtradモードではオブジェクトファイルは共通で、使用するライブラリを -Nfjomplip / libompオプションで指定可能です。(clangモードのオブジェクトファイルが含まれる場合は、libompのみ選択可能)

	Fortran	C/C++	
		tradモード	clangモード
libomp	○	○	○
fjomplib	○	○	×

LLVM OpenMPライブラリと富士通OpenMPライブラリ (2/2)

FUJITSU

- 選択方法
  - 翻訳時オプションで選択
    - -Nlibomp (デフォルト)：libompを使用
    - -Nfjomplib：富士通開発のfjomplibを使用
- 仕様の違い
  - スレッドスタックの大きさ

	デフォルトの大きさ	大きさ変更用環境変数
libomp	8MiB	OMP_STACKSIZE
fjomplib	• プロセススタックのサイズを継承 • プロセススタックサイズがunlimited指定の場合 (メモリサイズ / スレッド数) / 5	OMP_STACKSIZE または THREAD_STACK_SIZE

  - 環境変数
    - libompでは、以下の環境変数は未サポート(指定しても無視される)です。
      - 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
  - 共有ライブラリ
    - libompでは、以下の共有ライブラリがリンクされます。
      - libfjomp.so, libfjompcrt.so, libfjomphk.so

LLVM OpenMP ライブラリ (-Nlibomp) (1/2)

FUJITSU

- デフォルトの状態でソフトウェアバリア/セクタキャッシュが利用できます。
- ハードウェアバリアを利用するためには、環境変数 FLIB_BARRIERを指定します。
  - FLIB_BARRIER=HARD ：ハードウェアバリアを利用します。
  - FLIB_BARRIER=SOFT ：ソフトウェアバリアを利用します。（デフォルト）
- ハードウェアバリアを利用する場合の注意事項
  - スレッド数の制御(omp_set_num_threads()、num_threads 指示節)はできません。
  - スレッドアフィニティ(proc_bind指示節、OMP_PLACES、OMP_PROC_BIND)の制御はできません。
  - ネスト制御(omp_set_nested() 、OMP_NESTED)はできません。
  - タスク構文において常に即時実行タスク(undeffered task)が生成され、タスクは並列実行されません。
  - キャンセレーションは利用はできません
- OpenMP 4.5 に加えて OpenMP 5.0 の一部をサポートしています。最新の OpenMP規格を利用する場合は本ライブラリを利用してください。
- タスクやキャンセレーションなどの OpenMP4.5/5.0 の主要機能を利用する場合は、ソフトウェアバリアを選択してください。

LLVM OpenMP ライブラリ (-Nlibomp) (2/2)

FUJITSU

- 初期スレッドを特定のコアにバインドする場合は、numactl や taskset を利用してください。プログラム中の上限スレッド数に影響はありません。

```
#!/bin/sh -ex
  :
export FLIB_BARRIER=HARD   # ハードウェアバリアを利用
                           # （FLIB_BARRIERがSOFTか、未設定時はソフトウェアバリアになります）
numactl -C12 ./a.out       # 初期スレッドをC12に固定
```

- MPIプログラムの場合、MPIライブラリ内メモリコピー処理のスレッド並列化機能を利用することができます。翻訳／結合コマンドのオプションとして、-Kparallelと-Kopenmpの両方またはどちらか1つと、-Nlibompを指定してください。
  - mpifrtpx –Kopenmp –Nlibomp a.f90
- 以下の富士通 OpenMPライブラリ独自の環境変数は利用できません。設定しても無視されます。
  - 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

富士通 OpenMPライブラリ (-Nfjomplib)

FUJITSU

- 基本的な仕様は従来システム(京/FX100)と同じであるため、詳細な説明は省略します。
- デフォルトの状態でハードウェアバリア/セクタキャッシュが利用できます。
- ハードウェアバリア利用時に即時実行タスクが生成される制約はありません。
- OpenMP 3.1 をサポートしています。
- Fortran、C/C++ tradモードで使用する場合は -Nfjomplib をリンク時に指定してください。
  - Fortran
    - frtpx -Kopenmp -Nfjomplib main.f90
  - C/C++ (tradモード)
    - fccpx -Kopenmp -Nfjomplib main.c
- C/C++コンパイラの clangモードでは –Nfjomplib は使用できません。
- ジョブ実行時、スレッドはコアバインドされます。ジョブ実行しない場合、スレッドはコアバインドされませんが、環境変数FLIB_CPU_AFFINITYを使用することでコアバインドの制御が行えます。
- 以下のLLVM OpenMPライブラリ独自の環境変数は利用できません。設定しても無視されます。
  - FLIB_BARRIER

性能の確認(基礎性能)

FUJITSU

- OpenMP Microbench (Fortran、12スレッド実行)

処理	【参考】Skylake (2.7GHz) 実行時間 (μs)	libomp ソフトバリア 実行時間 (μs)	libomp ハードバリア 実行時間 (μs)	fjomplib ハードバリア 実行時間 (μs)	ハードバリア比較 (libomp/fjomplib)	Skylake比較 (libomp(hard)との比較)
PARALLEL	1.10	2.95	2.24	0.43	5.25	2.03
DO/FOR	0.82	1.60	0.13	0.13	0.97	0.16
PARALLEL_DO/FOR	1.12	2.95	2.23	0.45	5.00	2.00
BARRIER	0.76	1.55	0.12	0.12	1.00	0.16
SINGLE	1.18	1.68	1.37	0.60	2.26	1.16
CRITICAL	0.28	0.32	0.32	0.66	0.49	1.14
LOCK/UNLOCK	0.29	0.32	0.32	0.48	0.67	1.12
ORDERED	0.26	0.32	0.32	0.28	1.12	1.22
ATOMIC	0.19	0.65	0.69	0.69	1.00	3.65
REDUCTION	2.25	4.63	2.59	0.95	2.72	1.15

Libomp時、PARALLEL構文は要注意

従来システムからの移行

- A64FXで追加した新規オプション
- A64FXで仕様変更したオプション
- A64FXで廃止したオプション
- 京/FX10/FX100 ⇒ A64FX の互換情報
  - Fortran の互換項目
  - C の互換項目
  - C++ の互換項目
  - MPI の互換項目
  - 数学ライブラリ の互換項目
- MPI の拡張関数とデータ型
- エンディアン

A64FXで追加した新規オプション（1/8）

翻訳オプション	言語	意味
-Kswp_weak ★	共通	ソフトウェアパイプライニングを調整し、ループ内の実行文の重なりを小さくすることを指示する。実行時にソフトウェアパイプライニングが適用されたルートを通るために必要なループの繰返し数が小さくなるため、ループの繰返し数が翻訳時に不明であり、かつループの繰返し数が小さい場合に性能向上が期待できる。
-Kswp_freg_rate=N ★ -Kswp_ireg_rate=N ★ -Kswp_preg_rate=N	共通	ソフトウェアパイプライニングで使用可能なレジスタ数に関する条件を変更する。Nは1～1000 の整数値で、ソフトウェアパイプライニングで使用可能なレジスタ数の割合を百分率で指示する。100より大きな整数値を指定することで、ソフトウェアパイプライニングが適用できる場合がある。ただし、レジスタのメモリへの退避・復元命令が変化し、実行性能が低下する場合がある。 - freg: 浮動小数点レジスタおよびSVEのベクトルレジスタ - ireg: 整数レジスタ - preg: SVEのプレディケートレジスタ
-Kloop_fission_threashold=N ★	共通	ループ分割後のループの粒度（ループ内の命令数やレジスタ数など）を決める閾値を指示。Nの範囲は1～100で、defaultは50。Nを小さくすると、分割後のループが小さくなり、分割数が増える。
-Kloop_fission_ stripmining[={N|L1|L2}] -Kloop_fission_nostripmining	共通	-Kloop_fission_stripmining[={N|L1|L2}] ループ分割時にストリップマイニング最適化を指示。 Nはストリップの長さ1～100,000,000の範囲を指定。文字L1,L2を指定した場合、指定キャッシュを意識した長さに合わせる。指定がない場合は、コンパイラが自動で判断。

★ 注目すべきオプション

A64FXで追加した新規オプション（2/8）

FUJITSU

翻訳オプション	言語	意味
-Kassume ={shortloop| memory_bandwidth| time_consuming_compilation} ★	共通	-O1以上でオプション指定動作。 -Kassume=shortloop プログラム中の最内ループの回転数が翻訳時に不明な場合に、回転数が小さいとみなして、最適化制御 -Kassume=memory_bandwidth プログラム中の最内ループをメモリバンド幅がボトルネックとみなして最適化制御 -Kassume=time_consuming_compilation 翻訳時間が短くなるように最適化を制御（-Oのレベル制御のように、ある特定機能が止まるのではなく、プログラムが巨大になるほど最適化が制限/抑止される）
-Keval_[no]concurrent ★	共通	-Keval_concurrent tree-height-reduction最適化において、演算命令の並列性を優先する変形を実施。-O1以上かつ-Keval時、オプション指定で動作。 Software Pipeliningできなかった演算が多いループに適用することで、性能向上の可能性有。 defaultの-Keval_noconcurrentは、Software Pipeliningとの連携を配慮し、命令の並列性を抑え、FMA命令の利用を優先する変形を実施。
-Kloop_[no]perfect_nest	共通	-Kloop_noperfect_nest 不完全多重ループを分割し、完全多重ループにする最適化の抑止を指示。 本オプションは、既存で動作していた機能をオプション化。 -O2以下のdefaultは-Kloop_noperfect_nest、 -O3のdefaultは-Kloop_perfect_nest

A64FXで追加した新規オプション（3/8）

FUJITSU

翻訳オプション	言語	意味
-K[no]optlib_string ★	共通	-Koptlib_string 文字列操作関数の最適化版ライブラリをリンク。 defaultは、-Knooptlib_string。
-K[no]preload ★	共通	-Kpreload ifのthen/else節からif条件判定前にロード命令を投機実行することで、実行モジュールの高速化を狙う最適化を実施。 defaultは、-Knopreload。
-K[no]sibling_calls	共通	-Ksibling_calls 末尾呼出しの最適化を実施。-O2以上でdefault動作。
-Ksimd_reg_size ={128|256|512|agnostic}	共通	指定された値がSVEのベクトルレジスタのサイズとみなして命令を生成。 **agnostic**の指定では、CPUのSVEのベクトルレジスタのサイズに関わらず実行可能な命令を生成。
-Ksimd_[no]use_multiple _structures	共通	-Ksimd_nouse_multiple_structures SVEのload/store multiple structues命令を利用しないことを指示。 -Ksimdおよび-KSVE有効時のみ利用可能。 defaultは、-Ksimd_use_multiple_structures。
-Ksimd_[no]uncounted_loop	共通	-Ksimd_uncounted_loop do whileループ、do untilループおよびループを終了する文を含むdoループに対してSIMD化を実施（最適化対象は限定された範囲のみ） defaultは、-Ksimd_nouncounted_loop。

A64FXで追加した新規オプション（4/8）

FUJITSU

翻訳オプション	言語	意味
-K[no]sch_pre_ra ★	共通	-Knosch_pre_ra レジスタ割付前命令スケジューラ抑止。-O1以上で-Ksch_pre_ra適用。 SPILLが多い時に利用すると性能向上の可能性有。
-K[no]sch_post_ra	共通	-Knosch_post_ra レジスタ割付後命令スケジューラ抑止。-O1以上で-Ksch_post_ra適用
-Kunroll_and_jam[=N] -Knounroll_and_jam	共通	-Kunroll_and_jam[=N] アンロールアンドジャム(外側ループアンローリング)最適化を実施。Nには、 ループ展開数の上限を2～100の範囲で設定。-O2以上でオプション指定動作。
-K{align_loops[=N] |noalign_loops}	共通	-Kalign_loops[=N] ループの先頭alignmentを2の累乗バイト境界に合わせる。 Nはループの先頭アライメントのバイト境界の値で0～32,768までの2の累乗で指定。Nの指定省略またはN=0はコンパイラが自動決定。-O2以上でdefault動作。
-Kopenmp_[no]collapse _except_innermost	共通	-Kopenmp_collapse_except_innermost 最内をcollapse対象から外す。これにより、collapseによる実行性能の低下を防止できる場合がある。-Kopenmp指定時に有効なオプション。 defaultは、-Kopenmp_nocollapse_except_innermost。
-K[no]openmp_simd	共通	-Kopenmp_simd OpenMP仕様のSIMD構文、DECLARE SIMD構文のみを有効にする。
-Kcmodel={small|large}	共通	メモリモデルの指示。defaultはsmall。

A64FXで追加した新規オプション（5/8）

FUJITSU

翻訳オプション	言語	意味
-K[no]pc_relative_literal_loads	共通	-Kpc_relative_literal_loads 手続き内のコード領域が1MB以内であるとして扱い、リテラルプールに1命令でアクセスする。defaultは、-Knopc_relative_literal_loads。
-K[no]plt	共通	-Kplt 位置独立コード（PIC）での外部シンボルへのアクセスにProcedure Linkage Table（PLT）を使用するかどうかを指示。-K{pic|PIC}時に意味があり、defaultは-Kplt。
-Ktls_size={12|24|32|48} 	共通	スレッド・ローカル・ストレージ(Thread-Local Storage)へのアクセスに必要なオフセットのサイズを指定。単位はビット。
-KARM{V8_A|V8_1_A|V8_2_A|V8_3_A}	共通	指定された命令セットを含むオブジェクトファイルを生成。defaultは、-KARMV8_3_A。
-K[NO]SVE	共通	SVE拡張命令を利用するかどうかを指示。defaultは-KSVE。
-KA64FX	共通	A64FXプロセッサ向けオブジェクト生成を指示(default)
-KGENERIC_CPU	共通	汎用ARMプロセッサ向けオブジェクト生成を指示
-K[no]hpctag	共通	-Khpctag A64FXプロセッサのHPCタグアドレスオーバライド機能を利用することを指示。HPCタグアドレスオーバライド機能により、セクタキャッシュ機能やハードウェアプリフェッチアシスト機能（ストライドアクセスに対するハードウェアプリフェッチ機能など）が有効となる。 富岳ではdefaultで有効。 上記の機能を一括で抑止する方法として、-Knohpctagを利用。

A64FXで追加した新規オプション（6/8）

FUJITSU

翻訳オプション	言語	意味
-K[no]array_declaration_opt	共通	最適化を行うときに、配列の添字が配列宣言の範囲を超えないことを前提にするかどうかを指示。-O1以上でdefault動作。
-K[no]extract_stride_store	共通	SIMD化対象ループにあるストライドアクセスのストア命令を、スカラ命令に展開するかどうかを指示。 defaultは-Knoextract_stride_store
-K[no]fp_precision ★	共通	浮動小数点演算の計算誤差が生じないようなオプションの組合せを誘導するかどうかを指示。 defaultは-Knofp_precision
-K[no]subscript_opt	F	配列の添字の近傍データの利用を優先した最適化(例えば、ステンシル計算)を行うかどうかを指示。 defaultは-Knosubscript_opt
-Kswp_policy= { auto | small | large } ★	共通	ソフトウェアパイプライニングで使用する命令スケジューリングアルゴリズムの選択基準を指示。 defaultは-Kswp_policy=auto swp_policy=auto ループ毎に命令スケジューリングアルゴリズムを自動で選択 swp_policy=small 小さなループ(例えば、必要レジスタ数が少ないループ)に適した命令スケジューリングアルゴリズムを使用 swp_policy=large 大きなループ(例えば、必要レジスタ数が多いループ)に適した命令スケジューリングアルゴリズムを使用

A64FXで追加した新規オプション（7/8）

FUJITSU

翻訳オプション	言語	意味
-X08	F	Fortran 2008言語仕様レベルオプション。 a.f08, a.F08, a.for, a.FORの翻訳で誘導。
-std={c11|gnu11}	C	GNU11でコンパイル(default)
-std={c++17|gnu++17}	C++	GNU++17でコンパイル
-N[no]clang ★	C/C++	clang/LLVMベース(clangモード)のコンパイラの動作
-N[no]coverage --[no-]coverage	共通	-Ncoverage / --coverage コードカバレッジ機能を利用するための情報を生成 defaultは-Nnocoverage / --no-coverage
-Nnocheck_global	F	-Nnocheck_global -Ncheck_globalの否定オプションを追加。 -Nquickdbgおよび-Egから-Ncheck_globalが誘導されるが、翻訳時のエラー検出は行わず、実行時だけの検査を行いたい要件に対応。 -Nquickdbg -Nnocheck_globalと指定する。
-Nfmtl ={serial|SSL2|parallel|SSL2BLAMP}	C++	富士通マトリックステンプレートライブラリの利用を指示。 • serial: 逐次版 • SSL2:SSL2ライブラリを使用して高速化した逐次版 • parallel:スレッド並列版 • SSL2BLAMP:SSL2ライブラリを使用して高速化したスレッド並列版

A64FXで追加した新規オプション（8/8）

FUJITSU

翻訳オプション	言語	意味
-Nfjomplib ★	共通	並列処理に使用するライブラリとして、富士通独自 OpenMPライブラリを利用
-Nlibomp ★	共通	並列処理に使用するライブラリとして、LLVM OpenMPライブラリを利用 clangモードの時は常に-Nlibomp
-N[no]reordered_variable_stack	共通	-Nreordered_variable_stack データサイズの昇順に自動変数をスタック領域に割り付ける。 defaultは-Nnoreorederd_variable_stackで、プログラムの宣言文順に割り付ける。

A64FXで仕様変更したオプション（1/7）

FUJITSU

翻訳オプション	言語	意味 変更理由	旧仕様	新仕様
-Kfast	共通	ターゲットマシン上で高速に実行するオブジェクトプログラムの作成 汎用的な性能向上を目指す対応。 -Kprefetch_conditionalは性能低下要因となりうるため削除。 その他機能の仕様変更等に伴い変更。	Fortran -O3 -K**dalign**,eval, fp_contract,fp_relaxed, ilfunc,mfunc,**ns**,omitfp,**prefetch_conditional** C/C++ -O3 -K**dalign**,eval, fast_matmul,fp_contract, fp_relaxed,ilfunc,lib,mfunc, **ns**,omitfp,**prefetch_conditional**,rdconv -x-	Fortran -O3 -Keval,fp_contract, fp_relaxed,**fz**,ilfunc,mfunc,omitfp,**simd_packed_promotion** C/C++ -O3 -Keval,fast_matmul, fp_contract,fp_relaxed, **fz**,ilfunc,mfunc,omitfp, **simd_packed_promotion**
-Kauto	F	SAVE属性をもつ変数および初期値をもつ変数を除く局所変数をスタックに割り付け default高速化のため	個別指定で動作	-O0からdefault動作
-Ktemparraystack	F	配列演算の途中結果およびマスク式評価結果をスタックに割付け default高速化のため	個別指定で動作	-O0からdefault動作
-Kautoobjstack	F	自動割付けデータ実体をスタックに割付け default高速化のため	個別指定で動作	-O0からdefault動作

A64FXで仕様変更したオプション（2/7）

FUJITSU

翻訳オプション	言語	意味 変更理由	旧仕様	新仕様
-Klib	C/C++	sin等をユーザ定義関数ではなく、標準ライブラリ関数と認識 業界標準に合わせた	-O1以上で有効 -Kfastから誘導	-O0以上で有効 -O1からdefault動作 ★
-x-	C/C++	インライン展開の実施 -Xg default化に伴う変更	-Kfastから誘導	-O3から誘導
-x=quick	C++	インラインによる関数の巨大化防止 汎用的に効果が望めるため	-x=noquick(default)	-x=quick(default)
-Xg	C/C++	GNU C/C++コンパイラ仕様の言語仕様に基づいて翻訳することを指示 GNU C/C++コンパイラの拡張仕様をdefaultで受け付けるようにするため	3つの言語仕様モード -Xa, -Xc, -Xg defaultは-Xa、他はオプショナル ※旧仕様は廃止を通知し、GNU C/C++互換モードで翻訳継続	★ -Xa, -Xcは削除し、-Xgをdefault化
-Klto(F) -flto(C/C++) ★	共通	LTO（リンク時最適化） • ipoとltoをltoに統合 • FortranでのLTO動作	-Kipo (Cのみ) （注）旧仕様は警告し、翻訳継続	-Klto (F) -flto (C/C++ clangモード)

A64FXで仕様変更したオプション（3/7）

FUJITSU

翻訳オプション	言語	意味 変更理由	旧仕様	新仕様
-Kzfill ★	共通	ストア命令の高速化。 ストア領域のプリフェッチに酷似、ただし、データをメモリからロードせず、キャッシュ位置だけ確保 アーキ仕様に基づく変更	-KXFILL （”X”は不定値fillの意味） ※旧仕様であることを警告し、新オプションに変更して翻訳継続	-Kzfill （”z”はzero fillの意味）
-Kilfunc [={loop|procedure}]	共通	組込み関数および演算をインライン展開 各種アプリで効果が見込めたため	-Kilfuncのdefaultは、 -Kilfunc=loop	-Kilfuncのdefaultは、 -Kilfunc=procedure
-Kprefetch_stride [={soft|hard_auto| hard_always}] ★	共通	ループ内でキャッシュラインサイズよりも大きいストライドでアクセスされる配列データに対して、prefetch命令を生成 マイクロアーキを活かす機能の追加のため	-Kprefetch_stride	ハードウェアストライドプリフェッチ機能の利用が可能 -Kprefetch_stride [={soft|hard_auto|hard_always}]
-Kprefetch_strong[_L2]	共通	L1/L2 prefetchをstrong prefetch命令にすることを指示 マイクロアーキに基づき仕様変更	-Kprefetch_nostrong が default	-Kprefetch_strong が default

A64FXで仕様変更したオプション（4/7）

FUJITSU

翻訳オプション	言語	意味 変更理由	旧仕様	新仕様
-Kassume=shortloop	共通	プログラム中の最内ループの回転数が小さいとみなして、最適化制御 アプリ特徴に基づく最適化制御オプションに集約	-Kshortloop （注）旧仕様は警告し、翻訳継続	-Kassume=shortloop
-Kloop_part_simd	共通	ループ中にSIMD化の可/不可部分が混在している場合に分割してSIMD化 分割ではなく、SIMD機能の一部に位置づけ	-Kloop_part_simdは-Kloop_fission機能の一部として動作	-Ksimdが有効な場合に動作
-Kloop_part_parallel	共通	ループ中にスレッド並列化の可/不可部分が混在している場合に分割して並列化 分割でなく、並列機能の一部に位置づけ	-Kloop_part_parallelは-Kloop_fission機能の一部として動作	-Kparallelが有効な場合に動作
-Koptions	F	!options行で始まる行を翻訳指示行として扱う 当初廃止を検討したが、OCLがない-O[1-3]だけ残す方向とした。	!options -O[1-3] !options 一部の翻訳オプション	!options -O[1-3]

A64FXで仕様変更したオプション（5/7）

FUJITSU

翻訳オプション	言語	意味 変更理由	旧仕様	新仕様
-Krdconv[={1|2}]	C/C++	4バイト以下の符号付き整数型の式がオーバーフローしない、4バイト以下の符号無し整数型の式がラップアラウンドしないと仮定した最適化を実施 -Krdconv機能を細分化し、defaultは業界標準仕様とするため。	-Krdconv 4バイトの符号付き変数がオーバーフローしないと仮定した最適化を促進	-Krdconvは-Krdconv=1を誘導。-Krdconvは-O2で有効となったため、-Kfastからの誘導は止め。 -Krdconv=1 4バイト以下の符号付き整数型の式がオーバーフローしないと仮定した最適化を促進 -Krdconv=2 4バイトの符号無し整数型の式がラップアラウンドしないと仮定した最適化を促進
-Kstrict_aliasing ★	C/C++	厳密なaliasing ruleに従ったメモリ領域の重なりを考慮した最適化を指示 業界標準レベルの仕様に変更（オプション名も）	-Kmemalias ※旧仕様は非推奨かつ廃止予定と警告し、-Krestrict_aliasingを推奨と通知。翻訳は継続。	-Kstrict_aliasing
-Kfz	共通	flush-to-zeroモードの使用を指示 アーキ仕様に基づく変更	-Kns (SPARCのNS bit) ※廃止を通知し、翻訳は継続。	-Kfz (ARMのFZ bit)

A64FXで仕様変更したオプション（6/7）

FUJITSU

翻訳オプション	言語	意味 変更理由	旧仕様	新仕様
-Kalign_commons	F	共通ブロックに属する変数に対する記憶域への割付処理において、8バイト整数型、倍精度／4倍精度実数型／複素数型データに対して8バイトの境界調整を行うことを指示 従来機能を分割し、必要な機能のみ業界標準を意識した仕様に変更	-Kdalign (defau l tは-Knodalign) -Kdalignの２つの機能 ①common変数のメンバのalignを合わせる機能 ②SPARCにおいてldd命令を使うかどうかを指示する機能→不要 ※旧仕様は警告し、翻訳継続	Fortran -Kalign_commons (default) C/C++ 廃止
-Nline	C/C++	プロファイラで提供される実行時間のサンプリング機能に必要な追加情報を生成 京/FX100からの使い勝手を継続することを優先	-Nline (-Xa(default), -Xc時） -Nnoline （-Xg時）	-Xgがdefaultになったが、 -Nlineをdefaultとした。
-std= {c++14|gnu++14}	C++	GNU++14でコンパイル C++14のフルサポートに伴う仕様変更	C++14はオプショナル	C++14がdefault

A64FXで仕様変更したオプション（7/7）

FUJITSU

翻訳オプション	言語	意味 変更理由	旧仕様	新仕様
-gdwarf[-4]	共通	DWARFバージョン4のサポート DWARF4サポートに伴い変更	-gdwarf-2 (DWARF version 2) ※旧仕様は廃止を通知し、翻訳継続	-gdwarf[-4] (DWARF version 4)
-Ncheck_std= {03d|03e|03o|03s} -Ncheck_std= |08d|08e|08o|08s}	F	原始プログラムの言語規格検査を行う。 • -vオプションは他社では、他の意味で利用されている。紛らわしため、改名して欲しいという要望あり • Fortran 2008規格のサポート	-v{03d|03e|03o|03s} ※旧仕様であることを警告し、新オプションに変更して翻訳継続	-Ncheck_std= {03d|03e|03o|03s|08d|08e|08o|08s}

A64FXで廃止したオプション（1/2）

FUJITSU

翻訳オプション	言語	廃止理由	動作
-KHPC_ACE[2]	共通	京/FX100のSIMD拡張命令セットの利用オプションであるため	警告し、翻訳継続
-K[NO]FLTLD	共通	京/FX100アーキテクチャ向けオプションのため	警告し、翻訳継続
-K[NO]GREG_APPLI	C/C++	SPARC系アーキテクチャ向けオプションのため	警告し、翻訳継続
-Kadr{44|64}	共通	SPARCアーキテクチャ向けオプションのため	警告し、翻訳継続
-K[no]fed	共通	FX100アーキテクチャ向けオプションのため	警告し、翻訳継続
-Kfuncalign=N	C	京/FX100アーキテクチャ向けオプションのため	警告し、翻訳継続
-Kloop_[no]fission_if	共通	本機能はIF文を含むループの分割だが、性能低下リスクもあり、利用者もほとんどいなかった。新ループ分割で代替えするが、新ループ分割はSIMD化により、分岐がなくなった後で分割するため削除	警告し、翻訳継続
-K[no]nf	共通	FX100アーキテクチャ向けオプションのため	警告し、翻訳継続
-Kopenmp_tls	共通	OpenMPのthreadprivateの実装を独自方式から業界標準のTLSのみに変更したため	廃止を通知し、翻訳継続
-Ksimd_[no]separate_stride	共通	FX100のstride SIMD命令に対するオプションのため	警告し、翻訳継続
-K[no]uxsimd	共通	京/FX10の2SIMD用機能であるため廃止	警告し、翻訳継続
-Kvppocl	F	ベクトル機時代の互換オプションで利用者もいないため	廃止を通知し、翻訳継続

A64FXで廃止したオプション（2/2）

FUJITSU

翻訳オプション	言語	廃止理由	動作
-Nrt_tune_io	C/C++	実行時出力機能として出力していた入出力情報は、富岳では性能解析ツールとして出力	警告し、翻訳継続
-Nstl={500|521|500fast} -Kstl_fast_new	C++	STLport系のSTLを廃止し、libc++をdefaultのSTLに変更したため	警告し、翻訳継続
-fcall_used_g7 -ffixed-g{4|5}	共通	SPARCアーキテクチャ向けオプションのため	警告し、翻訳継続
-fvisibility={default|internal|hidden|protected}	C/C++	（tradモードのみ廃止）Clangモードの同名オプションで代替可能なため	警告し、翻訳継続

京/FX10/FX100 ⇒ A64FX の互換情報 (1/11)

FUJITSU

- Fortran の互換項目 (1/3)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
1	[京/FX10/FX100]	-Kautoオプションのデフォルト化
2	[京/FX10/FX100]	-Ktemparraystackオプションのデフォルト化
3	[京/FX10/FX100]	-Kautoobjstackオプションのデフォルト化
4	[FX100]	-Kltoオプションと同時に-Sまたは-Waオプションを指定した場合のオプションの扱いを変更
5	[京/FX10]	-Ksimd[=level]オプションのlevelの省略値変更
6	[京/FX10/FX100]	GAMMA系の組込み関数の結果
7	[京/FX10/FX100]	G形編集の出力結果
8	[京/FX10/FX100]	停留出力文実行時のIOSTAT指定子の返却値
9	[京/FX10/FX100]	実行時情報出力機能の仕様変更
10	[京/FX10/FX100]	LLVM OpenMPライブラリの導入
11	[京/FX10/FX100]	Fortran言語における非サポート項目
12	[京/FX10/FX100]	既定義バージョンマクロの値を変更
13	[京/FX10/FX100]	OpenMP を使用した場合のオブジェクト互換
14	[京/FX10/FX100]	Fortran規格に準じた翻訳時のエラー検査
15	[京/FX10/FX100]	Fortran規格の新仕様・仕様変更、および実数型と複素数型の種別型パラメタ(拡張仕様)
16	[京/FX10/FX100]	共配列とOpenMP仕様における、翻訳時のエラー検査
17	[京/FX10/FX100]	翻訳時オプション-H有効時に、誤りのあるプログラムに対して実行時にメッセージを出力
18	[京/FX10/FX100]	ファイルサフィックス．F、．f、．FOR、および．forから誘導される言語仕様レベルオプションを-X03から-X08に変更
19	[京/FX10/FX100]	Fortran規格に準じた実行時メッセージ出力
20	[京/FX10/FX100]	-v03d、-v03e、-v03oおよび-v03sオプションを-Ncheck_std=に変更
21	[京/FX10/FX100]	OpenMP API version 4.0仕様のサポートに伴う、マクロおよび名前付き定数の値変更
22	[京/FX10/FX100]	翻訳時オプション-K{openmp_tls|openmp_notls}の廃止

京/FX10/FX100 ⇒ A64FX の互換情報 (2/11)

FUJITSU

- Fortran の互換項目 (2/3)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
23	[京/FX10/FX100]	OpenMP THREADPRIVATE指示文のエラー検査
24	[京/FX10/FX100]	SPARC命令セットアーキテクチャ固有のオプションの削除
25	[京/FX10/FX100]	HPC-ACEおよびHPC-ACE2固有の翻訳時オプションの削除
26	[FX100]	命令セットアーキテクチャに依存する仕様の変更
27	[京/FX10/FX100]	-Kvppoclオプションの廃止
28	[京/FX10/FX100]	OpenMP指示文の翻訳時のエラー検査
29	[京/FX10/FX100]	翻訳指示行“!optoins”と-Koptionsオプションの廃止
30	[京/FX10/FX100]	-Nuse_rodataオプションの廃止と仕様変更
31	[京/FX10/FX100]	-Qオプションの廃止
32	[京/FX10/FX100]	-K{XFILL|NOXFILL}オプションおよび最適化指示子{XFILL|NOXFILL}の変更と最適化情報の出力の変更
33	[京/FX10/FX100]	ENTRY文を含む手続きのトレースバックの仕様変更
34	[京/FX10/FX100]	-K{shortloop=N|noshortloop}オプションおよび最適化指示子{SHORTLOOP(n)|NOSHORTLOOP}の廃止
35	[京/FX10/FX100]	-K{uxsimd|nouxsimd}オプションおよび最適化指示子{UXSIMD|NOUXSIMD}の廃止
36	[京/FX10/FX100]	-Kswpおよび-Kswp_strongオプション仕様の変更
37	[京/FX10/FX100]	翻訳情報(SIMD化)の表示形式の変更
38	[京/FX10/FX100]	翻訳メッセージ(ループ分割)の表示形式の変更
39	[京/FX10/FX100]	翻訳情報(ループ分割)の表示形式の変更
40	[京/FX10/FX100]	翻訳情報(ソフトウェアパイプライニング)の表示形式の変更
41	[京/FX10/FX100]	-Kdalignオプションの廃止
42	[京]	最適化レベルと翻訳時オプション-Kloop_fissionの誘導関係の変更
43	[京/FX10/FX100]	翻訳時オプション-Kloop_part_simdおよび最適化指示子LOOP_PART_SIMDの変更
44	[京/FX10/FX100]	翻訳時オプション-Kloop_part_parallelおよび最適化指示子LOOP_PART_PARALLELの変更

京/FX10/FX100 ⇒ A64FX の互換情報 (3/11)

FUJITSU

- Fortran の互換項目 (3/3)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
45	[京/FX10/FX100]	翻訳時オプション-K{loop_fission_if|loop_nofission_if}の廃止
46	[京/FX10/FX100]	-Kprefetch_strongをデフォルトに変更
47	[京/FX10/FX100]	-Kprefetch_nostrongおよび-Kprefetch_nostrong_L2を有効にするため-Khpctag が必要になる
48	[京]	京固有の翻訳時オプションの削除
49	[FX10]	FX10固有の翻訳時オプションの削除
50	[FX100]	SIMD長の倍数冗長実行によるSIMD化機能のデフォルト化、および最適化指示子simd_redundant_vlの削除
51	[京/FX10/FX100]	実行時メッセージの修正
52	[京/FX10/FX100]	認識できないオプションが指定された場合の動作変更
53	[京/FX10/FX100]	翻訳時オプション-Kfastから誘導されるオプションの変更
54	[京/FX10/FX100]	書式付き流れ探査入出力および書式付き停留入出力の実行結果変更
55	[京/FX10/FX100]	並列処理時に出力される実行時メッセージ
56	[FX10/FX100]	LLVM OpenMPライブラリにおける型変換オプション
57	[京/FX10/FX100]	LLVM OpenMPライブラリにおけるIOバッファ並列転送
58	[京/FX10/FX100]	浮動小数点例外捕捉時に出力される実行時メッセージ
59	[京/FX10/FX100]	jwd8205o-iの和文メッセージ変更
60	[京/FX10/FX100]	組込み手続きFRACTION、RRSPACING、SPACINGにおけるFortran2008仕様への変更

京/FX10/FX100 ⇒ A64FX の互換情報 (4/11)

FUJITSU

- C の互換項目 (1/3)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
1	[京/FX10/FX100]	-Klibオプションのデフォルト化
2	[京/FX10/FX100]	-Kipoオプションの廃止
3	[FX100]	-Kltoオプションと同時に-Sまたは-Waオプションを指定した場合のオプションの扱いを変更
4	[京/FX10]	-Ksimd[=level]オプションのlevelの省略値変更
5	[京/FX10/FX100]	実行時情報出力機能の仕様変更
6	[京/FX10/FX100]	LLVM OpenMPライブラリの導入
7	[京/FX10/FX100]	-gdwarf-2オプションの廃止、-gdwarfおよび-gdwarf-4オプションの追加
8	[京/FX10/FX100]	翻訳時オプション-noansiの廃止
9	[京/FX10/FX100]	翻訳時オプション-f{signed-char|unsigned-char}の省略値変更
10	[京/FX10/FX100]	既定義バージョンマクロの値を変更
11	[京/FX10/FX100]	翻訳時オプション-Xの廃止(言語仕様のモードはGNU互換モードのみ)
12	[京/FX10/FX100]	GNU C互換バージョンの変更
13	[京/FX10/FX100]	翻訳時オプション-stdの省略値変更
14	[京/FX10/FX100]	マクロ__STRICT_ANSI__、linux、unix、および__STDC_VERSION__の定義変更
15	[京/FX10/FX100]	翻訳時オプション-ansiまたは-std有効時の言語仕様レベルを変更
16	[京/FX10/FX100]	翻訳時オプション-Dnameと-Unameを同時に指定した場合の動作変更
17	[京/FX10/FX100]	翻訳時オプション-P指定時の動作変更
18	[京/FX10/FX100]	翻訳時オプション-N{line|noline}の省略値変更
19	[京/FX10/FX100]	OpenMP を使用した場合のオブジェクト互換
20	[京/FX10/FX100]	翻訳時オプション-K{openmp_tls|openmp_notls}の廃止
21	[京/FX10/FX100]	SPARC命令セットアーキテクチャ固有のオプションの削除
22	[京/FX10/FX100]	SPARC命令セットアーキテクチャ固有のGNU C互換オプションの廃止

京/FX10/FX100 ⇒ A64FX の互換情報 (5/11)

FUJITSU

- C の互換項目 (2/3)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
23	[京/FX10/FX100]	HPC-ACEおよびHPC-ACE2固有の翻訳時オプションの削除
24	[FX100]	命令セットアーキテクチャに依存する仕様の変更
25	[京/FX10/FX100]	-Kstrict_aliasing関連オプションの仕様変更
26	[京/FX10/FX100]	-Kfuncalign=Nオプションの廃止
27	[京/FX10/FX100]	-Nuse_rodataオプションの廃止と仕様変更
28	[京/FX10/FX100]	翻訳オプション-O2、-O3オプションから誘導されるオプションの変更
29	[京/FX10/FX100]	-K{XFILL|NOXFILL}オプションおよび最適化指示子{XFILL|NOXFILL}の変更と最適化情報の出力の変更
30	[京/FX10/FX100]	-K{shortloop=N|noshortloop}オプションおよび最適化指示子{SHORTLOOP(n)|NOSHORTLOOP}の廃止
31	[京/FX10/FX100]	-K{uxsimd|nouxsimd}オプションおよび最適化指示子{UXSIMD|NOUXSIMD}の廃止
32	[京/FX10/FX100]	-Kswpおよび-Kswp_strongオプション仕様の変更
33	[京/FX10/FX100]	翻訳情報(SIMD化)の表示形式の変更
34	[京/FX10/FX100]	翻訳メッセージ(ループ分割)の表示形式の変更
35	[京/FX10/FX100]	翻訳情報(ループ分割)の表示形式の変更
36	[京/FX10/FX100]	翻訳情報(ソフトウェアパイプライニング)の表示形式の変更
37	[京/FX10/FX100]	-Kdalignオプションの廃止
38	[京/FX10/FX100]	ビルトイン関数__sync_fetch_and_nandおよび__sync_nand_and_fetchの仕様変更
39	[京]	最適化レベルと翻訳時オプション-Kloop_fissionの誘導関係の変更
40	[京/FX10/FX100]	翻訳時オプション-Kloop_part_simdおよび最適化指示子LOOP_PART_SIMDの変更
41	[京/FX10/FX100]	翻訳時オプション-Kloop_part_parallelおよび最適化指示子LOOP_PART_PARALLELの変更
42	[京/FX10/FX100]	翻訳時オプション-K{loop_fission_if|loop_nofission_if}の廃止
43	[京/FX10/FX100]	-Kprefetch_strongをデフォルトに変更
44	[京/FX10/FX100]	-Kprefetch_nostrongおよび-Kprefetch_nostrong_L2を有効にするため-Khpctag が必要になる

京/FX10/FX100 ⇒ A64FX の互換情報 (6/11)

FUJITSU

- C の互換項目 (3/3)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
45	[京]	京固有の翻訳時オプションの削除
46	[FX10]	FX10固有の翻訳時オプションの削除
47	[FX100]	SIMD長の倍数冗長実行によるSIMD化機能のデフォルト化、および最適化指示子simd_redundant_vlの削除
48	[京/FX10/FX100]	実行時メッセージの修正
49	[京/FX10/FX100]	認識できないオプションが指定された場合の動作変更
50	[京/FX10/FX100]	翻訳時オプション-Kfastから誘導されるオプションの変更
51	[京/FX10/FX100]	並列処理時に出力される実行時メッセージ
52	[京/FX10/FX100]	浮動小数点例外捕捉時に出力される実行時メッセージ
53	[京/FX10/FX100]	jwd8205o-iの和文メッセージ変更

京/FX10/FX100 ⇒ A64FX の互換情報 (7/11)

FUJITSU

- C++ の互換項目 (1/3)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
1	[京/FX10/FX100]	-Klibオプションのデフォルト化
2	[FX100]	-Kltoオプションと同時に-Sまたは-Waオプションを指定した場合のオプションの扱いを変更
3	[京/FX10]	-Ksimd[=level]オプションのlevelの省略値変更
4	[京/FX10/FX100]	実行時情報出力機能の仕様変更
5	[京/FX10/FX100]	LLVM OpenMPライブラリの導入
6	[京/FX10/FX100]	-gdwarf-2オプションの廃止、-gdwarfおよび-gdwarf-4オプションの追加
7	[京/FX10/FX100]	C++03仕様またはC++11仕様における標準テンプレートライブラリ(STL)の変更
8	[京/FX10/FX100]	翻訳時オプション-stdlibおよび-Nstlの廃止
9	[京/FX10/FX100]	翻訳時オプション-K{stl_fast_new|nostl_fast_new}の廃止
10	[京/FX10/FX100]	翻訳時オプション-f{signed-char|unsigned-char}の省略値変更
11	[京/FX10/FX100]	既定義バージョンマクロの値を変更
12	[京/FX10/FX100]	翻訳時オプション-Xの廃止(言語仕様のモードはGNU互換モードのみ)
13	[京/FX10/FX100]	GNU C++互換バージョンの変更
14	[京/FX10/FX100]	翻訳時オプション-stdの省略値変更
15	[京/FX10/FX100]	マクロ__STRICT_ANSI__、linux、unix、および__cplusplusの定義変更
16	[京/FX10/FX100]	翻訳時オプション-Dnameと-Unameを同時に指定した場合の動作変更
17	[京/FX10/FX100]	翻訳時オプション-P指定時の動作変更
18	[京/FX10/FX100]	翻訳時オプション-N{line|noline}の省略値変更
19	[京/FX10/FX100]	OpenMP を使用した場合のオブジェクト互換
20	[京/FX10/FX100]	翻訳時オプション-K{openmp_tls|openmp_notls}の廃止
21	[京/FX10/FX100]	SPARC命令セットアーキテクチャ固有のオプションの削除
22	[京/FX10/FX100]	SPARC命令セットアーキテクチャ固有のGNU C++互換オプションの廃止

京/FX10/FX100 ⇒ A64FX の互換情報 (8/11)

FUJITSU

- C++ の互換項目 (2/3)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
23	[京/FX10/FX100]	HPC-ACEおよびHPC-ACE2固有の翻訳時オプションの削除
24	[FX100]	命令セットアーキテクチャに依存する仕様の変更
25	[京/FX10/FX100]	-Kstrict_aliasing関連オプションの仕様変更
26	[京/FX10/FX100]	-Nuse_rodataオプションの廃止と仕様変更
27	[京/FX10/FX100]	翻訳オプション-O2、-O3オプションから誘導されるオプションの変更
28	[京/FX10/FX100]	-K{XFILL|NOXFILL}オプションおよび最適化指示子{XFILL|NOXFILL}の変更と最適化情報の出力の変更
29	[京/FX10/FX100]	-K{shortloop=N|noshortloop}オプションおよび最適化指示子{SHORTLOOP(n)|NOSHORTLOOP}の廃止
30	[京/FX10/FX100]	-K{uxsimd|nouxsimd}オプションおよび最適化指示子{UXSIMD|NOUXSIMD}の廃止
31	[京/FX10/FX100]	-Kswpおよび-Kswp_strongオプション仕様の変更
32	[京/FX10/FX100]	翻訳情報(SIMD化)の表示形式の変更
33	[京/FX10/FX100]	翻訳メッセージ(ループ分割)の表示形式の変更
34	[京/FX10/FX100]	翻訳情報(ループ分割)の表示形式の変更
35	[京/FX10/FX100]	翻訳情報(ソフトウェアパイプライニング)の表示形式の変更
36	[京/FX10/FX100]	-Kdalignオプションの廃止
37	[京/FX10/FX100]	ビルトイン関数__sync_fetch_and_nandおよび__sync_nand_and_fetchの仕様変更
38	[京]	最適化レベルと翻訳時オプション-Kloop_fissionの誘導関係の変更
39	[京/FX10/FX100]	翻訳時オプション-Kloop_part_simdおよび最適化指示子LOOP_PART_SIMDの変更
40	[京/FX10/FX100]	翻訳時オプション-Kloop_part_parallelおよび最適化指示子LOOP_PART_PARALLELの変更
41	[京/FX10/FX100]	翻訳時オプション-K{loop_fission_if|loop_nofission_if}の廃止
42	[京/FX10/FX100]	-Kprefetch_strongをデフォルトに変更
43	[京/FX10/FX100]	-Kprefetch_nostrongおよび-Kprefetch_nostrong_L2を有効にするため-Khpctag が必要になる
44	[京]	京固有の翻訳時オプションの削除

京/FX10/FX100 ⇒ A64FX の互換情報 (9/11)

FUJITSU

- C++ の互換項目 (3/3)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
45	[FX10]	FX10固有の翻訳時オプションの削除
46	[FX100]	SIMD長の倍数冗長実行によるSIMD化機能のデフォルト化、および最適化指示子simd_redundant_vlの削除
47	[京/FX10/FX100]	実行時メッセージの修正
48	[京/FX10/FX100]	認識できないオプションが指定された場合の動作変更
49	[京/FX10/FX100]	翻訳時オプション-Kfastから誘導されるオプションの変更
50	[京/FX10/FX100]	並列処理時に出力される実行時メッセージ
51	[京/FX10/FX100]	浮動小数点例外捕捉時に出力される実行時メッセージ
52	[京/FX10/FX100]	jwd8205o-iの和文メッセージ変更

京/FX10/FX100 ⇒ A64FX の互換情報 (10/11)

FUJITSU

- MPI の互換項目 (1/2)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
1	[京/FX10/FX100]	Hasty Rendezvous通信機能を廃止
2	[京/FX10/FX100]	MCAパラメーターdpm_ple_socket_timeoutを廃止
3	[京/FX10/FX100]	拡張RDMAインターフェースを廃止
4	[京/FX10/FX100]	ランク問合せインターフェースを変更
5	[京/FX10/FX100]	mpiexecコマンドのオプションに関するエラーメッセージの出力を変更
6	[京/FX10/FX100]	mpiexecコマンドをオプション指定なしで実行した場合、エラーメッセージを出力するよう仕様変更
7	[京/FX10/FX100]	メモリ不足のエラーメッセージを変更
8	[京/FX10/FX100]	エラーメッセージの変更および削除
9	[京/FX10/FX100]	警告メッセージの変更
10	[京/FX10/FX100]	シグナルSIGCHLDの削除
11	[京/FX10/FX100]	Fortranのプロファイリングインターフェース利用で、C言語のMPI関数はフックされなくなる
12	[京/FX10/FX100]	MCAパラメーターorte_abort_print_stackをopal_abort_print_stackに変更
13	[京/FX10/FX100]	MPIプログラム内からのプロセス生成時の制約事項を変更
14	[FX100]	MCAパラメーターcommon_tofu_packet_mtuの省略値を1792から1920に変更
15	[京/FX10/FX100]	エラーメッセージ[mpi::common-tofu::tofu-signal-*]を変更
16	[FX10/FX100]	mpiexecコマンドの-nompiオプションを削除
17	[FX10/FX100]	MCAパラメーター dpm_ple_no_establish_connection のパラメーター名を変更
18	[京/FX10/FX100]	集団通信で使用するリダクション演算のアルゴリズムを変更
19	[京/FX10/FX100]	MCAパラメーターcoll_tbi_use_on_max_minの省略値を変更
20	[京/FX10/FX100]	MPI_REDUCEルーチンおよびMPI_ALLREDUCEルーチンにおけるバリア通信機能の適用条件を変更
21	[京/FX10/FX100]	MCAパラメーターcoll_tuned_use_6d_algorithmとcoll_tuned_scatterv_use_linear_syncを廃止
22	[FX100]	MCAパラメーター opal_progress_thread_mode に不正な値(0-3以外)を指定したときの動作の変更

京/FX10/FX100 ⇒ A64FX の互換情報 (11/11)

FUJITSU

- MPI の互換項目 (2/2)
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
23	[京]	通信タイムアウト設定機能に関するMCAパラメーター名とエラーメッセージを変更
24	[FX10/FX100]	MCAパラメーターmpi_deadlock_timeoutおよびmpi_deadlock_timeout_delayの廃止

- 数学ライブラリ の互換項目
  - 詳細は、システム管理者にお問い合わせください。

No.	対象マシン	項目
1	[FX100]	BLASのヘッダファイルcblas.hを最新版に更新

FUJITSU

MPI の拡張関数とデータ型

- MPI の拡張関数とデータ型

MPI の拡張関数とデータ型

FUJITSU

- 拡張関数

関数名	A64FX の動作	FX700 の動作
FJMPI_Collection_*	区間指定MPI統計情報	情報収集をせず、結果が表示されない
FJMPI_Progress_*	アシスタントコアを用いた通信の促進	通信と計算がオーバーラップしない（呼ばないのと同じ）
FJMPI_Prequest_*	拡張持続的通信要求	MPI_Send_init などのラッパとして動く
FJMPI_Topology_*	Tofu座標の取得	ランク番号に基づきTofuの1次元ジョブを模擬した値を返す
FJMPI_Mswk_*	マスターワーカ機能	A64FX と同じ
MPIX_*_init	持続的集団通信要求の先取り実装	A64FX と同じ

- データ型
  - 半精度データ型が追加されます。

エンディアン

- エンディアン

エンディアン

FUJITSU

- エンディアン

ノード種別	京	A64FX
計算ノード	ビッグエンディアン	リトルエンディアン
ログインノード（IAサーバ）	リトルエンディアン	リトルエンディアン

- 実行時の自動変換（富士通Fortran）
  - 富士通Fortranの実行時オプション（-Wl,-Tu_num）でビッグエンディアンのデータ入出力が可能です。
  - 指定できる装置番号は一つのみです。装置番号を省略した場合、書式なしファイルと接続しているすべての装置番号が対象となります。

【指定例：実行時オプション（装置番号=10）】

```
./a.out  -Wl,-T10
```

【指定例：環境変数（装置番号=10）】

```
export FORT90L="-Wl,-T10"
./a.out
```

FUJITSU

コンパイラによる高速化

- プリフェッチ
- SIMD化
- ソフトウェアパイプライニング
- ループ最適化と命令スケジューリング
- ループに対する最適化
- 自動並列化

FUJITSU

プリフェッチ

- プリフェッチについて
- ハードウェアプリフェッチ
- ソフトウェアプリフェッチ
- プリフェッチの協調

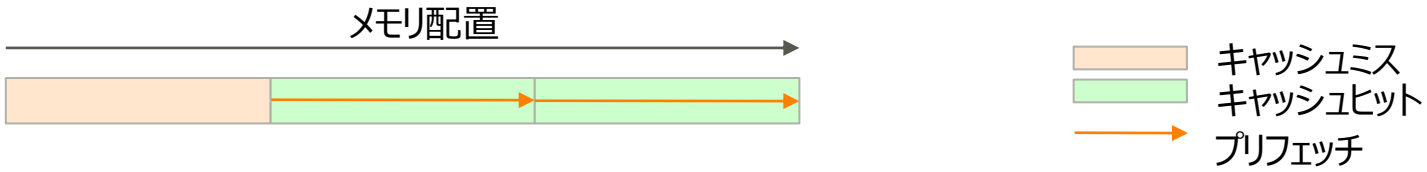
FUJITSU

プリフェッチについて

- プリフェッチとは
- プリフェッチについて
- 【参考】レイテンシの隠蔽とは

プリフェッチとは

プリフェッチとは、実行される命令によってデータが必要になる前に、キャッシュにデータをロードしておくことで、パフォーマンスを向上させる仕組みです。

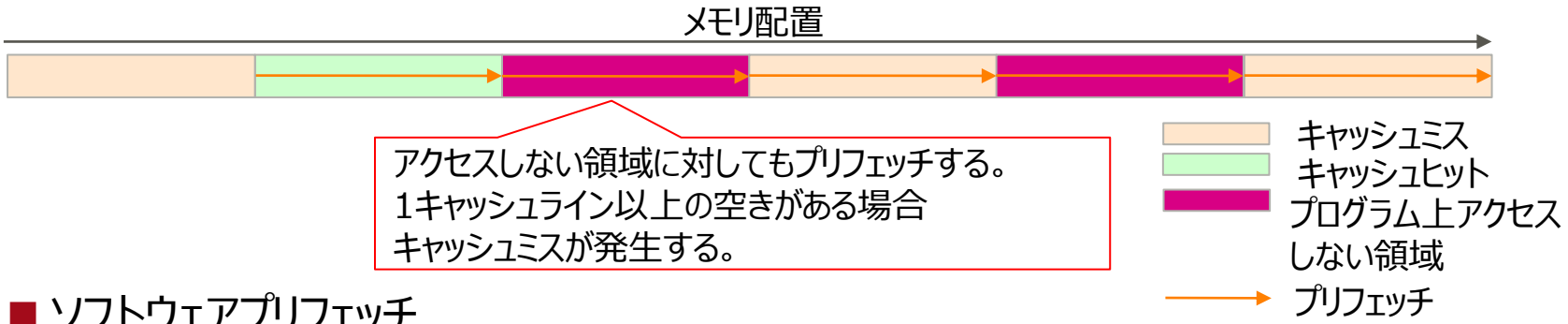


プリフェッチにはハードウェアプリフェッチとソフトウェアプリフェッチがあります。

■ ハードウェアプリフェッチ

プログラムのメモリアクセスの規則性からハードウェアがデータアクセスを予測しプリフェッチします。

プログラム上アクセスしない領域がある場合にもプリフェッチしてしまうため、そのようなプログラムではキャッシュ効率が非常に悪くなります。その場合はソフトウェアプリフェッチを利用します。



■ ソフトウェアプリフェッチ

ソフトウェア（コンパイラ）がプログラムを解析し、prefetch命令を生成することでプリフェッチします。

プリフェッチについて (1/2)

FUJITSU

- A64FXでは、ハードウェアとソフトウェアのプリフェッチとして、以下の新機能をサポートしている。
  - ハードウェアプリフェッチ(HWPF)
    - ハードウェアプリフェッチ距離設定機能
    - ハードウェアストライドプリフェッチ機能
  - ソフトウェアプリフェッチ(SWPF)
    - プリフェッチ距離の自動調整
    - SVE Gatherプリフェッチ命令のサポート

- これらのプリフェッチを利用することにより、データアクセスのレイテンシを隠蔽しアプリの高速実行を可能としている。（レイテンシ隠蔽）

プリフェッチについて (2/2)

FUJITSU

- プリフェッチの距離について

  ハードウェアプリフェッチ・ソフトウェアプリフェッチでは、以下のライン先のデータに対しプリフェッチを行う。

	ハードウェアプリフェッチ		ソフトウェアプリフェッチ	
	L1プリフェッチ	L2プリフェッチ	L1プリフェッチ	L2プリフェッチ
FX100	2ライン	最大16ライン	3ライン	15ライン
A64FX	**最大6ライン**	**最大40ライン**	**自動**	**自動**

ハードウェアプリフェッチの距離はユーザの設定も可能になる

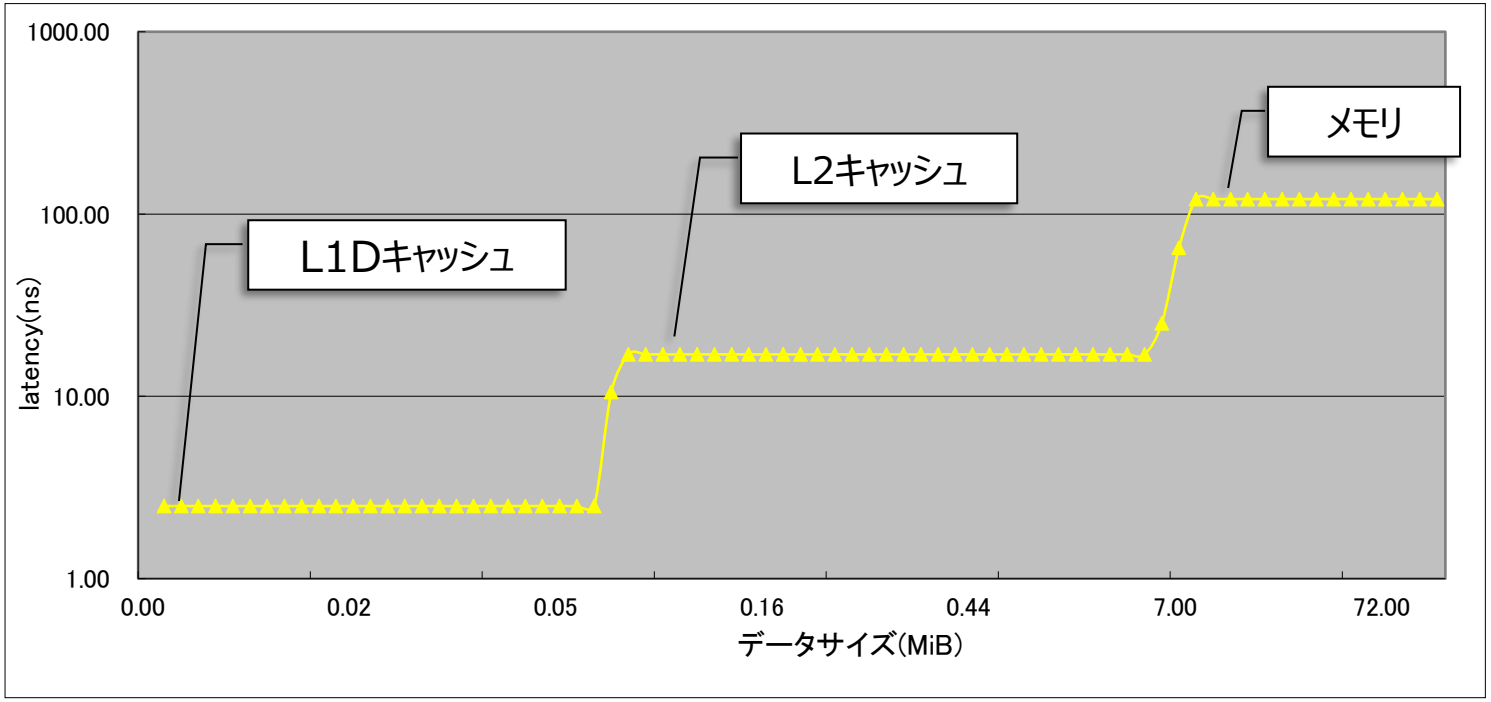
ソフトウェアプリフェッチは距離の自動調整がある

- ループブロッキングや外側ループアンローリングのチューニング実施時、内側ループの繰返し回数が少ないとハードウェアプリフェッチが動作しなくなる場合があるので注意が必要です。

【参考】レイテンシの隠蔽とは

- レイテンシの隠蔽とは、データアクセスのレイテンシ（データの転送要求をしてから戻ってくるまでの時間）をプリフェッチで隠蔽することを意味する。データアクセスは,L1Dキャッシュ、L2キャッシュ、メモリの 3 種類がある。そのうちL2キャッシュおよびメモリのレイテンシ隠蔽がプリフェッチの対象となる。

- LMbench に よるデータアクセスレイテンシの測定結果 （整数アクセス時）



ハードウェアプリフェッチ

- ハードウェアプリフェッチの動作
- ハードウェアプリフェッチ距離設定機能
- ハードウェアストライドプリフェッチ機能

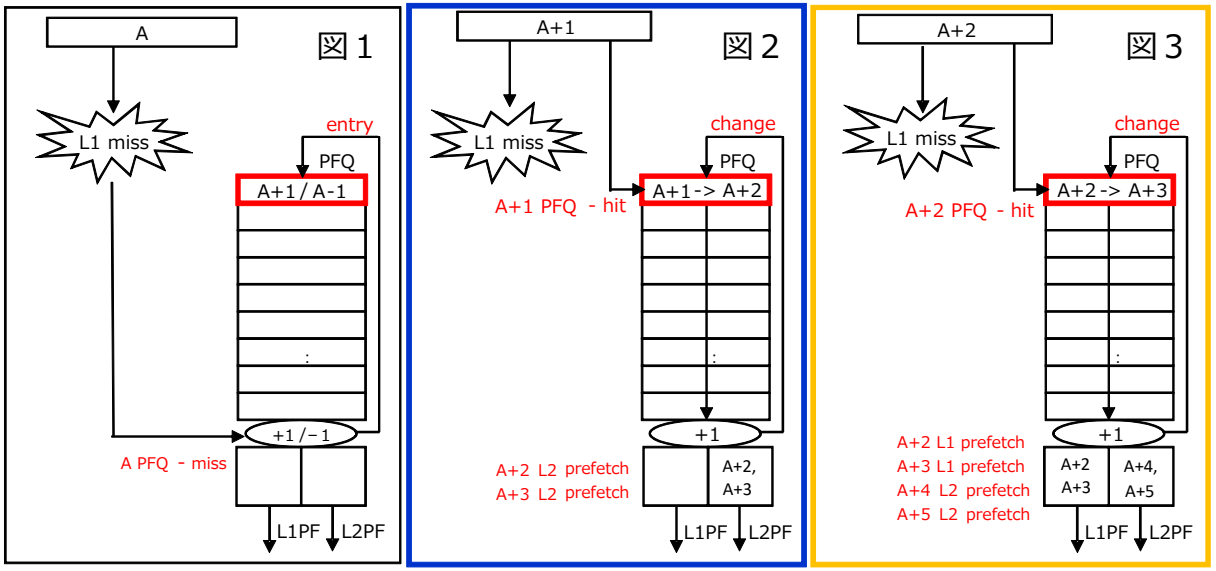
ハードウェアプリフェッチの動作

- ハードウェアプリフェッチ動作条件
  - L1ミスを起因とする
  - L1ミスがキャッシュライン単位で連続に進む
- ハードウェアプリフェッチの動作アルゴリズム

1. キャッシュラインA のミスが発生すると、ラインA+1/A-1が、PFQ(プリフェッチキュー)と呼ばれる16エントリあるFIFO(キュー)に新規登録される。(図１)
2. 後続アクセスがラインA+1にアクセスすると、PFQに登録されているA+1にヒットした時に昇順ストリームアクセスと判断して、以降は昇順方向にHWPFを開始する。また、PFQのA+1はA+2に更新される。(図２)
3. 上記のストリームアクセス検出後は、L2プリフェッチは1度に2ライン分のデータをプリフェッチすることで40ライン先まで拡張する。A64FXではL1プリフェッチもL2プリフェッチ同様に1度に2ライン分のデータをプリフェッチすることで6ライン先まで拡張する。(図３)

A64FXで機能拡張
(FX100では距離固定/1ライン分)

■ プリフェッチの概略



注)表中の数字はラインで示している。バイト表記では１ライン＝２５６バイトで読み替えること。

■ PFQ とプリフェッチアドレスの対応

PFQ-hit	L2HWPF-ADRS	L1PHWF-ADRS
A+ 1	A+ 2, A+ 3	
A+ 2	A+ 4, A+ 5	A+ 2, A+ 3
A+ 3	A+ 6, A+ 7	A+ 4, A+ 5
A+ 4	A+ 8, A+ 9	A+ 6, A+ 7
A+ 5	A+10, A+11	A+ 8, A+ 9
A+ 6	A+12, A+13	A+10
A+ 7	A+14, A+15	A+11
A+ 8	A+16, A+17	A+12
A+ 9	A+18, A+19	A+13
A+10	A+20	A+14
A+11	A+21	A+15
A+12	A+22	A+16
A+13	A+23	A+17
A+14	A+24	A+18
A+15	A+25	A+19

※L2PF=10ライン先、L1PF=4ライン先の場合

ハードウェアプリフェッチ距離設定機能

FUJITSU

- ハードウェアプリフェッチ距離設定用コマンド(hwpfctl)の提供

項目	内容
書式	hwpfctl [--disableL1] [--disableL2] [--distL1 lines_l1] [--distL2 lines_l2] [--weakL1] [--weakL2] [--verbose] command {arguments ...} hwpfctl --default [--verbose] command {arguments ...} hwpfctl --reset [--verbose] hwpfctl --help
説明	hwpfctlコマンドは、A64FXに搭載されているハードウェアプリフェッチ(stream detect mode)の振る舞いを変更する。 変更対象となるCPUコアは、プロセスアフィニティに準ずる。
オプション	--disableL1 --disableL2 L1/L2キャッシュに対するハードウェアプリフェッチを無効とする。省略時はハードウェアプリフェッチが有効である。 --distL1=lines_l1 --distL2=lines_l2 L1/L2キャッシュに対してプリフェッチするキャッシュラインを、キャッシュミスが発生したキャッシュラインを起点とするキャッシュラインの数で指定する。lines_l1にはL1キャッシュに対するプリフェッチ対象ラインを1から15までの値を指定できる。また、lines_l2にはL2キャッシュに対するプリフェッチ対象ラインを1から60までの値を指定できる。ただし、lines_l2に指定した値は4の倍数に切り上げられてシステムレジスタに書き込まれる。0を指定した場合はCPUのデフォルト値で動作する。省略時ならびに無効な値が指定された場合は0が指定されたものとみなす。 --weakL1 --weakL2 L1/L2キャッシュに対するプリフェッチ要求の優先度をweakとすることを指示する。省略時はstrongである。 --default デフォルト設定でcommandを起動する。--verbose以外のオプションは無視する。 --reset システムレジスタの値を初期化する。--verbose以外のオプションは無視する。 --verbose システムレジスタの変更前後の値を出力する。 --help 使用方法を表示する。

- ハードウェアプリフェッチ距離設定用コマンド(hwpfctl)の使用例

```
hwpfctl  --distL1=6  --distL2=40  a.out
```

ハードウェアストライドプリフェッチ機能

FUJITSU

- A64FXでは、ハードウェアストライドプリフェッチ機能 がサポートされる。
  - -Kprefetch_stride オプションの拡張により実装される。

-K{prefetch_stride[={soft|hard_auto|hard_always}]|prefetch_nostride}
ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対して、プリフェッチを実施するか否かを指示します。
プリフェッチするアドレスが翻訳時に確定しないループを含みます。
-Kprefetch_strideオプションが指定された場合、実施します。省略時は、-Kprefetch_nostrideオプションが適用されます。
-Kprefetch_strideオプションの={soft|hard_auto|hard_always}が省略された場合、
-Kprefetch_stride=softオプションが適用されます。
-Kprefetch_stride=softオプションおよび-Kprefetch_nostrideオプションは、-O1オプション以上が有効な場合に意味があります。
-Kprefetch_stride=hard_autoオプションおよび-Kprefetch_stride=hard_alwaysオプションは、-Khpctagオプションおよび-O1オプション以上が有効な場合に意味があります。

-Kprefetch_stride=soft
prefetch命令を生成して、プリフェッチを実施します。

-Kprefetch_stride=hard_auto
ハードウェアストライドプリフェッチャーを利用して、プリフェッチを実施します。
本オプションを指定した場合、キャッシュ上にないデータのみプリフェッチするようハードウェアストライドプリフェッチャーを設定します。

-Kprefetch_stride=hard_always
ハードウェアストライドプリフェッチャーを利用して、プリフェッチを実施します。
本オプションを指定した場合、-Kprefetch_stride=hard_autoオプションと異なり、常にプリフェッチを行うようハードウェアストライドプリフェッチャーを設定します。

FUJITSU

ソフトウェアプリフェッチ

- プリフェッチ距離の自動調整について
- 【参考】A64FXで提供されるプリフェッチ命令
- プリフェッチのオプション
- プリフェッチの最適化指示子(OCL)

プリフェッチ距離の自動調整について

FUJITSU

- 適切なプリフェッチ距離
  - 一般的にプリフェッチ距離はレイテンシが隠蔽できる程度に長ければよい。
  - しかし、必要以上に長くすると、以下の弊害がある。
    - ループの始めの方の参照領域（データ）がプリフェッチされずキャッシュミスが増える
    - キャッシュ上に同時に置かなければならないライン数が増える
  - 適切なプリフェッチ距離は、ループ1イタレーションあたりの実行時間とキャッシュ使用量によりループ毎に異なる。これまでのプリフェッチ距離は、ユーザの指定がない場合は長めの固定値を使用していたため、上記の弊害が発生することがあった。

A64FXでは、ユーザの指定がない場合のプリフェッチ距離は以下を考慮し、ループ毎にヒューリスティックに適切な値をコンパイラが自動で選ぶ。

- ストリーム数
- ループ長

※ プリフェッチ距離の自動調整機能は、プリフェッチ距離を明に指定していない場合に動作する

【参考】A64FXで提供されるプリフェッチ命令

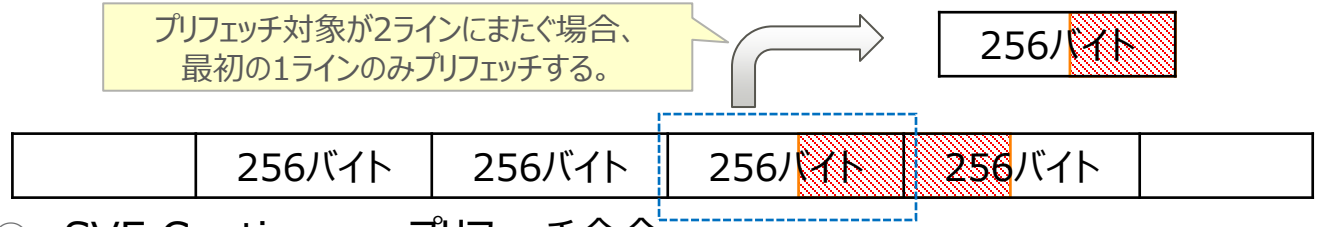
FUJITSU

- A64FXで提供されているプリフェッチ命令

①ARMv8プリフェッチ命令と、
②SVE Contiguousプリフェッチ命令は、
連続アクセスの際に使用される命令

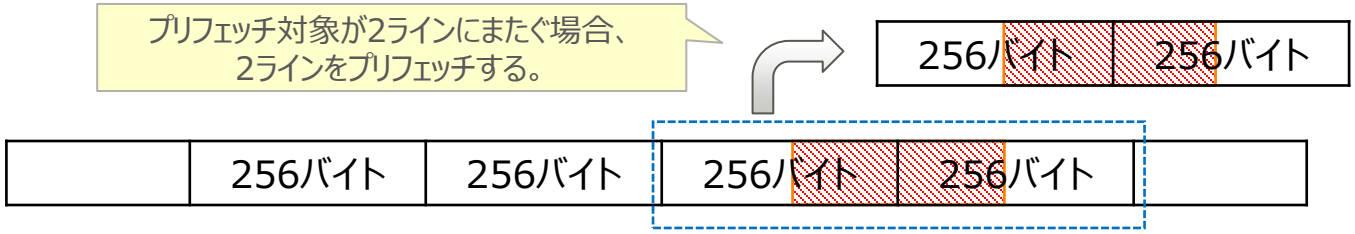
① ARMv8 プリフェッチ命令

連続ロード/ストア命令に対応するプリフェッチ命令(ラインをまたぐ際の考慮なし)



② SVE Contiguous プリフェッチ命令

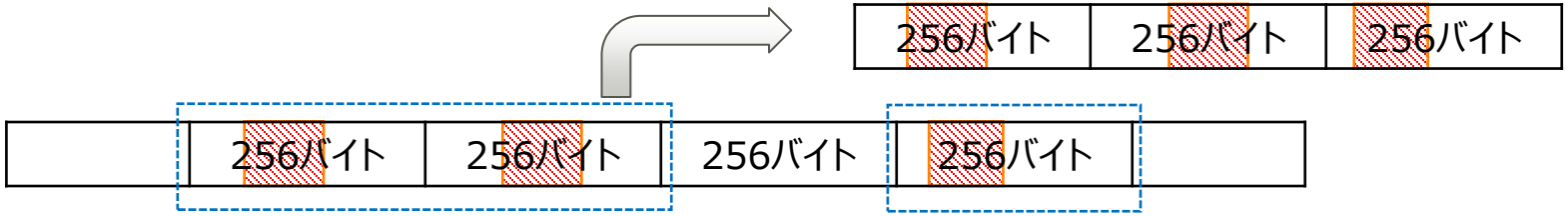
連続ロード/ストア命令に対応するプリフェッチ命令(ラインをまたぐ際の考慮あり)



③ SVE Gather プリフェッチ命令

離散アクセス命令(Gather/Scatter)に対応するプリフェッチ命令

③SVE Gatherプリフェッチ命令は、
インダイレクトアクセスの際に使用される命令



プリフェッチのオプション（1/3）

FUJITSU

オプション形式	機能概要
-Knoprefetch	prefetch命令を使用したオブジェクトを生成しないことを指示します。
-Kprefetch_cache_level = { 1 | 2 | all }	どのキャッシュレベルにデータをプリフェッチするかを指示します。
-K{ prefetch_sequential[=kind] | prefetch_nosequential } kind: { auto | soft }	ループ内で使用される連続的にアクセスされる配列データに対して、prefetch命令を使用したオブジェクトを生成するかどうかを指示します。
-K{prefetch_stride [={soft|hard_auto|hard_always}] |prefetch_nostride}	ループ内で使用されるキャッシュのラインサイズよりも大きなストライドでアクセスされる配列データに対して、プリフェッチを実施するか否かを指示します。プリフェッチするアドレスが翻訳時に確定しないループを含みます。-Kprefetch_strideオプションが指定された場合、実施します。
-K{ prefetch_indirect | prefetch_noindirect }	ループ内で使用される間接的にアクセス(リストアクセス)される配列データに対して、prefetch命令を使用したオブジェクトを生成するかどうかを指示します。(-O1オプション以上が有効な場合に意味があります)
-K{ prefetch_conditional | prefetch_noconditional }	IF構文やCASE構文に含まれるブロックの中で使用される配列データに対してprefetch命令を生成するかどうかを指示します。
-K{ prefetch_infer | prefetch_noinfer }	プリフェッチする距離が不明な場合でも連続アクセスのprefetch命令を生成するかどうかを指示します。

プリフェッチのオプション（2/3）

FUJITSU

オプション形式	機能概要
-K{ prefetch_strong | prefetch_nostrong }	1次キャッシュに対するprefetch命令を生成する際、strong prefetch命令を生成するかどうかを指示します。
-K{ prefetch_strong_L2 | prefetch_nostrong_L2 }	2次キャッシュに対するprefetch命令を生成する際、strong prefetch命令を生成するかどうかを指示します。
-Kprefetch_iteration=N 1 ≦N ≦ 10000	prefetch命令を生成する際、ループのN回転後に参照または定義されるデータを対象とすることを指示します。SIMD化が適用されるループの場合、NにはSIMD化された後のループの繰返し数を指定してください。
-Kprefetch_iteration_L2=N 1 ≦N ≦ 10000	prefetch命令を生成する際、ループのN回転後に参照または定義されるデータを対象とすることを指示します。SIMD化が適用されるループの場合、NにはSIMD化された後のループの繰返し数を指定してください。
-Kprefetch_line=N 1 ≦N ≦ 100	prefetch命令を生成する際、Nキャッシュライン先に該当するデータをプリフェッチの対象とすることを指示します。
-Kprefetch_line_L2=N 1 ≦N ≦ 100	prefetch命令を生成する際、Nキャッシュライン先に該当するデータをプリフェッチの対象とすることを指示します。

プリフェッチのオプション（3/3）

FUJITSU

オプション	実用例
-Kprefetch_indirect	-Kprefetch_indirectは、下記プログラムのa(m(i))のような添字に配列が書かれたリストアクセスに対して、以下の最適化制御行のようなprefetch命令を出力したい場合に指定します。ただし、ループのキャッシュ効率、分岐の有無、または添字の複雑さによって、実行性能が低下する可能性があります。 do i=1,N /* a(m(i+α)) に対して、L2 prefetch命令を出力 */ /* a(m(i+β)) に対して、L1 prefetch命令を出力 */ b(i) = b(i) + a(m(i)) enddo
-Kprefetch_stride	-Kprefetch_strideは、下記プログラムのように、ループの増分値が大きく、現回転でアクセスする配列要素と次回転でアクセスする配列要素が同一キャッシュラインに載らないようなストライドアクセスの場合に、以下の最適化制御行のようなprefetch命令を出力したい場合に指定します。ただし、ループのキャッシュ効率、分岐の有無によって、実行性能が低下する可能性があります。 do i=1,N,100 /* a(i+100*a) に対して、L2 prefetch命令を出力 */ /* a(i+100*β) に対して、L1 prefetch命令を出力 */ /* b(i+100*a) に対して、L2 prefetch命令を出力 */ /* b(i+100*β) に対して、L1 prefetch命令を出力 */ b(i) = a(i) + 1.0 enddo

プリフェッチの最適化指示子(OCL) (1/2)

FUJITSU

最適化指示子	意味	指定可能な最適化制御行			
		プログラム単位	DOループ単位	文単位	配列代入文単位
PREFETCH	コンパイラの自動prefetch機能を有効にします。	○	○	×	×
NOPREFETCH	コンパイラの自動prefetch機能を無効にします。	○	○	×	×
PREFETCH_CACHE_LEVEL (c-level)	キャッシュレベルc-level にデータをプリフェッチすることを指示します。 c-level は1、2 またはall です。	○	○	×	○
PREFETCH_INFER	プリフェッチの距離が不明な場合でも連続アクセスのプリフェッチを出力することを指示します。	○	○	×	○
PREFETCH_NOINFER	プリフェッチの距離が不明な場合は、連続アクセスのプリフェッチを出力しないことを指示します。	○	○	×	○
PREFETCH_ITERATION(n)	prefetch 命令を生成する際、ループのn 回転後に参照または定義されるデータを対象とすることを指示します。 n は1 ～ 10000 の10 進数です。	○	○	×	×
PREFETCH_ITERATION_L2(n)	prefetch 命令を生成する際、ループのn 回転後に参照または定義されるデータを対象とすることを指示します。 ただし、本機能では二次キャッシュにのみプリフェッチするprefetch 命令を対象とします。 n は1 ～ 10000 の10 進数です。	○	○	×	○

プリフェッチの最適化指示子(OCL) (2/2)

FUJITSU

最適化指示子	意味	指定可能な最適化制御行			
		プログラム単位	DOループ単位	文単位	配列代入文単位
PREFETCH_READ (name[,level={1|2}] [,strong={0|1}])	参照されているデータに対してprefetch 命令を生成することを指示します。 name は配列要素名、level はプリフェッチを行うキャッシュレベル、strongはstrong prefetchとするかどうかです。	×	×	○	×
PREFETCH_WRITE (name[,level={1|2}] [,strong={0|1}])	定義されているデータに対してprefetch 命令を生成することを指示します。 name は配列要素名、level はプリフェッチを行うキャッシュレベル、strongはstrong prefetchとするかどうかです。	×	×	○	×
PREFETCH_SEQUENTIAL [(AUTO|SOFT)]	連続的にアクセスされる配列データに対してprefetch命令を生成することを指示します。	○	○	×	○
PREFETCH_STRONG	一次キャッシュに対して生成されるprefetch命令をstrong prefetchとすることを指示します。	○	○	×	○
PREFETCH_NOSTRONG	一次キャッシュに対して生成されるprefetch命令をstrong prefetchとしないことを指示します。	○	○	×	○
PREFETCH_STRONG_L2	二次キャッシュに対して生成されるprefetch命令をstrong prefetchとすることを指示します	○	○	×	○
PREFETCH_NOSTRONG_L2	二次キャッシュに対して生成されるprefetch命令をstrong prefetchとしないことを指示します。	○	○	×	○

FUJITSU

プリフェッチの協調

- ハードウェアとソフトウェアのプリフェッチ協調

ハードウェアとソフトウェアのプリフェッチ協調

以下のケースではハードウェアプリフェッチが生成されない。ソフトウェアプリフェッチで補完することで、理想的な性能を実現することができる。

- ストリーム数が16個を超える場合

  → コンパイラが自動で識別し、ユーザが意識することなく行われている。

- ブロックストライド
  - ループブロッキングしているケース（配列置換、行列演算等に有効）
  - 外側ループでアンローリングしているケース
- マスク付きSIMDによるアクセス
  if文内にあるストリームをアクセスする際、if文の真率により連続アクセスとはならず、ハードウェアプリフェッチが生成されないケースがある。

ハードウェアプリフェッチが生成されないケースをソフトウェアプリフェッチで補う
という意味で「ハードウェアとソフトウェアのプリフェッチ協調」と呼ぶ。

FUJITSU

ビルトインプリフェッチ

- ビルトインプリフェッチについて

ビルトインプリフェッチについて

FUJITSU

C/C++ clangモードでは、ビルトインプリフェッチがサポートされている

- 指定方法

```
__builtin_prefetch(*addr, rw, localy)
```

  - *addr*：プリフェッチを行うメモリのアドレスを指定する
  - *rw*：プリフェッチを行う種類を指定する
    読み取りのプリフェッチの場合には0を、書き込みのプリフェッチの場合には1を指定する
  - *localy*：0~3を指定する
    L1プリフェッチを出す場合には3、L2プリフェッチを出す場合には2を指定する

FUJITSU

SIMD化（Single Instruction Multiple Data）

- SIMD化の基本原理
- 連続データのSIMD化例
- SIMD化の確認
- SIMD化可能なループ

※　ソースコードからの説明で使用している load/storeなどの命令は、アセンブライメージです。

SIMD化の基本原理

- SIMD (Single Instruction Multiple Data)
  1つの命令で，複数の演算を並列処理すること
- A64FX SVEのSIMDの特徴
  最大512bitのSIMD幅をサポート
  可変長SIMD幅をサポート
  様々なアクセス＆演算命令をサポート
  1つのコアで2つの積和演算命令を同時に実行可能

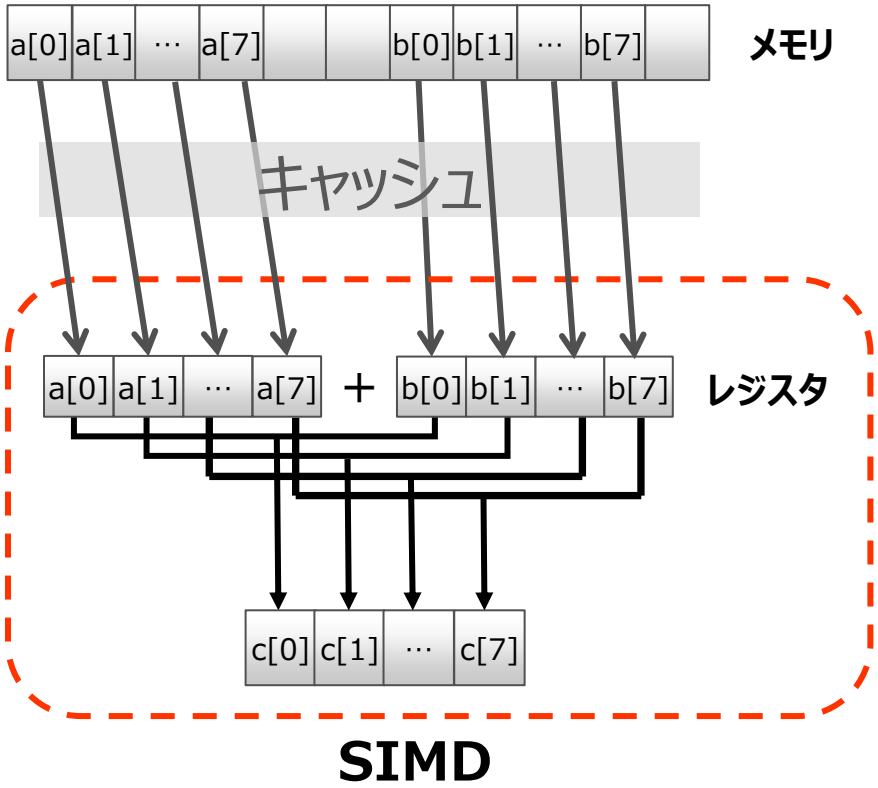
倍精度の場合：
1つのコアで16個(*)の演算処理を同時に実行可能
* 8SIMD幅 x 2演算パイプライン

単精度場合は32個の演算処理を同時に実行可能
整数型もSIMD可能

「アプリが使いやすいSIMD」、
「計算処理の高速化」を実現

プログラム例

```
for (i=0;i<7;++i) {
  c[ i ]=a[ i ]+b[ i ];
}
```



連続データのSIMD化例

FUJITSU

- 倍精度型データの場合

ソースコード

```
void foo(double a[N][N],
const double b[N][N], const double c[N][N])
{
    int i, j;
    for (i = 0; i < N; ++i) {
        for (j = 0; j < N; ++j) {
            a[ i ][ j ] = b[ i ][ j ] * c[ i ][ j ] ;
        }
    }
}
```

SIMD化しない場合

```
ldr     d1, [x15, x7, lsl #3]
ldr     x15, [x11]
fmul    d1, d1, d0
str     d1, [x15, x7, lsl #3]
```

→ スカラ命令

SIMD化した場合

```
ld1d    {z1.d}, p0/z, [x7, x1, lsl #3]
ld1d    {z0.d}, p0/z, [x4, x1, lsl #3]
fmul    z1.d, z0.d, z1.d
st1d    {z1.d}, p0, [x2, x1, lsl #3]
```

→ SIMD命令

b[i] [j]～ b[i] [j+7]のように1つの命令で8要素分のデータを処理します

最内ループの配列に対して８繰り返しの配列要素をSIMD命令化し、実行命令数を１/８にすることで高速化します。

SIMD化の確認

- 翻訳情報の出力例

```
(line-no.)(optimize)
      1            #define N 1000
      2            void foo(double a[N][N],const double b[N][N])
      3            {
      4              int i, j;
                   <<< Loop-information Sta
                   <<<  [PARALLELIZATION]
                   <<<    Standard iteration
                   <<< Loop-information  End >>>
      5  pp          for (i = 0; i < N; ++i) {
                   <<< Loop-information Start >>>
                   <<<  [OPTIMIZATION]
                   <<<    SIMD(VL: 8)
                   <<< Loop-information  End >>>
      6   p    2v      for (j = 0; j < N; ++j) {
      7   p    2v        a[ i ][ j ] = b[ i ][ j ] *
      8   p    2v      }
      9              }
     10              }
```

ループのSIMD化情報
- v ：SIMD化された
- m ：SIMD化された部分とSIMD化されなかった部分を含む
- s ：SIMD化されなかった
- 空白 ：SIMD化対象でない

8SIMD化された

実行文のSIMD化情報
- v ：SIMD化可能
- m ：SIMD化可能な部分とSIMD化不可能な部分を含む
- s ：SIMD化不可能

SIMD化可能なループ

FUJITSU

- コンパイラのSIMD化に必要なループの性質
  - ループの繰り返し回数がループに入る前に決定できること。(*1)
  - ループ内に手続/関数の呼出しが無いこと。(*2)
  - ループの出口が2つ以上無いこと。
  - ループ内に入出力文が無いこと。
  - データの定義・参照の順序（データの依存関係）が逐次実行と変わらないこと。

(*1) SVEではwhileループも可能。

(*2) インライン展開によりSIMD化可能になるケースあり。

FUJITSU

ソフトウェアパイプライニング

- ソフトウェアパイプライニングの基本原理
- ソフトウェアパイプライニングの確認

ソフトウェアパイプライニングの基本原理

FUJITSU

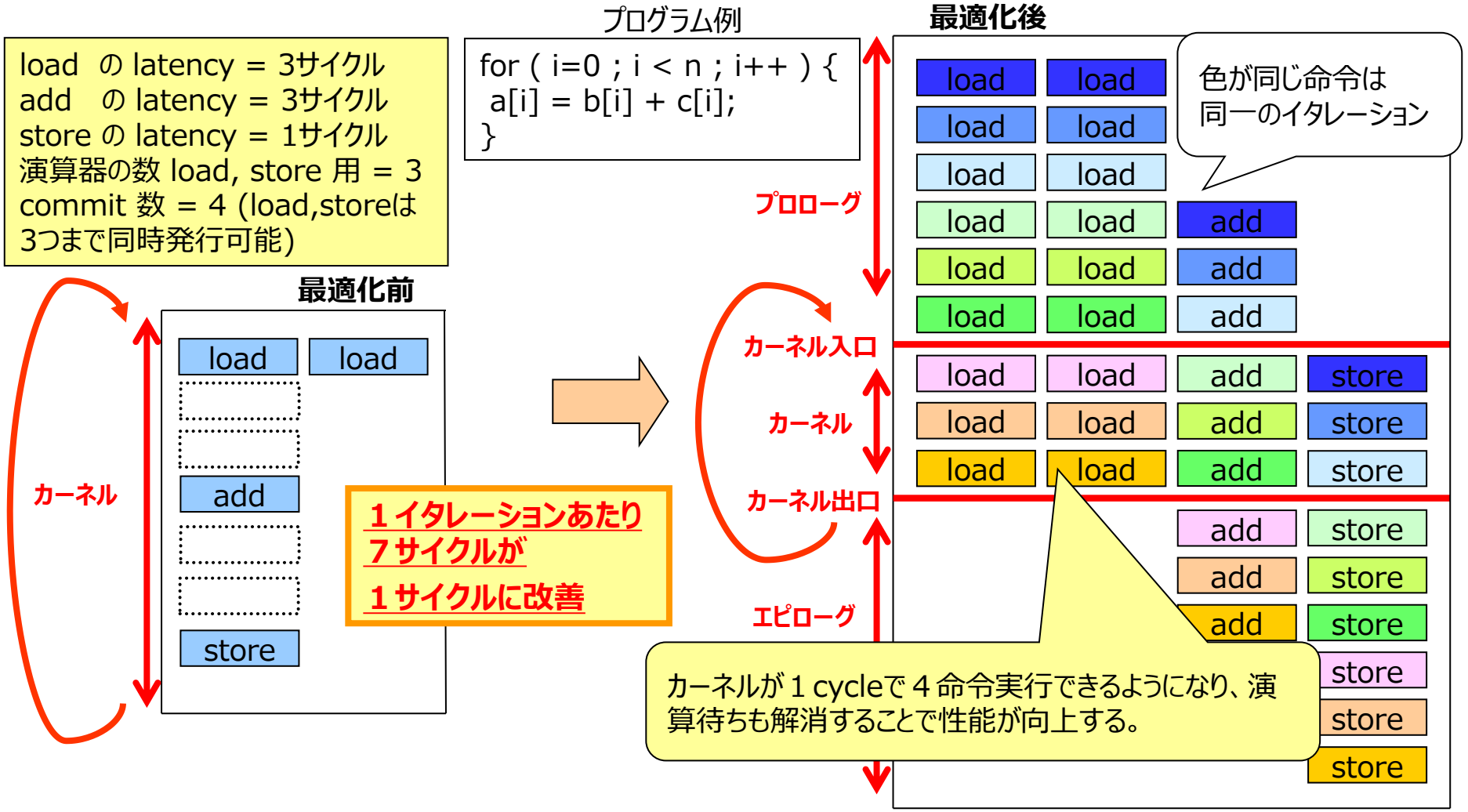
- ループの次のイタレーションを重ね合わせることで、カーネルループにおける命令レベルの並列性を向上させます。
  - 概念の説明のためマシンモデルは以下を想定

プログラムの特徴を最も表しているループを カーネルループ と呼んでいます。

load の latency = 3サイクル
add の latency = 3サイクル
store の latency = 1サイクル
演算器の数 load, store 用 = 3
commit 数 = 4 (load,storeは3つまで同時発行可能)

プログラム例

```
for ( i=0 ; i < n ; i++ ) {
 a[i] = b[i] + c[i];
}
```



ソフトウェアパイプライニングの確認

FUJITSU

- 翻訳情報の出力例

```
(line-no.)(optimize)
                :
    <<< Loop-information Start >>>
    <<<  [PARALLELIZATION]
    <<<    Standard iteration count: 1067
    <<<  [OPTIMIZATION]
    <<<    SIMD(VL: 8)
    <<<    SOFTWARE PIPELINING(IPC: 3.00, ITR: 144, MVE: 4, POL: S)
    <<< Loop-information  End >>>
 6   s     2v    for (i
 7   p     2v      a[ i
 8   p     2v    }
                :
```

ループがソフトウェアパイプライニングされたことを示します。

SOFTWARE PIPELINING(IPC: ipc, ITR: itr, MVE: mve, POL: pol)

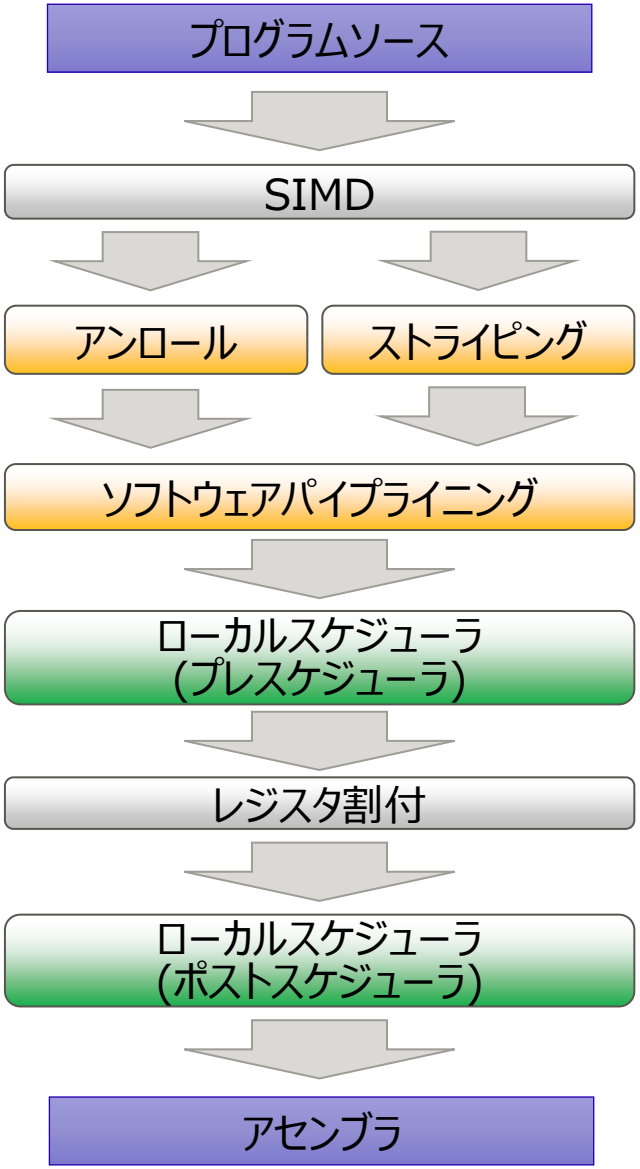
- ipcは、ソフトウェアパイプライニングが適用されたループの、サイクル当たりの命令数(Instructions Per Cycle)の予測値です。
- itrは、ソフトウェアパイプライニングが適用されたループが実行時に選択されるために、必要なループの繰返し数です。翻訳時メッセージjwd8205o-iで出力される値と同じです。
- mveは、ソフトウェアパイプライニングによるループの展開数です。
- polは、使用された命令スケジューリングアルゴリズムを示します。
  `S`は小さなループに、`L`は大きなループに適したアルゴリズムが使用されたこ とを意味します。

FUJITSU

ループ最適化と命令スケジューリング

- 最適化の順序
- アンロールの動き
- ストライピングの動き
- ループ最適化と制御の流れ

最適化の順序



- 最適化の順序と説明
  - SIMD
    - SIMD化を行う
  - アンロール
    - ループ内のすべての実行文をループ内へn重に展開し、ループの回転数を n分の1 にする最適化を行う
  - ストライピング
    - ループ内の実行文を一定数(ストライプ長)だけ展開する最適化を行う。実行文の入れ替えを強制的に行う。
  - ソフトウェアパイプライニング
    - ループの次のイタレーションを重ね合わせることで、カーネルループにおける命令レベルの並列性を向上させる最適化を行う
  - ローカルスケジューラ(プレスケジューラ)
    - 命令のレイテンシやレジスタのライブレンジを考慮して命令の入替え(スケジューリング)を行う
  - レジスタ割付
    - 変数や配列の使用するレジスタを割付けを行う
  - ローカルスケジューラ(ポストスケジューラ)
    - レジスタ割付で追加となった命令(スピル)を考慮して命令の入替え(スケジューリング)を行う

アンロールの動き

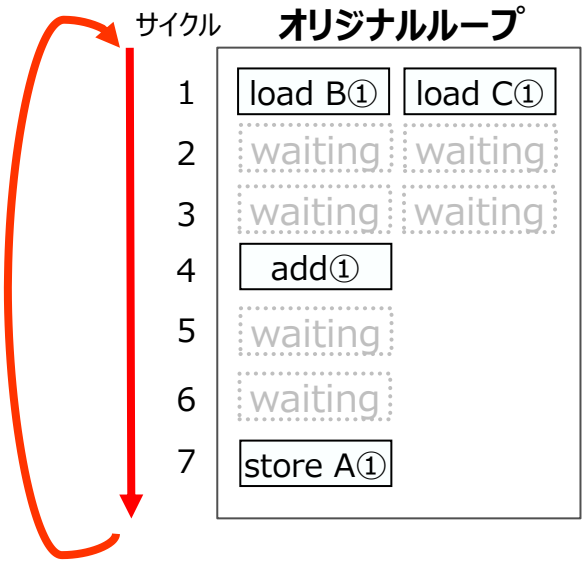
FUJITSU

- アンロール(SWPLなし) の動き
  - 概念の説明のためマシンモデルは以下を想定

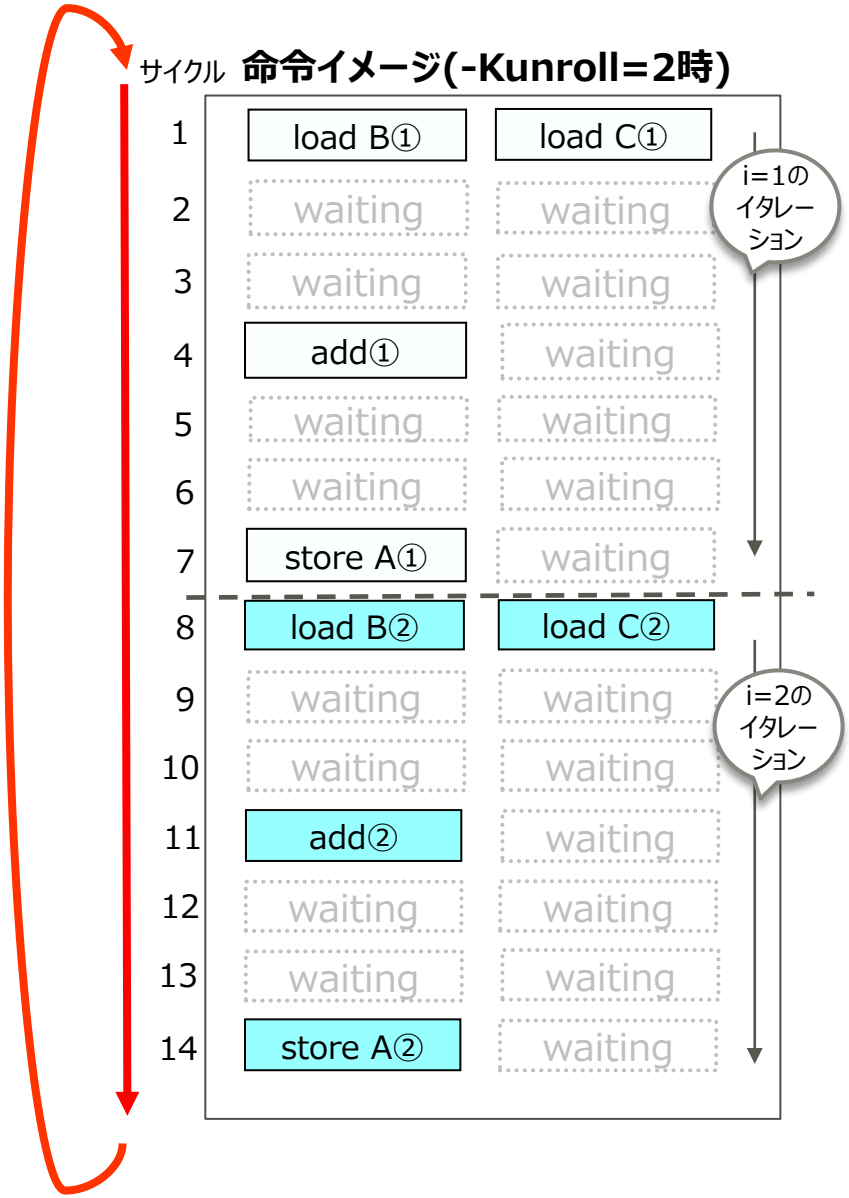
ロードレイテンシィ	3 サイクル
ADDレイテンシィ	3 サイクル
ストアレイテンシィ	1 サイクル
ロードパイプ数	2 本
ストアパイプ数	1 本
同時命令コミット数	4 命令

想定プログラム

```
do i=1,n
  A(i) = B(i) + C(i)
enddo
```



アンロールでは、複数イタレーションの命令を展開する(命令スケジューリングは行わない)



ストライピングの動き

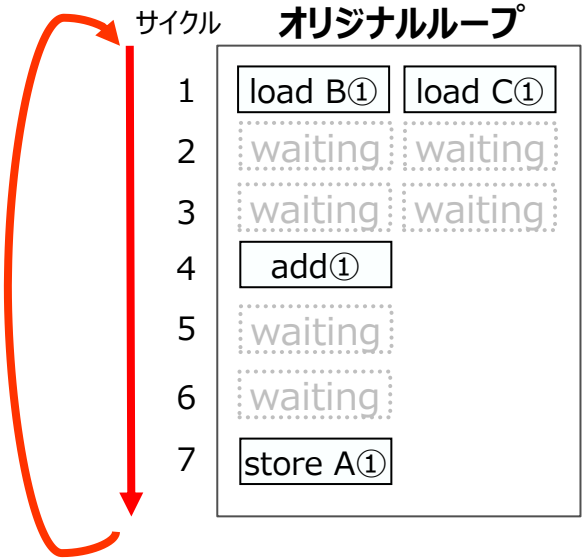
FUJITSU

- ストライピング(SWPLなし) の動き
  - 概念の説明のためマシンモデルは以下を想定

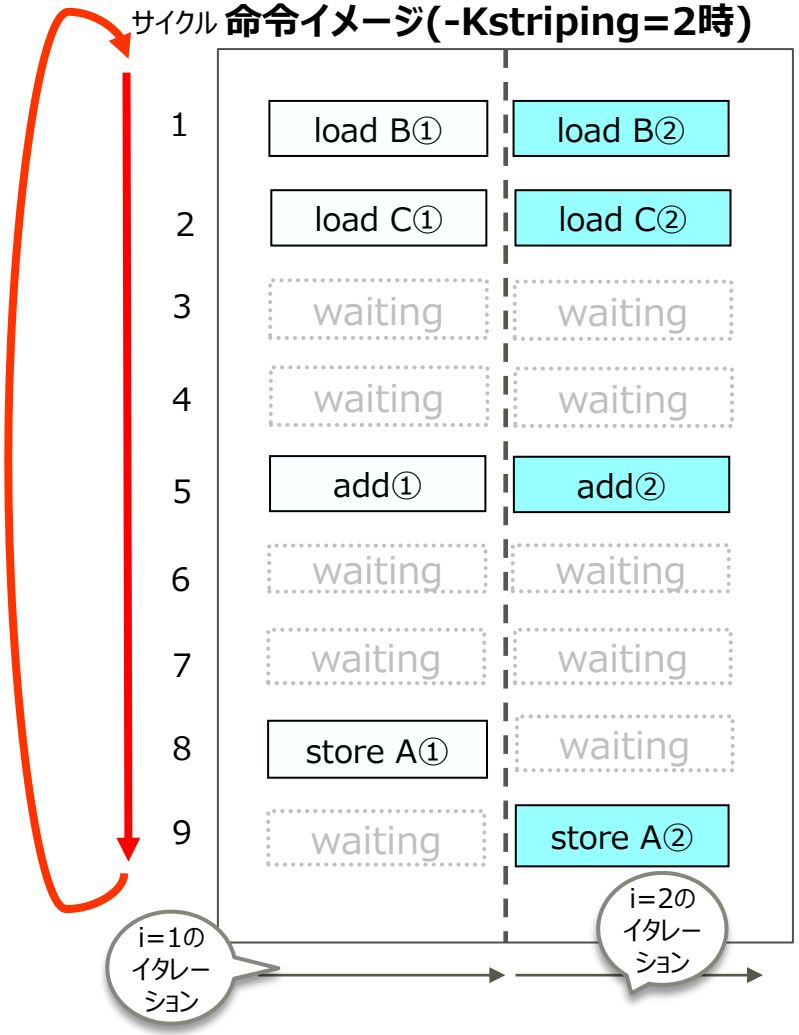
ロードレイテンシィ	3 サイクル
ADDレイテンシィ	3 サイクル
ストアレイテンシィ	1 サイクル
ロードパイプ数	2 本
ストアパイプ数	1 本
同時命令コミット数	4 命令

想定プログラム

```
do i=1,n
  A(i) = B(i) + C(i)
enddo
```



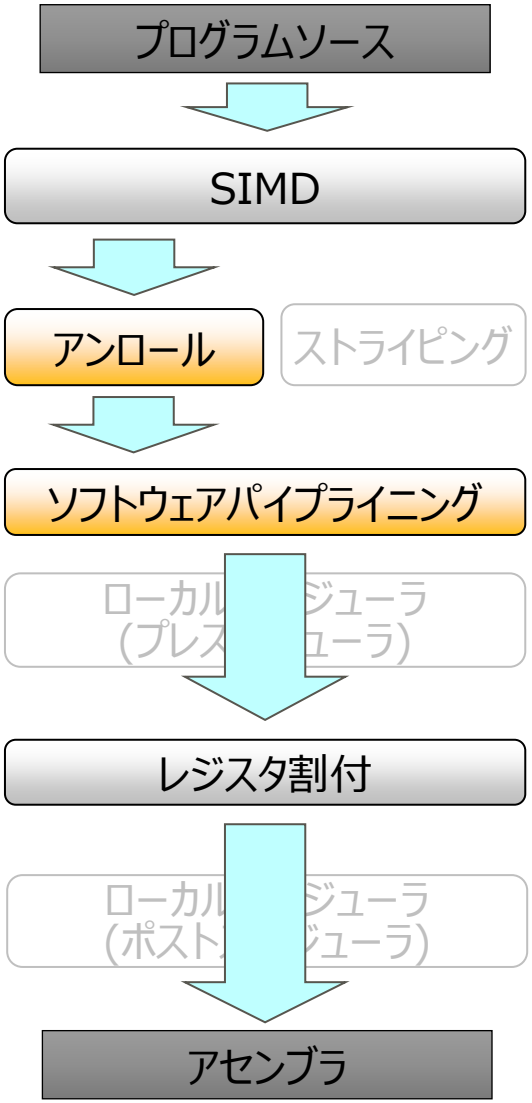
ストライピングでは、複数イタレーションの命令を交互に展開する(部分的に命令スケジューリングを行う)



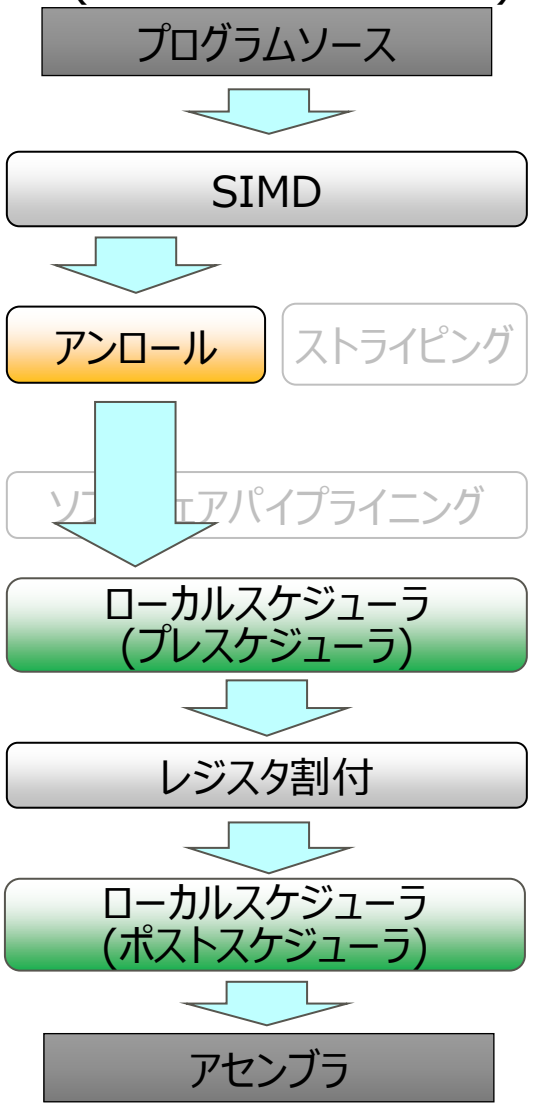
ループ最適化と制御の流れ

FUJITSU

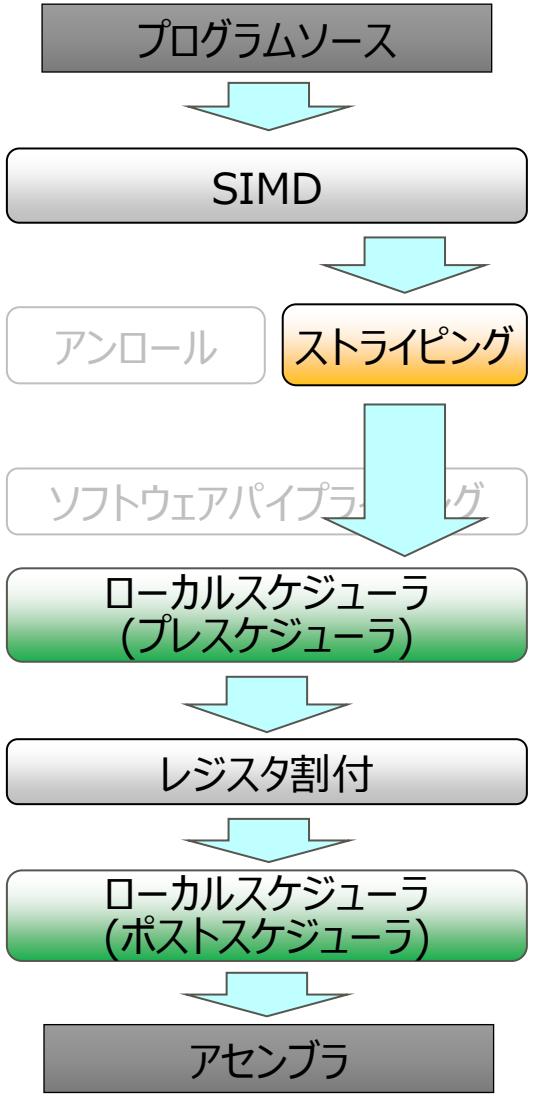
- ソフトウェアパイプライニング



- アンロール(SWPLなし)
  (+ローカルスケジューラ)



- ストライピング(SWPLなし)
  (+ローカルスケジューラ)



FUJITSU

ループに対する最適化

- ループ最適化について
- ループ交換
- ループ融合
- ループ展開
- ループ一重化

ループ最適化について

FUJITSU

- ループ最適化
  - コンパイラで自動的に行っている代表的なループ最適化について、以下に紹介します。

ループ最適化・変形の種類	効果
ループ交換 (loop interchange)	- データの局所化 - 外側ループでの並列化(粒度大)
ループ融合 (loop fusion)	- データの局所化 - 命令レベルの並列性向上
ループ展開 (loop unrolling)	- 命令の削減 - 命令レベルの並列性向上
ループ一重化 (loop collapse)	- スケジューリング効率向上 - ロードバランスの向上

ループ交換

FUJITSU

- 狙い
  - データの局所化
    - 配列bが連続アクセスとなることで、キャッシュの利用効率が向上します。

オリジナルソース	コンパイラ最適化イメージ

double a[N][M];
const double b[N][N][M];
void foo()
{
  int i, j, k;
  for (j = 0; j < M; ++j) {
  for (k = 0; k < N; ++k) {
  for (i = 0; i < N; ++i) {
  a[k][j] = a[k][j] + b[k][i][j];
  }
  }
  }
}
	double a[N][M];
	const double b[N][N][M];
	void foo()
	{
	  int i, j, k;
	  for (k = 0; k < N; ++k) {
	  for (i = 0; i < N; ++i) {
	  for (j = 0; j < M; ++j) {
	  a[k][j] = a[k][j] + b[k][i][j];
	  }
	  }
	  }
	}

ストライドアクセス

連続アクセス

ループ交換の確認

- 翻訳情報の出力例（ループ交換）

```
(line-no.)(optimize)
                    :
 <<< Loop-information Start >>>
            <<<  [OPTIMIZATION]
            <<<    INTERCHANGED(nest: 3)
            <<<    SIMD(VL: 8)
            <<<    SOFTWARE PIPELINING(IPC: 3.25, ITR: 192,
            <<< Loop-information  End >>>
 8   p      v    for (j = 0; j < M; ++j) {
            <<< Loop-information Start >>>
            <<<  [PARALLELIZATION]
            <<<    Standard iteration count: 2
            <<<  [OPTIMIZATION]
            <<<    INTERCHANGED(nest: 1)
            <<< Loop-information  End >>>
 9  pp           for (k = 0; k < N; ++k) {
            <<< Loop-information Start >>>
            <<<  [OPTIMIZATION]
            <<<    INTERCHANGED(nest: 2)
            <<< Loop-information  End >>>
10   p     2v       for (i = 0; i < N; ++i) {
11   p     2v         a[ k ][ j ] = a[ k ][ j ] + b[ k ][ i ][ j ];
12   p     2v       }
13              }
14            }                :
```

ループ交換が行われ
j のループがネスト3
k のループがネスト1
i のループがネスト2 となった

ループ融合

- 狙い
  - データの局所化
    - ループ融合することで、配列aが再利用できるようになります。
  - 命令レベルの並列性向上
    - ループ融合することで、ループ内の命令数が増え、命令スケジューリングにより命令レベルの並列性が向上します。

オリジナルソース	コンパイラ最適化イメージ

void sub()
{
  int i;
  for (i = 0; i < N; ++i) {
  a[i] = b[i] + c[i];
  }
  for (i = 0; i < N; ++i) {
  d[i] = a[i] + e[i];
  }
}
	void sub()
	{
	  int i;
	  for (i = 0; i < N; ++i) {
	  a[i] = b[i] + c[i];
	  d[i] = a[i] + e[i];
	  }
	}

ループ融合の確認

- 翻訳情報の出力例（ループ融合）

```
(line-no.)(optimize)
          :
             <<< Loop-information Start >>>
             <<<  [PARALLELIZATION]
             <<<    Standard iteration count: 728
             <<<  [OPTIMIZATION]
             <<<    FUSED(lines: 6,9)
             <<<    SIMD(VL: 8)
             <<<    SOFTWARE PIPELINING(IPC: 3.00, ITR: 144, MVE: 4, POL: S)
             <<<    PREFETCH(HARD) Expected by compiler :
             <<<      d, c, e, a, b
             <<< Loop-information  End >>>
 6  pp     2v    for (i = 0; i < N; ++i) {
 7   p     2v      a[ i ] = b[ i ] + c[ i ];
 8   p     2v    }
             <<< Loop-information Start >>>
             <<<  [OPTIMIZATION]
             <<<  FUSED
             <<< Loop-information  End >>>
 9    p     2v    for (i = 0; i < N; ++i) {
10    p     2v      d[ i ] = a[ i ] + e[ i ];
11    p     2v    }
          :
```

６行目のループと９行目のループが融合された

ループ展開

- 狙い
  - 命令の削減
    - ループの繰り返し回数が減少するため、分岐命令が削減されます。
    - b[i+1]+b[i+2]が共通式となるため、命令が削減されます。
  - 命令レベルの並列性向上
    - １回転当たりの命令数が増えることで、スケジューリングの余地が増え、命令レベルの並列性が向上します。

オリジナルソース

```
#define N 10000
double a[N], b[N];
void sub()
{
   int i;
   for (i = 0; i < N-3; ++i) {
      a[ i ] = b[ i ] + b[ i+1 ] + b[ i+2 ];
   }
}
```

コンパイラ最適化イメージ

```
#define N 10000
double a[N], b[N];
void sub()
{
   int i;
   double t;
   for (i = 0; i < N-3; i += 2) {
      t = b[ i+1 ] + b[ i+2 ];
      a[ i ] = b[ i ] + t;
      a[ i+1 ] = t + b[ i+3 ];
   }
}
```

ループ展開の確認

FUJITSU

- 翻訳情報の出力例（ループ展開）

```
(line-no.)(optimize)
           :
      <<< Loop-information Start >>>
      <<<  [PARALLELIZATION]
      <<<    Standard iteration count: 843
      <<<  [OPTIMIZATION]
      <<<    SIMD(VL: 8)
      <<<    SOFTWARE PIPELINING(IPC: 2.57, ITR: 96, MVE: 2, POL: S)
      <<< Loop-information  End >>>
6  pp    2v    for (i = 0; i < N-3; ++i) {
7   p    2v      a[ i ] = b[ i ] + b[ i+1 ] + b[ i+2 ];
8   p    2v    }
           :
```

ループが２展開された

ループ一重化

- 狙い
  - ソフトウェアパイプライニングの効率向上
    - 一重化したことにより、ループの繰り返し回数が増え、ソフトウェアパイプライニングの効果が向上します。
  - ロード・インバランスの改善
    - 一重化したことにより、Mの値に依存せずに、ロードバランスが向上する可能性が高まります。例えば、Mの値が1や2などの場合、オリジナルソースの外側で並列化するのは不利となります。

オリジナルソース	コンパイラ最適化イメージ

double a[M][N], c;
void sub()
{
  int i,j;
  c=1.0;
  for (j = 0; j < M; ++j) {
  for (i = 0; i < N; ++i) {
  a[j][i] = a[j][i] + c;
  }
  }
}
	double a[M*N], c;
	void sub()
	{
	  int i,j;
	  c=1.0;
	  for (ij = 0; ij < N*M; ++ij) {
	  a[ij] = a[ij] + c;
	  }
	}

ループ一重化の確認

FUJITSU

- 翻訳情報の出力例（ループ一重化）

```
(line-no.)(optimize)
              :
             <<< Loop-information Start >>>
             <<<  [PARALLELIZATION]
             <<<    Standard iteration count: 1231
             <<<  [OPTIMIZATION]
             <<<    COLLAPSED
             <<<    SIMD(VL: 8)
             <<<    SOFTWARE PIPELINING(IPC: 2.25, ITR: 192, MVE: 7, POL: S)
             <<< Loop-information  End >>>
 8  pp     2v    for (j = 0; j < M; ++j) {
             <<< Loop-information Start >>>
             <<<  [OPTIMIZATION]
             <<<    COLLAPSED
             <<< Loop-information  End >>>
 9   p     2v      for (i = 0; i < N; ++i) {
10   p     2v        a[j][i] = a[j][i] + c;
11   p     2v      }
12           }              :
```

ループが一重化された

FUJITSU

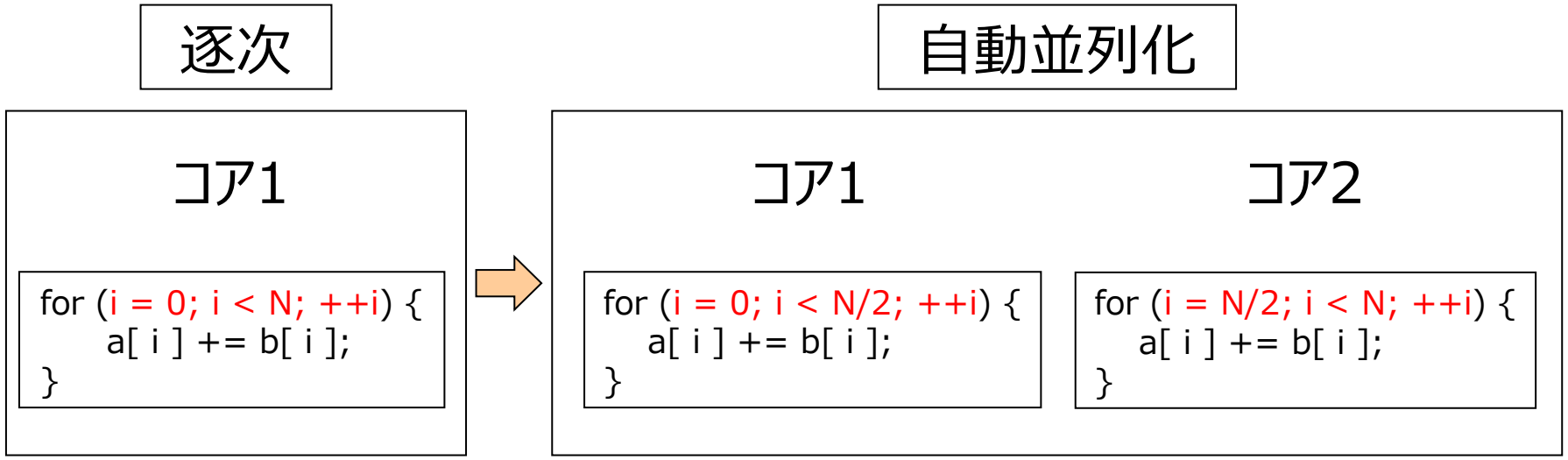
自動並列化

- 単純ループスライス
- リダクションによるループスライス
- 並列化可否の判断
- 自動並列化の確認
- パイプライン並列

単純ループスライス

FUJITSU

- ループインデックスを複数のスレッドに割り当てて並列化します。



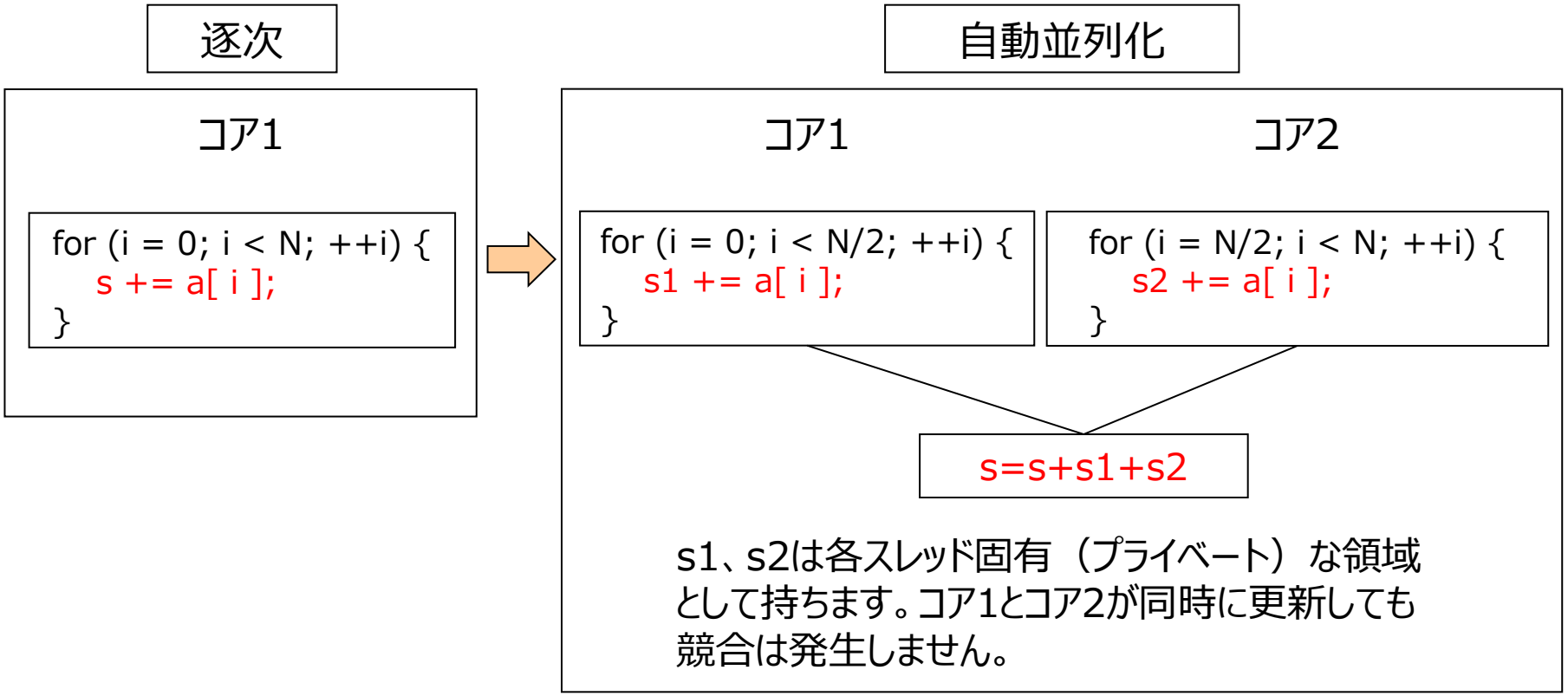
(2スレッドの場合の例)

- 並列の数は実行時に環境変数(PARALLEL)で指定します。

リダクションによるループスライス

FUJITSU

- データ競合を回避するようにして並列化



注意：逐次と演算順序が異なるため、計算誤差が発生する可能性があります。

-Knoeval や –Knoreduction が指定された場合は並列化できません。

並列化可否の判断

FUJITSU

- 以下のケースを、並列化対象から除外します。
  ①コンパイル時、ループのコストが少ないと判明。
  ②実行時、ループのコストが少ないと判明。
  ※コンパイラはループのコストが大きい時だけ並列化する動的制御を出力します。

①

```
double a[N][N], b[N][N];
void sub() {
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            a[ i ][ j ] = b[ i ][ j ] + 1.0;
        }
    }
}
```

②

```
extern double a[3][3], b[3][3];
void sub(int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            a[ i ][ j ] = b[ i ][ j ]+1.0;
        }
    }
}
```

自動並列化の確認

- 翻訳情報の出力例

```
(line-no.)(optimize)
                    :
              <<< Loop-information Start >>>
              <<<  [PARALLELIZATION]
              <<<    Standard iteration count: 1000
              <<<  [OPTIMIZAT
              <<<    SIMD(VL: 4
              <<<    SOFTWARE
              <<< Loop-informa
    5  pp     8v    for (i = 0; i < N; ++i) {
    6  p      8v      a[ i ] = b[ i ] + b[ i+1 ];
    7  p      8v    }
                    :
```

DOループの並列化情報
- pp ：並列化された
- m ：並列化された部分と並列化されなかった部分を含む
- s ：並列化されなかった
- 空白 ：並列化対象でない

実行文の並列化情報
- p ：並列化可能
- m ：並列化可能な部分と並列化不可能な部分を含む
- s ：並列化不可能

パイプライン並列

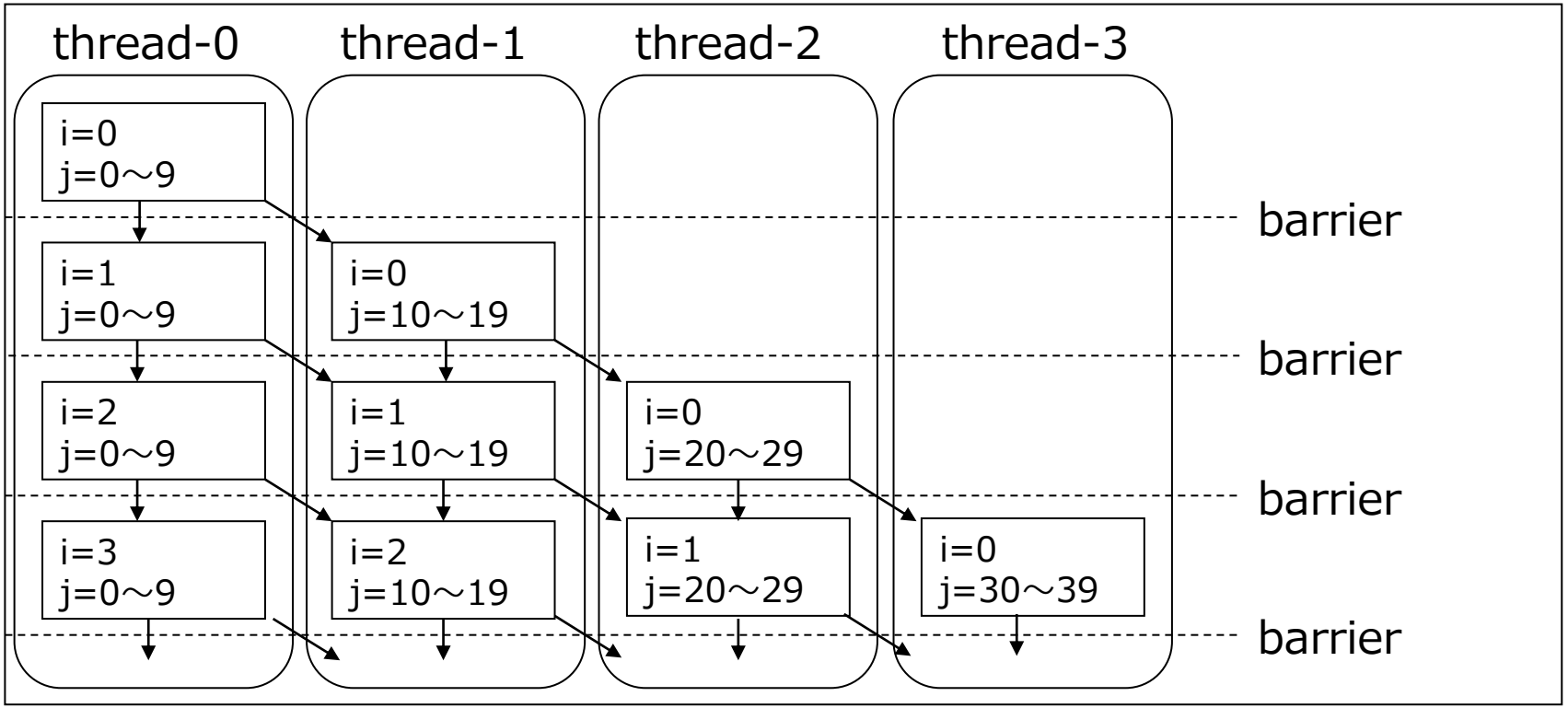
FUJITSU

- パイプライン並列

```
void sub() {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < 40; ++j) {
            a[ i ][ j ] = a[ i+1 ][ j ] + a[ i ][ j+1 ];
        }
    }
}
```

回転を跨いだデータ依存あり
→ 通常の並列化は不可能

斜め方向にはデータ依存なし
→ 斜めに内側ループをスライスして並列化



FUJITSU

Fortran、C/C++ tradモードのデバッグ機能

- 組込みデバッグ機能
- 異常終了時のデバッグ機能
- フック機能

組込みデバッグ機能(1/2)

FUJITSU

- C/C++コンパイラでは、以下の組込みデバッグ機能が利用できます。

翻訳オプション	-Nquickdbg
検査項目	subchk heapchk
出力情報	• エラーメッセージ • エラー発生時の行番号 • 変数名
スレッド並列対応	OpenMP、自動並列に対応

組込みデバッグ機能(2/2)

FUJITSU

- 使用方法
  次の翻訳オプションを指定

```
-Nquickdbg [=subchk | heapchk | inf_detail | inf_simple]
```

```
$ fccpx  -Nquickdbg  test.c
```

※ サブパラメタが指定されていない場合は以下と等価
−Nquickdbg=subchk −Nquickdbg=heapchk −Nquickdbg=inf_detail

- 検査用引数（サブパラメタ）

引数	検査内容
subchk	配列引用時の範囲検査
heapchk	メモリの解放および領域外書き込み検査

- 診断メッセージ表示用引数（サブパラメタ）

引数	表示内容
inf_detail	エラーメッセージ、行番号、変数名
inf_simple	エラーメッセージ、行番号　※実行性能への影響が軽減されます※

異常終了時のデバッグ機能(1/2)

- プログラムが異常終了した場合、原因追究の手助けとなる情報を実行時に出力できます。

翻訳オプション		-NRtrap
捕捉するシグナル		SIGILL(04)・・・不当な命令の実行 SIGFPE(08)・・・算術例外 SIGBUS(10)・・・記憶保護例外 SIGSEGV(11)・・・セグメンテーション例外 SIGXCPU(30)・・・CPU 占有時間による打切り
出力情報	一般的な異常終了時	• シグナル番号 • 異常終了の要因となったシグナルコード • シグナルコードに対する詳細情報
	SIGXCPU異常終了時	• メッセージ

異常終了時のデバッグ機能(2/2)

FUJITSU

- 使用方法
  次の翻訳オプションを指定

```
-NRtrap
```

```
$ fccpx -NRtrap  test.c
```

- 出力情報

シグナル番号	出力メッセージ
SIGILL SIGBUS SIGSEGV	jwe0019i-u The program was terminated abnormally with signal number SSSSSSS.signal identifier = NNNNNNNNNN,(Detailed information.) SSSSSSS : SIGILL、SIGBUS またはSIGSEGV NNNNNNNNNN : 異常終了の要因となったシグナルコード (Detailed information.) : シグナルコードに対する詳細情報
SIGXCPU	jwe0017i-u The program was terminated with signal number SIGXCPU.

フック機能(1/2)

- プログラムの特定箇所からユーザ定義関数を呼び出すことで、プログラムの動作確認に利用できます。

翻訳オプション	-Nhook_func
ユーザ定義関数名	user_defined_proc
ユーザ定義関数の引数	FLAG・・・ユーザ定義関数の呼び出し元情報 NAME・・・呼び出し元の関数名 LINE・・・呼び出し元の行番号 THREAD・・・スレッドの識別番号（OpenMP/自動並列化の場合）
ユーザ定義関数の呼び出し箇所	• プログラムの入口/出口 • 関数の入口/出口 -Kopenmpまたは-Kparallelが有効な場合、上記に加え以下の箇所からも呼び出されます。 • パラレルリージョン(OpenMP/自動並列化)の入口/出口

フック機能(2/2)

- 使用方法
  次の翻訳オプションを指定

```
-Nhook_func
```

```
$ fccpx  -Nhook_func  test.c
```

- ユーザ定義関数の形式
  - 形式

```
#include "fjhook.h"
void user_defined_proc(int *FLAG, char *NAME, int *LINE, int *THREAD)
```

  - 引数

FLAG：ユーザ定義関数の呼び出し元を示します。
　0：プログラムの入口　1：プログラムの出口　2：関数の入口　3：関数の出口
　4：パラレルリージョンの入口　5：パラレルリージョンの出口
　6～99：システムリザーブ　100：利用者側で利用可能
NAME：呼び出し元の関数名を示します。
　FLAGが2、3、4、5または100以上の場合のみ参照可能です。
LINE：呼び出し元の行番号を示します。
　FLAGが2、3、4、5または100以上の場合のみ参照可能です。
THREAD：ユーザ定義関数を呼び出したスレッドの識別番号を示します。(OpenMP/自動並列化)
　FLAGが2、3、4、5または100以上の場合のみ参照可能です。

FUJITSU

ラージページ

- ラージページについて
- ラージページ仕様
- ラージページ設定用環境変数
- メモリ使用量に関する注意事項(副作用)について
- ラージページのページングポリシー
- ラージページのロックタイプ
- ラージページ不足時の動作(富岳のみ)

ラージページについて

FUJITSU

- ラージページとは
  - 大規模なデータを扱うアプリケーションに対して、通常のページ(ノーマルページ)より大きなページサイズのメモリ(ラージページ)を割り当てることで、
    - CPUのアドレス変換処理によるオーバーヘッドを低減します。
    - メモリアクセス性能を向上させます。
  - A64FXシステム環境での、ノーマルページのサイズは64KiBであり、ラージページとして利用可能なサイズは、2MiBです。
    - 環境変数設定により以下の動作設定が可能
      - ラージページ割り当て動作の有効化/無効化
      - スタック領域のラージページ割り当て動作の有効化/無効化
      - 各メモリ領域のページング方式(ページの割り当て契機)の選択
    - 32MiB/1GiB/16GiBなどの様々なページサイズはMcKernelで実現可能。

※詳細は、ジョブ運用ソフトウェア　エンドユーザ向けガイド　HPC拡張機能編 の 第3章 ラージページライブラリ を参照してください。

ラージページ仕様

FUJITSU

- ラージページ仕様

メモリ領域	MP10/FX10/FX100	A64FX			
	ページサイズ	ページサイズ			ページング (_付はdefault)
		ノーマルページ	ラージページ base	ラージページ base+stack (default)	
テキスト (.text)	8KiB	64KiB	64KiB	64KiB	–
静的データ (.data)	4MiB (default), 8KiB, 32MiB, 256MiB	64KiB	2MiB	2MiB	常にprepage
静的データ (.bss)		64KiB	2MiB	2MiB	demand | prepage
スタック (*1)		64KiB	64KiB	2MiB	demand | prepage
動的メモリ (*2)		64KiB	2MiB	2MiB	demand | prepage
共有メモリ		64KiB	64KiB	64KiB	–

＊１：プロセススタック/メインスレッド用スタック/スレッドスタック領域が対象
＊２： プロセスヒープ/メインスレッド用ヒープ/スレッドヒープ/mmap領域が対象

ラージページ設定用環境変数 (1/2)

FUJITSU

●基本設定/ページング方式の設定

環境変数名	指定値 (_付はdefault)	説明
XOS_MMM_L_HPAGE_TYPE	hugetlbfs | none	ラージページライブラリによるラージページ割り当て動作の有効化/無効化を選択する設定です。 「hugetlbfs」の場合、HugeTLBfsによるラージページ化をします。 「none」の場合、ラージページライブラリによるラージページ化は行いません。
XOS_MMM_L_LPG_MODE	base+stack | base	スタック領域およびスレッドスタック領域のラージページ割り当て動作の有効化/無効化を選択する設定です。 「base+stack」の場合、静的データおよび動的メモリ確保領域だけではなく、スタック領域およびスレッドスタック領域もラージページ化します。 「base」の場合、静的データおよび動的メモリ確保領域のみラージページ化します。スタック領域およびスレッドスタック領域はラージページ化しません。
XOS_MMM_L_PAGING_POLICY	[demand | prepage]: [demand | prepage]: [demand | prepage]	各メモリ領域のページング方式(ページの割り当て契機)を選択する設定です。 「demand」はデマンドページング方式、「prepage」はプリページング方式を意味します。本変数はコロン(:)区切りで3つのメモリ領域のページング方式を指定します。 第1指定は、静的データの.bss領域です。(静的データの.data領域はページング方式指定の対象外で常にprepageとなります。) 第2指定は、スタック領域およびスレッドスタック領域です。 第3指定は、動的メモリ確保領域です。 指定値以外の値を指定した場合は、「prepage:demand:prepage」を指定したものとみなします。

ラージページ設定用環境変数 (2/2)

FUJITSU

- チューニング用の設定 (ラージページライブラリ独自の環境変数)

環境変数名	指定値 (_付はdefault)	説明
XOS_MMM_L_ARENA_FREE	1 | 2	free(3)で解放されるヒープ領域の扱いに関する設定です。 「1」を指定した場合は、解放可能なメモリを即時に解放します。「2」を指定した場合は、メモリを一切解放せず、全メモリをプールして再利用します。
XOS_MMM_L_ARENA_LOCK_TYPE	0 | 1	メモリ割り当てポリシーに関する設定です。 「0」はメモリ獲得性能優先、「1」はメモリ使用効率優先を意味します。
XOS_MMM_L_MAX_ARENA_NUM	1 以上INT_MAX 以下の整数値 [10進数]	生成可能なアリーナ(プロセスヒープとスレッドヒープ領域の総和)の数を設定します。XOS_MMM_L_ARENA_LOCK_TYPE=0のときに有効になります。
XOS_MMM_L_HEAP_SIZE_MB	MALLOC_MMAP_THRESHOLD_の2倍 以上 ULONG_MAX以下の整数値 <MiB単位> [10進数]	スレッドヒープ領域を使用する場合に、スレッドヒープ領域の生成時および拡張時に獲得するメモリサイズを設定します。
XOS_MMM_L_COLORING	0 | 1	キャッシュカラーリングの有無を設定します。プロセッサのL1キャッシュのコンフリクトを軽減します。「0」の場合、キャッシュカラーリングを行いません。 「1」の場合、MALLOC_MMAP_THRESHOLD_(デフォルトは128MiB)以上のサイズで行われるmmap(2)によるメモリ獲得時にはキャッシュカラーリングをします。
XOS_MMM_L_FORCE_MMAP_THRESHOLD	0 | 1	MALLOC_MMAP_THRESHOLD_(デフォルトは128MiB)以上のサイズのメモリ獲得時にmmap(2)を優先するかどうかの設定です。 「0」の場合、mmap(2) は優先しません。まずヒープ領域の空きを検索し、空きがあればヒープ領域の空きメモリを返します。ヒープ領域の空きが見つからないときにのみ mmap(2) でメモリを獲得します。「1」の場合、mmap(2) を優先します。ヒープ領域の空きは検索せず、(例え空きがあっても)mmap(2) でメモリを獲得します。

- glibcの環境変数(MALLOC_MMAP_THRESHOLD_等)についてはユーザ向けガイドを参照

メモリ使用量に関する注意事項(副作用)について

FUJITSU

- 静的データ(.data)領域

  常に①、②の副作用が発生します。

- 静的データ(.bss)領域

  以下の条件を満たす場合、①、②の副作用が発生します。

  a. .dynsym セクション(dynamic section)に存在するシンボルである
  b. シンボルが(メインプログラムの) bss 領域(bss_start, bss_end)の範囲内のアドレスにある
  c. シンボルがグローバル(STB_GLOBAL)またはウィーク(STB_WEAK)である
  d. シンボルのタイプが変数(STT_OBJECT)である
  e. シンボルのサイズが 0 より大きい

- 副作用

  ① 2倍または3倍のメモリ領域を使用することがあります。
  ② XOS_MMM_L_PAGING_POLICYで静的データ(.bss)領域のページング方式をデマンドページング方式("demand")に設定していても、プリーページング方式("prepage")で動作します。

ラージページのページングポリシー

- XOS_MMM_L_PAGING_POLICY=prepage:demand:prepage
  - スレッド並列で複数CMGにて走行する時は、以下のメモリアロケーションとなる。
    - プリページングでは、ロードモジュール起動時に、CMG0からデータが載る。
    - デマンドページングでは、初回アクセス時に実行CMGにデータが載る。
  - 複数CMGをまたぐときはデマンドページングを推奨します。

```
14                    Subroutine sub(n,iter,x1,x2,y1)
15                      real(8) :: x1(n), x2(n), y1(n),c0
16                      integer n,i,k
17                      c0=2.0
 :                        ……
20    1                  do k=1,iter
21    1                 !$omp parallel do
            <<< Loop-information Start >>>
            <<<  [OPTIMIZATION]
            <<<    SIMD(VL: 8)
            <<<    SOFTWARE PIPELINING(IPC: 2.45, ITR: 128, MVE: 2, POL: S)
            <<<    PREFETCH(SOFT) : 10
            <<<     SEQUENTIAL : 10
            <<<      x2: 4, x1: 4, y1: 2
            <<<    ZFILL          :
            <<<      y1
            <<< Loop-information  End >>>
22    2  p   v           do i=1,n
23    2  p   v             y1(i) = x1(i) + c0 * x2(i)
24    2  p   v           end do
25    1                 enddo
 :                        ……
30                    parameter(N=45000000,ITER=100)
31                    real*8 x1(N),x2(N),y1(N)
32                    call init(N,ITER,x1,x2,y1)
33                    call sub(N,ITER,x1,x2,y1)
```

ストリーム(データサイズは約1GB)

プリページング(prepage指定)ではCMG0からデータが載るため、48スレッドのストリームの性能がでません。
デマンドページング(demand指定)に変更することで実行CMGにデータが載り、性能が大幅に向上します。

ページングポリシー	Memory throughput (GB/s)
prepage指定 (default)	93GB/s
demand指定	804GB/s

実行時のページングポリシー:
XOS_MMM_L_PAGING_POLICY=
demand:demand:demand

ラージページのロックタイプ

- XOS_MMM_L_ARENA_LOCK_TYPE
  - メモリ割り当てポリシーに関する設定です。
  - 「0」はメモリ獲得性能優先。パラレルリージョン内でmallocしている場合に推奨します。(メモリの獲得/開放がスレッド毎に独立したメモリプールから行われるようになり、デフォルト設定よりも排他制御のコストが減って性能改善する場合があります)
  - 「1」はメモリ使用効率優先 (デフォルト)。メモリ使用量が多い場合に推奨します。

```
 1                  subroutine sub(n,m,iter,x1,x2,y2)
 2                    integer(8) :: pZ1(iter)
 3                    real(8) :: x1(n), x2(n), y2(n,m),c0
 4                    c0=2.0
 5
 6                  !$omp parallel do shared(n,m,iter,x1,x2,c0,y2)
                                      private(pZ1,i,j,k) default(none)
 :                     ......
 7    1   p           do k=1,m
 :                     ......
 8    2   p   s         do j=1,iter
 9    2   p   m           pZ1(j) = malloc(8 * n)
10    2   p   v         end do
11    1
 :                     ......
12    2   p   2v        do i=1,n
13    2   p   2v          y2(i,k) = x1(i) + c0 * x2(i)
14    2   p   2v        end do
15    1
16    2   p   s         do j=1,iter
17    2   p   s           call free( pZ1(j))
18    2   p   s         end do
19    1   p           end do
20                    end subroutine sub
21
22                    program main
23                      parameter(N=1048512,ITER=80)
24                      real*8 x1(N),x2(N),y2(N,12)
25                      call sub(N,12,ITER,x1,x2,y2)
```

ラージページのロックタイプ(環境変数XOS_MMM_L_ARENA_LOCK_TYPE)をメモリ使用率優先(1)からメモリ獲得優先(0)にすることで、mallocの性能が向上

ロックタイプ	実行時間(秒)
使用効率優先(=1) [default]	0.56
メモリ獲得性能優先(=0)	0.35

実行時のロックタイプ:
XOS_MMM_L_ARENA_LOCK_TYPE=0

ラージページ不足時の動作(富岳のみ)

FUJITSU

- XOS_MMM_L_HUGETLB_FALLBACK
  - malloc時にラージページが不足した場合の動作を指定する設定です。
  - [1]を指定すると、不足分のメモリについてノーマルページによる割り当てを試みます。
    ノーマルページでもmalloc要求を満たすメモリを獲得できない場合は、メモリ枯渇でプロセスが終了します。本設定は、例えば、ノーマルページ指定(XOS_MMM_L_HPAGE_TYPE=none)では動作するが、ラージページ指定ではメモリ枯渇で実行に失敗するようなプログラムで効果が期待できます。本設定を有効にすることで、最大限ラージページを使いつつ不足分をノーマルページで補い、mallocで獲得できるメモリの総量を増やすことで、プログラムが動作するようになる可能があります。
  - [0]を指定すると、ラージページが不足した時点でメモリ枯渇でプロセスが終了します(デフォルト)。
  - 注意点1: 本設定を有効化するためには、以下の設定が必要です。
    I. XOS_MMM_L_HPAGE_TYPE=hugetlbfsである、かつ、
    - II. XOS_MMM_L_PAGING_POLICY=any:any:prepageである、かつ、
    - III. XOS_MMM_L_ARENA_LOCK_TYPE=1である、かつ、
    - XOS_MMM_L_MAX_ARENA_NUM=1である。
    - この4つの条件はラージページライブラリのデフォルト設定であるため、特に他の指定を行っていなければ明に指定は不要です。
  - 注意点2: 本設定を有効化(1を指定)した場合、ノーマルページとラージページが混在したmalloc領域を生成するが、この領域をTofuの通信バッファとして使用した場合、Tofu上ではこの領域全体はノーマルページとして管理します。そのため、通信バッファとして利用した際の通信性能については、ノーマルページ指定の場合からの向上は見込めません。

FUJITSU

Fortranがサポートするタイマー

- タイマーの仕様
- タイマーの精度

タイマーの仕様 (1/3)

FUJITSU

- 主要なタイマールーチンの仕様

No.	ルーチン名	機能	形式	計測対象	ルーチンが返す単位
1	DATE_AND_TIME 組込みサブルーチン	日付と時間を取得します。	CALL DATE_AND_TIME ([DATE , TIME , ZONE , VALUES]) DATE：現在の日付がCCYYMMDD の形式で設定される。 （CC:世紀,YY:西暦年,MM:月,DD:日） TIME：現在の時間がhhmmss.sssの形式で設定される。 （hh:時,mm:分,ss.sss:秒） ZONE：UTC からの時差がshhmm の形式で設定される。 （s:符号,hh:時,mm:分） VALUES(1)：年 VALUES(2)：月 VALUES(3)：日 VALUES(4)：分で表したUTCからの時差。 VALUES(5)：時間 VALUES(6)：分 VALUES(7)：秒 VALUES(8)：ミリ秒	現在の日時	ー
2	GETTIM サービスサブルーチン	現在の時刻を取得します。	CALL GETTIM (hour , minute , second , second1_100) hour：現在の時が設定される。 minute：現在の分が設定される。 second：現在の秒が設定される。 second1_100：現在の100分の1秒が設定される。	現在の時間	ー
3	FDATE サービスサブルーチン	現在の日付と時刻をASCII コードに変換して、取得します。	CALL FDATE (string) string：現在の日付と時刻が曜日、月、日、時刻、年の順番で設定される。	現在の日付と時刻	ー
4	ITIME サービスサブルーチン	現在の時、分、秒を取得します。	CALL ITIME (ia) ia(1)：現在の時が設定される。 ia(2)：現在の分が設定される。 ia(3)：現在の秒が設定される。	現在の時、分、秒	ー
5	GETTOD サービスサブルーチン	現在の実時間を取得します。実時間は過去のある任意の時間からのマイクロ秒単位の経過時間です。 通常、システムブートからの時間で表されます。	CALL GETTOD (g) g：マイクロ秒単位の経過時間が設定される。	wall clock time	マイクロ秒

タイマーの仕様 (2/3)

FUJITSU

- 主要なタイマールーチンの仕様

No.	ルーチン名	機能	形式	計測対象	ルーチンが返す単位
6	GMTIME サービスサブルーチン	指定したシステム時間をグリニッジ標準時間に従って、秒、分、時間、日、月、年、曜日、1月1日からの 通算日付、夏時間かどうかを表す情報を取得します。	CALL GMTIME (time , t) time : システム時間を指定する。 timeに指定したシステム時間がグリニッジ標準時間に従い、以下の配列に設定される。 t(1) : 秒 t(2) : 分 t(3) : 時 t(4) : 日 t(5) : 月 t(6) : 1900年からの通算年 t(7) : 日曜日からの通算曜日 t(8) : 1月1日からの通算日 t(9) : 標準時間は0、夏時間は1が設定される。	wall clock time	ー
7	LTIME サービスサブルーチン	指定したシステム時間をローカル時間に従って、秒、分、時間、日、月、年、曜日、1月1日からの 通算日付、夏時間かどうかを表す情報を取得します。	CALL LTIME (time , t) time : システム時間を指定する。 timeに指定したシステム時間がローカル時間に従い、以下の配列に設定される。 t(1) : 秒 t(2) : 分 t(3) : 時 t(4) : 日 t(5) : 月 t(6) : 1900年からの通算年 t(7) : 日曜日からの通算曜日 t(8) : 1月1日からの通算日 t(9) : 標準時間は0、夏時間は1が設定される。	wall clock time	ー
8	omp_get_wtime ルーチン	現在の実時間を取得します。実時間は過去のある任意の時間からの秒単位の経過時間です。通常、システムブートからの時間で表されます。	y = omp_get_wtime （ ） y : 秒単位の経過時間が返却される。	wall clock time	秒
9	SECNDS サービス関数	午前0時からのシステム時間の経過秒数から第1引数で指定した秒数を引いた秒数を取得します。	y = SECNDS (sec) y : 午前0時からのシステム時間の経過秒数からsec 秒を引いた秒数が返却される。 sec : 午前0時からのシステム時間の経過秒数から引く値を秒単位で指定する。	wall clock time	秒
10	SYSTEM_CLOCK 組込みサブルーチン	午前0時から現在までの通算時間を取得します。通算時間は秒単位の経過時間です。通常、システムブートからの時間で表されます。	CALL SYSTEM_CLOCK ([COUNT , COUNT_RATE , COUNT_MAX]) COUNT : 午前0時から現在までの経過時間が設定される。 COUNT_RATE : 1秒間に処理系が刻む回数(=1000)が設定される。 COUNT_MAX : COUNT の最大値(=86399999)が設定される。	wall clock time	ミリ秒
11	RTC サービス関数	1970 年1月1日午前0時からのUTC の通算秒を取得します。	y = RTC ( ) y : 1970年1月1日午前0時からのUTC の通算秒が設定される。	wall clock time	秒

タイマーの仕様 (3/3)

FUJITSU

- 主要なタイマールーチンの仕様

No.	ルーチン名	機能	形式	計測対象	ルーチンが返す単位
12	TIME サービス関数	00:00:00 GMT（1970年1月1日）からの秒単位の経過時間を取得します。	iy = TIME ( ) iy : 00:00:00 GMT（1970年1月1日）からの秒単位の経過時間が返却される。	wall clock time	秒
13	TIMEF サービス関数	直前に呼ばれたTIMEF サービス関数からの経過時間を返却します。	y = TIMEF ( ) y : 直前に実行されたTIMEFサービス関数からの経過時間が返却される。	wall clock time	秒
14	TIMER サービスサブルーチン	午前0時からの通算1/100秒を取得します。	CALL TIMER(ix) ix : 午前0時からの通算1/100秒が設定される。	wall clock time	秒
15	CLOCK サービスサブルーチン	実行可能プログラムの実行開始からのCPU 時間を取得します。CPU時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU時間です。	CALL CLOCK (g , i1 , i2) g ：i1で指定した単位のCPU時間が設定される。 i1 : 返却単位（秒単位、ミリ秒単位、マイクロ秒単位）を指定。 i2 : gに指定した変数の型を指定。	現行プロセスとそのプロセス内の全スレッドで利用されたCPU時間	指定に従い以下のいずれか。 ・秒 ・ミリ秒 ・マイクロ秒単位
16	CLOCKM サービスサブルーチン	実行可能プログラムの実行開始からのCPU 時間を取得します。CPU時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU時間です。	CALL CLOCKM (i) i : ミリ秒単位のCPU時間が設定される。	現行プロセスとそのプロセス内の全スレッドで利用されたCPU時間	ミリ秒
17	CLOCKV サービスサブルーチン	実行可能プログラムの実行開始からのCPU 時間を取得します。CPU時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU時間です。 ベクトル機の互換ルーチン。	CALL CLOCKV (g1 , g2 , i1 , i2) g1 : 常に0が設定される。※ベクトル機ではVUタイムが設定されていた。 g2 : i1で指定した単位のCPU時間が設定される。 i1 : 返却単位（秒単位、ミリ秒単位、マイクロ秒単位）を指定。 i2 : g2に指定した変数の型を指定。	現行プロセスとそのプロセス内の全スレッドで利用されたCPU時間	指定に従い以下のいずれか。 ・秒 ・ミリ秒 ・マイクロ秒単位
18	CPU_TIME 組込みサブルーチン	実行可能プログラムの実行開始からのCPU 時間を取得します。CPU時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU時間です。	CALL CPU_TIME (TIME) TIME : 秒単位のCPU時間が設定される。	現行プロセスとそのプロセス内の全スレッドで利用されたCPU時間	秒
19	DTIME サービス関数	直前に呼ばれたDTIME サービス関数からのCPU 時間を取得します。CPU時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU時間です。	y = DTIME (tm) y : 直前に呼ばれたDTIME サービス関数からのCPU 時間が返却される。 tm(1) : 秒単位のユーザCPU時間が設定される。 tm(2) : 秒単位のシステムCPU時間が設定される。	現行プロセスとそのプロセス内の全スレッドで利用されたCPU時間	秒
20	ETIME サービス関数	実行可能プログラムの実行開始からのCPU 時間を取得します。CPU時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU時間です。	y = ETIME (tm) y : 実行可能プログラムの実行開始からのCPU 時間を返却される。 tm(1) : 秒単位のユーザCPU時間が設定される。 tm(2) : 秒単位のシステムCPU時間が設定される。	現行プロセスとそのプロセス内の全スレッドで利用されたCPU時間	秒
21	SECOND サービス関数	実行可能プログラムの実行開始からのユーザCPU 時間を取得します。CPU時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU時間です。	y = SECOND ( ) y : 実行可能プログラムの実行開始時からのユーザCPU時間が秒単位で返却される。	現行プロセスとそのプロセス内の全スレッドで利用されたCPU時間	秒

タイマーの精度 (1/2)

FUJITSU

- 主要なタイマールーチンの精度(タイマーのオーバーヘッド)

No.	ルーチン名	精度分解能	オーバーヘッド (μs)	スレッドセーフ性	実装	GCCオーバーヘッド(μs)	富士通/GCC	備考
1	DATE_AND_TIME 組込みサブルーチン	1/1,000,000	77.37 (改善版) 47.27	○	gettimeofday localtime	163.13	0.47	―
2	GETTIM サービスサブルーチン	1/1,000,000	38.37	○	gettimeofday localtime	―	―	―
3	FDATE サービスサブルーチン	1/1,000,000	18.58	○	time ctime_r	78.71	0.24	―
4	ITIME サービスサブルーチン	1/1,000,000	36.07	○	time localtime	69.27	0.52	―
5	GETTOD サービスサブルーチン	1/100,000,000	0.02	○	arm asm命令 time stamp counter gmtime_r localtime_r	―	―	分解能は、(1/cntfrq_el0)の値
6	GMTIME サービスサブルーチン	―	5.72	○	gmtime_r	58.21	0.10	―
7	LTIME サービスサブルーチン	―	10.81	○	localtime_r	67.59	0.16	―
8	omp_get_wtime ルーチン	FJOMP: 1/100,000,000 libomp: 1/1,000,000	1.00	○	FJOMP: arm asm命令　time stamp counter libomp: gettimeofday localtime	6.05	0.17	分解能は、(1/cntfrq_el0)の値
9	SECNDS サービス関数	1/1,000,000	49.05	○	gettimeofday localtime	77.31	0.63	―
10	SYSTEM_CLOCK 組込みサブルーチン	1/100,000,000	20.98 (改善版) 1.42	○	arm asm命令 time stamp counter	5.23	4.01 (改善版) 0.27	分解能は、(1/cntfrq_el0)の値
11	RTC サービス関数	1/1,000,000	5.21	○	time	―	―	―

タイマーの精度 (2/2)

FUJITSU

- 主要なタイマールーチンの精度(タイマーのオーバーヘッド)

No.	ルーチン名	精度分解能	オーバーヘッド(μs)	スレッドセーフ性	実装	GCCオーバーヘッド(μs)	富士通/GCC	備考
12	TIME サービス関数	1/1,000,000	5.28	○	time	5.03	1.05	―
13	TIMEF サービス関数	1/1,000,000	7.26	○	time	―	―	―
14	TIMER サービスサブルーチン	1/1,000,000	49.85	○	gettimeofday localtime	―	―	―
15	CLOCK サービスサブルーチン	1/1,000,000	12.82	○	getrusage	―	―	―
16	CLOCKM サービスサブルーチン	1/1,000,000	13.70	○	getrusage	―	―	―
17	CLOCKV サービスサブルーチン	1/1,000,000	14.12	○	getrusage	―	―	
18	CPU_TIME 組込みサブルーチン	1/1,000,000	18.09	○	getrusage	5.39	3.36	―
19	DTIME サービス関数	1/1,000,000	14.12	○	getrusage	10.20	1.38	―
20	ETIME サービス関数	1/1,000,000	13.33	○	getrusage	9.37	1.42	―
21	SECOND サービス関数	1/1,000,000	14.27	○	getrusage	5.91	2.41	―

FUJITSU

コンパイラ実行時ライブラリの主なエントリ名

- コンパイラ実行時ライブラリの主なエントリ名

コンパイラ実行時ライブラリの主なエントリ名

FUJITSU

エントリ名	機能概略
__jwe_c	コンパイラから呼び出されるモジュール（チェックルーチン、Fortran多相型処理など）
__jwe_ca	COARRAY
__jwe_d	組み込みデバッグ、quickデバッグ
__jwe_e	実行時ライブラリのエラー処理
__jwe_f	実行時ライブラリ内で呼び出す演算、型変換などの内部ルーチン
__jwe_hook	HOOK機能
__jwe_i	Fortran IO処理
__jwe_o	並列処理（主に、OpenMP）
__jwe_p	並列処理（並列制御、自動並列）
__jwe_s	Fortranサービスルーチンの処理モジュール
__jwe_t	実行時情報出力機能
__jwe_x	実行時ライブラリ制御（初期化・終了、例外、領域管理等）
__mpc__	C/C++（tradモード）OpenMP
__f_	Fortran関数（変形関数、配列関数、問い合わせ関数等）
__g_	浮動小数点数値演算系関数
__plvla	mfunc=1
__vm_	mfunc=2
__v_	mfunc=3
omp_	OpenMPルーチン
kmp_	LLVM OpenMP

演算区間経過時間のバラつきについて

- 演算区間経過時間のバラつきの要因
  - ノード内OSジッタによる遅延
  - L2キャッシュアクセスレイテンシの差による遅延
  - ディレクトリパス名変更による性能への影響

演算区間経過時間のバラつきの要因

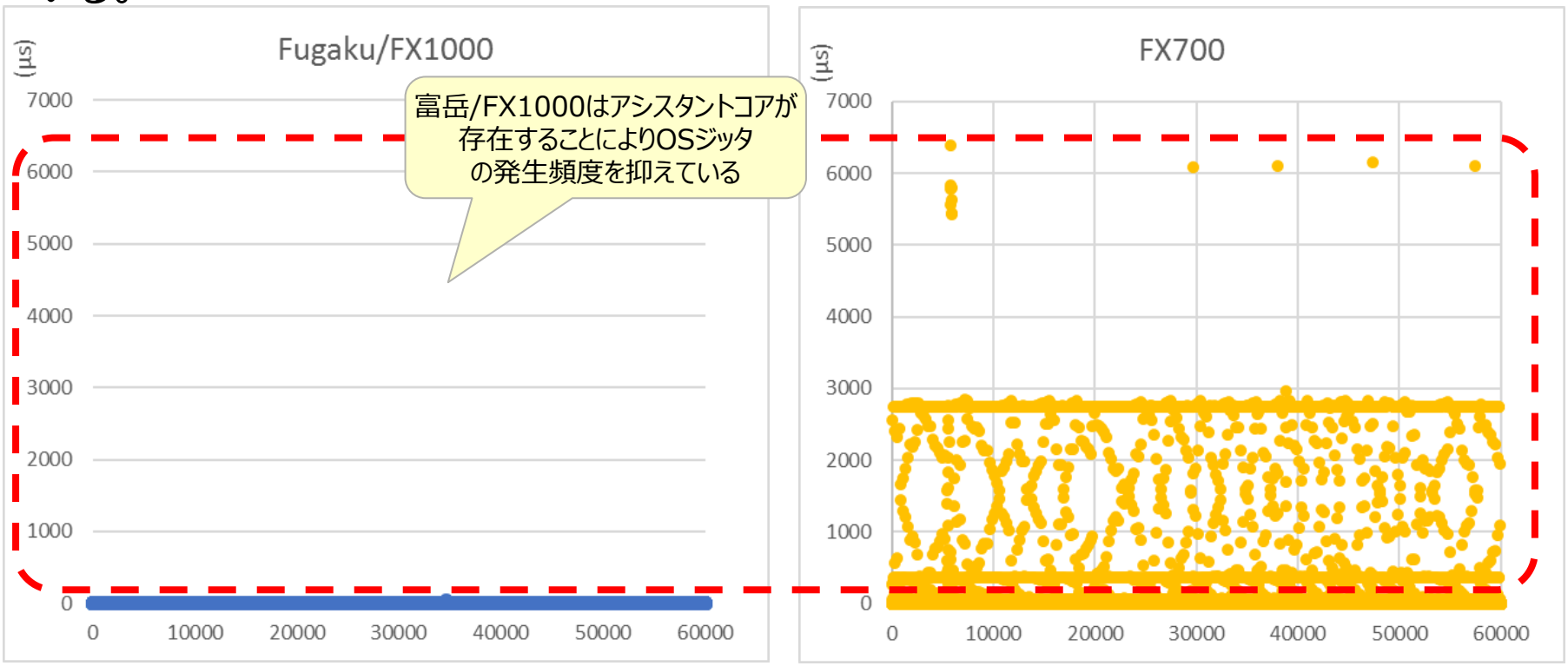
FUJITSU

- 同じ演算処理をしているにも関わらず経過時間に差が生じるケースがある。要因としては以下の可能性がある。
  - ① ノード内OSジッタによる遅延
  - ② 他CMGメモリの使用
    → 使用するメモリがCMG間を跨いで使用されている可能性がある。
    主に動的ライブラリの使用によるものがある
  - ③ アクセス配列のアドレス(物理アドレス)の差異
  - ④ OoO動作による実行タイミングブレ
    → 演算パイプラインa,bの偏り、ロード・ストアのパイプライン制御、等
  - ⑤ L2キャッシュアクセスレイテンシの差
  - ⑥ ディレクトリパス名変更による性能への影響

①、⑤、⑥については、次ページ以降で説明する

ノード内OSジッタの影響

- 富岳/FX1000ではCPUにアシスタントコアが搭載されている。
  そのため、FX700に比べOSジッタの発生頻度が低くなる。
- 以下はノード内48コア(48スレッド)実行で、FWQの実行を行い集計した結果となっている。



L2キャッシュアクセスレイテンシの差による遅延

FUJITSU

- L1Dレイテンシの採取
  CPU内全コアで実行(12スレッド×4MPI実行)し、fappコマンドを使用しパフォーマンスイベントを採取、下の計算式から各コアのL1Dキャッシュ・ミス処理の平均レイテンシを算出した。
- レイテンシ計算式

  L1D キャッシュ・ミス処理の平均レイテンシ
  = L1_MISS_WAIT / L1D_CACHE_REFILL

  ※ A64FX マイクロアーキテクチャマニュアルより
- 検証コード

```
39                !$omp parallel
                <<< Loop-information Start >>>
                <<<  [OPTIMIZATION]
                <<<    PREFETCH(HARD) Expected by compiler :
                <<<      x2, x1, y
                <<< Loop-information  End >>>
40    1                 DO j = 1, iter
41    1             !$omp do
                <<< Loop-information Start >>>
                <<<  [OPTIMIZATION]
                <<<    SIMD(VL: 8)
                <<<    SOFTWARE PIPELINING(IPC: 3.25, ITR: 144, MVE: 4,
POL: S)
                <<<    PREFETCH(HARD) Expected by compiler :
                <<<      x2, x1, y
                <<< Loop-information  End >>>
42    2  p  2v               DO i = 1, n
43    2  p  2v                 y(i)=x1(i) + c0 * x2(i)
44    2  p  2v               ENDDO
45    1             !$omp end do nowait
46    1                 ENDDO
47                !$omp end parallel
```

iter=12000

- 実行結果

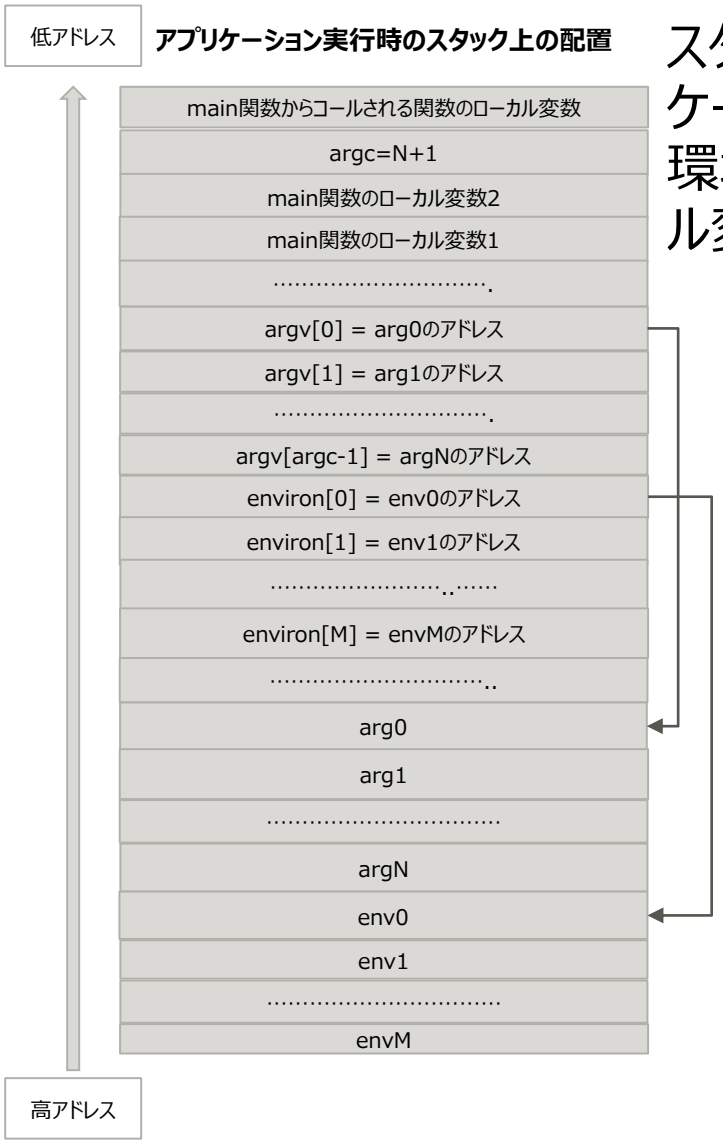
評価環境は
プロセス1が遅延する環境

プロセス0	プロセス1
コア12 : 60.3 コア13 : 61.4 コア14 : 61.4 コア15 : 60.2 コア16 : 61.8 コア17 : 59.6 コア18 : 61.9 コア19 : 60.7 コア20 : 63.1 コア21 : 62.8 コア22 : 63.3 コア23 : 66.3	コア24 : 67.2 コア25 : 50.2 コア26 : 68.6 コア27 : 67.8 コア28 : 50.3 コア29 : 68.5 コア30 : 53.8 コア31 : 68.8 コア32 : 54.9 コア33 : 69.7 コア34 : 59.3 コア35 : 71.0
プロセス2	**プロセス3**
コア36 : 60.3 コア37 : 61.5 コア38 : 61.5 コア39 : 60.2 コア40 : 61.8 コア41 : 59.6 コア42 : 61.9 コア43 : 60.7 コア44 : 63.1 コア45 : 62.8 コア46 : 63.3 コア47 : 66.3	コア48 : 60.3 コア49 : 61.4 コア50 : 61.4 コア51 : 60.2 コア52 : 61.8 コア53 : 59.6 コア54 : 61.8 コア55 : 60.7 コア56 : 63.1 コア57 : 62.7 コア58 : 63.3 コア59 : 66.3

他のプロセスの
最遅コアに比べ
約7%遅延

コア毎にレイテンシ差異があることが判る

ディレクトリパス名変更による性能への影響(1/3)



スタックは高アドレスから低アドレスに向かって使われ、アプリケーション実行時のスタック上の配置は、左図に示すように環境変数(env1～envM)、引数(arg1～argN)、ローカル変数の順になります。

● 性能への影響

ロードモジュールを配置するディレクトリパス名の変更やロードモジュール実行時に環境変数を追加すると、ロードモジュール実行時の性能に影響がある可能性があります。

● 性能凸凹の原因

ディレクトリパス名の変更やロードモジュール実行時の環境変数の追加により、関数内に定義されたローカル変数(スタック領域上に配置される)の配置アドレスがずれるためです。

ディレクトリパス名変更による性能への影響(2/3)

FUJITSU

● スタック上の配置アドレスがずれる要因

各種シェルにより、設定される環境変数に違いがありますが、/bin/bashを例に説明します。

特殊な環境変数として、以下の3つがあります。

PWD	カレントワーキングディレクトリ名
OLDPWD	cdコマンドで移動する前のカレントワーキングディレクトリ名
_	ロードモジュールのパス名

スタック上の配置アドレスがずれる要因としては、上記の３つの環境変数に関連して、以下の6つがあります。

1. ロードモジュールを実行する場合にフルパス名または、相対パス名とした場合、フルパス名または、相対パス名がそのまま、環境変数"_"に設定されます。
2. ロードモジュールを実行時にロードモジュールのファイル名だけ指定した場合、ロードモジュールのパス名は、環境変数**PATH**により補完されますが、この場合は、フルパス名が環境変数"_"に設定されます。
3. cdコマンドでディレクトリを移動すると、移動前のディレクトリ名が環境変数**OLDPWD**に設定され、cdコマンドで移動後のディレクトリが環境変数**PWD**に設定されます。
4. このため、フルパス名でロードモジュールを指定した場合と、環境変数**PATH**でロードモジュールのフルパス名を補完した場合とで微妙に引数のアドレスにずれが生じます。この結果、main関数から呼ばれる関数のローカル変数のアドレスにもずれが生じます。
5. また、フルパス名の長さを変えると、環境変数"_"に設定されるロードモジュールのフルパス名の長さが変わるため、環境変数として積まれる文字列が長くなるため、main関数から呼ばれる関数のローカル変数のアドレスにもずれが生じます。
6. ロードモジュールの実行前に、cdコマンドでカレントディレクトリを変更すると、**OLDPWD**という特殊な環境変数が設定されるため、main関数から呼ばれる関数のローカル変数のアドレスにもずれが生じます。

ディレクトリパス名変更による性能への影響(3/3)

FUJITSU

- 実機上での動作確認結果

(1) root(/)直下にロードモジュールを配置した場合
(2) rootのホームディレクトリ(/root)直下にロードモジュールを配置した場合((1)に比べて、16バイトずれています)
(3) rootのロードモジュール配置ディレクトリ(/root/bin)にロードモジュールを配置し、PATH環境変数でフルパス名を補完した場合((1)に比べて、16バイトずれています)
(4) rootのホームディレクトリ(/root)直下のosディレクトリ(/root/os)にロードモジュールを配置し、/root/osに移動し、相対パスで実行した場合((1)に比べて、32バイトずれています)

(1) root(/)直下にロードモジュールを配置した場合

```
# /hellomodule4 1 2 3 4
hello, function call arg=function call argument address: 0x0000ffffffffea68
hello, function call local world: 0x0000ffffffffea90
hello, main argc=5: 0x0000ffffffffeaec
hello, main initialized function local world: 0x0000ffffffffeaf0
hello, main not initialized local world: 0x0000ffffffffeb30
hello, /hellomodule4: 0x0000ffffffffef61
hello, 1: 0x0000ffffffffef6f
hello, 2: 0x0000ffffffffef71
hello, 3: 0x0000ffffffffef73
hello, 4: 0x0000ffffffffef75
.....
hello, _=/hellomodule4: 0x0000ffffffffffda
```

(2) rootのホームディレクトリ(/root)直下にロードモジュールを配置した場合

```
# /root/hellomodule4 1 2 3 4
hello, function call arg=function call argument address: 0x0000ffffffffea58
hello, function call local world: 0x0000ffffffffea80
hello, main argc=5: 0x0000ffffffffeadc
hello, main initialized function local world: 0x0000ffffffffeae0
hello, main not initialized local world: 0x0000ffffffffeb20
hello, /root/hellomodule4: 0x0000ffffffffef52
hello, 1: 0x0000ffffffffef65
hello, 2: 0x0000ffffffffef67
hello, 3: 0x0000ffffffffef69
hello, 4: 0x0000ffffffffef6b
.....
hello, _=/root/hellomodule4: 0x0000ffffffffffd0
```

(3) rootのロードモジュール配置ディレクトリ(/root/bin)にロードモジュールを配置し、PATH環境変数でフルパス名を補完した場合

```
# hellomodule4 1 2 3 4
hello, function call arg=function call argument address: 0x0000ffffffffea58
hello, function call local world: 0x0000ffffffffea80
hello, main argc=5: 0x0000ffffffffeadc
hello, main initialized function local world: 0x0000ffffffffeae0
hello, main not initialized local world: 0x0000ffffffffeb20
hello, hellomodule4: 0x0000ffffffffef50
hello, 1: 0x0000ffffffffef5d
hello, 2: 0x0000ffffffffef5f
hello, 3: 0x0000ffffffffef61
hello, 4: 0x0000ffffffffef63
.....
hello, _=/root/bin/hellomodule4: 0x0000ffffffffffc8
```

(4) rootのホームディレクトリ(/root)直下のosディレクトリ(/root/os)にロードモジュールを配置し、/root/osに移動し、相対パスで実行した場合

```
# cd os
# ./hellomodule4 1 2 3 4
hello, function call arg=function call argument address: 0x0000ffffffffea48
hello, function call local world: 0x0000ffffffffea70
hello, main argc=5: 0x0000ffffffffeacc
hello, main initialized function local world: 0x0000ffffffffead0
hello, main not initialized local world: 0x0000ffffffffeb10
hello, ./hellomodule4: 0x0000ffffffffef4e
hello, 1: 0x0000ffffffffef5d
hello, 2: 0x0000ffffffffef5f
hello, 3: 0x0000ffffffffef61
hello, 4: 0x0000ffffffffef63
,.....
hello, _=./hellomodule4: 0x0000ffffffffffcb
hello, OLDPWD=/root: 0x0000ffffffffffdc
```

更新履歴

更新履歴

版数	発行月	変更理由及び内容
初版(1.0版)	2020/2	・新規作成
1.1版	2020/3	・記事見直しによる誤字の修正
1.2版	2020/9	・記事見直しによる誤字や表現の修正
1.3版	2021/3	・ソフトウェア版数アップによる差分修正、および、記事見直しによる誤字や表現の修正
1.4版	2021/8	・ソフトウェア版数アップによる差分修正、および、記事見直しによる誤字や表現の修正 ・「ラージページ不足時の動作(富岳のみ)」ページの追加 ・「LLVM OpenMPライブラリと富士通OpenMPライブラリ(2/2)」共有ライブラリ情報の追加
1.5版	2022/7	・記事の見直しによる誤字や表現の修正
1.6版	2023/3	・ソフトウェア版数アップによる差分修正、および、記事見直しによる誤字や表現の修正

FUJITSU