A64FXシステムアプリ性能検討WG

FUJITSU
shaping tomorrow with you

A64FXプログラミングガイド 入門編

2022年10月
富士通株式会社

はじめに

FUJITSU

- 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

目次

FUJITSU

shaping tomorrow with you

A64FXプログラミングガイド 入門編

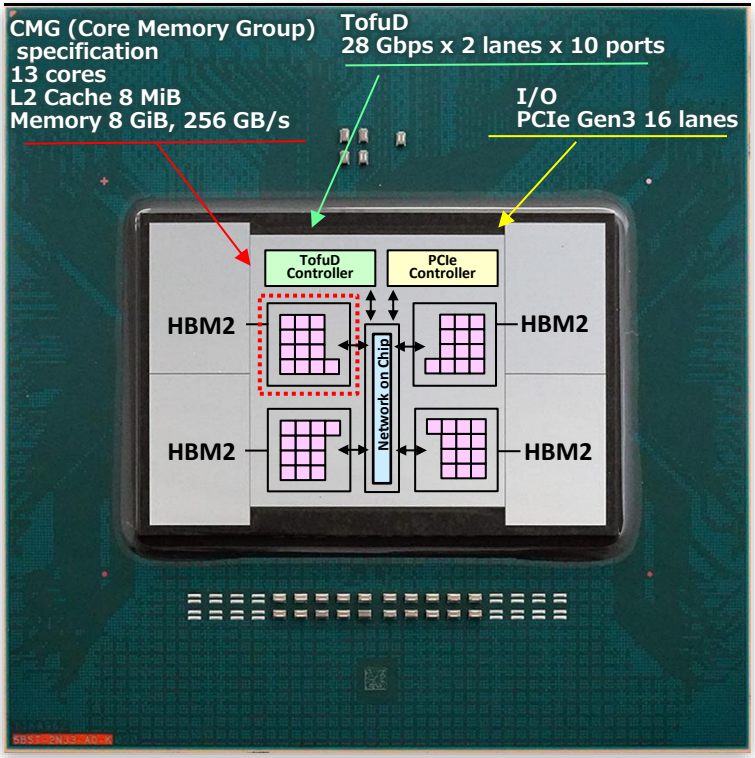
FUJITSU

A64FXプロセッサ概要

A64FXプロセッサ概要 (1/3)

FUJITSU

- Arm SVE を採用した高性能・高効率CPU
  - 倍精度演算性能 ： 3.072TFLOPS@2GHz , 90+%@DGEMM
  - メモリバンド幅 ： 1024 GB/s , 80+%@STREAM Triad



	A64FX
ISA (Base, extension)	Armv8.2-A, SVE
プロセステクノロジ	7 nm
倍精度ピーク性能	3.072TFLOPS@2GHz
SIMD幅	512-bit 256-bit/128-bit もサポート
コア数	48 + 4
メモリ容量	32 GiB (HBM2 x4)
メモリバンド幅	1024 GB/s
PCIe	Gen3 16 lanes
インターコネクト	TofuD(*) integrated

(*1) Tofu interconnect D

A64FXプロセッサ概要 (2/3)

FUJITSU

- A64FXプロセッサ (以下、A64FXと記述する) はHigh Performance Computing (HPC) 向けに設計され、ARMv8-A profile アーキテクチャ、および Scalable Vector Extension for ARMv8-A に準拠したアウト・オブ・オーダー実行型スーパースカラ・プロセッサである。
  A64FX はHPC向けにいくつかの特徴的なアーキテクチャを採用している。

  - Scalable Vector Extension

    A64FX は ARM命令セットアーキテクチャのベクトル拡張である Scalable Vector Extension (SVE) をサポートする。

  - Core Memory Group

    A64FXは13個のプロセッサ・コア、独立したL2キャッシュ、独立したメモリ・コントローラからなる Core Memory Group (CMG) と呼ばれるグループを持つ。
    プロセッサは4つのCMGを持ち、CMG間は Non-Uniform Memory Access (NUMA) 構成である。

  - セクタ・キャッシュ

    キャッシュを Way単位で仮想的に分割し、命令レベルで使用できる領域を指定できる機能である。プログラムはタグド・アドレスを使用することで領域を指定できる。
    L1キャッシュは4パーティション、L2キャッシュは2パーティションを2グループ持つ。

A64FXプロセッサ概要 (3/3)

FUJITSU

- ハードウェア・バリア

  ソフトウェアのプロセス、またはスレッド間の同期をハードウェアでサポートする機能である。
  この機能により、メモリ操作を行わずに高速に同期処理ができる。
  ハードウェア・バリア可能な範囲はCMG内であり、CMG間の同期処理は、ソフトウェア・バリアで行う。

- ハードウェア・プリフェッチ・アシスト

  ハードウェア・プリフェッチの振る舞いをプログラムから制御できる機能である。プログラムはシステムレジスタとタグド・アドレスを用いてハードウェアのプリフェッチ機構に情報を与えることができる。

- High Bandwidth Memory

  メインメモリにHigh Bandwidth Memory Gen2 (HBM2) を採用し、高いメモリ帯域を提供している。

FUJITSU

A64FXプロセッサ諸元

A64FXプロセッサ諸元

項目	諸元
プロセッサ・コア数	52 (13 cores / CMG)
CMG数	4
L1Iキャッシュ・サイズ	64KiB / 4way
L1Dキャッシュ・サイズ	64KiB / 4way
L2キャッシュ・サイズ	32MiB / 16way (8MiB / CMG)
キャッシュライン・サイズ	256B
メモリ・サイズ	32GiB(8GiB/CMG)
インターコネクト	Tofu Interconnect D
I/O	PCI-Express Gen3 16 Lanes
命令セット・アーキテクチャ	ARMv8-A, ARMv8.1, ARMv8.2, ARMv8.3 (*1), SVE

(*1) ARMv8.3 は Complex number support 命令のみサポートする。

A64FXプロセッサ諸元：バンド幅、レイテンシ

		A64FX	備考
周波数【GHz】		2.0	
CPU数/ノード		1	
演算コア数/ノード		48	
CMG数/ノード		4	
メモリ容量/ノード【GiB】		32	
キャッシュ容量	L1【KiB/コア】	64(命令)+64(データ)	
	L2【MiB/CMG】	8	(注意) アシスタントコア付きノードの場合、アプリケーション利用は7MiB/14way と考える。
キャッシュレイテンシ【cycle】	L1	5 (EX, short) 8 (FL, short) 11 (FL, long)	
	L2	37～47	平均42
キャッシュバンド幅【B/cycle/コア】	L1	ヒット時：128	
	L2	42.7	
ノードあたり演算性能 (1コアあたり)【GFlops】	倍精度 単精度 半精度	3072 (64) 6144 (128) 12288 (256)	
基本演算レイテンシ【cycle】	整数add命令 整数mult命令 FMA命令	1 5 9	
メモリレイテンシ【ns】		150	
ノードあたり理論メモリバンド幅 (CMGあたり)【GB/秒】		1024(256)	
CMG間バンド幅		128GB/s×2(双方向)	

A64FXプロセッサ諸元：その他

	A64FX	備考
最大デコード数　cycle当たり	4	
ハードウェアプリフェッチキュー	16	

FUJITSU

セクタキャッシュ

- セクタキャッシュとは
- セクタキャッシュの使用方法

セクタキャッシュとは

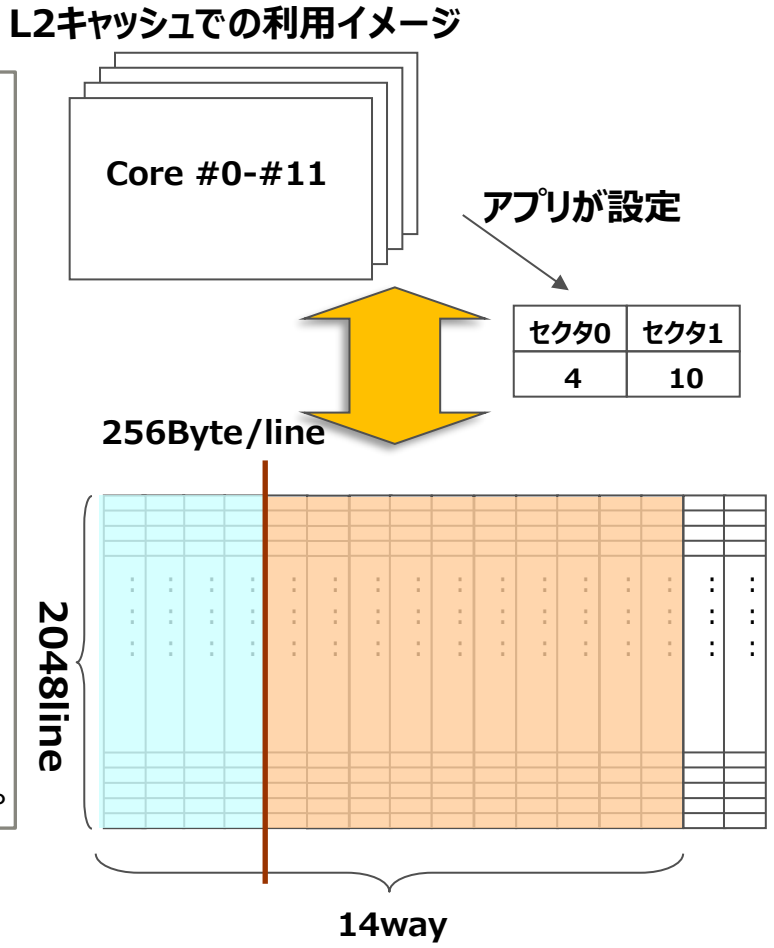
FUJITSU

セクタキャッシュ

セクタキャッシュとは、再利用性のあるデータが、再利用性の無いデータによってキャッシュから追い出されることを防ぐことができるキャッシュ機構です。アプリが再利用性のあるデータと再利用性の無いデータをセクタ毎に分けて配置することができます。（再利用する配列はセクタ1を使用し、その他はセクタ0を使用）

セクタキャッシュの詳細

- L1Dキャッシュ・L2キャッシュいずれにも複数個のセクタを設定できます。セクタの最大数はL1Dが4,L2は2です。
- 各セクタの容量はWAY数で指定します。
- 容量は目標値として働きます。
  ハードは、ラインリプレース時に各セクタが指定された容量に近づくように制御します。
  → 容量オーバーしていても強制的に無効化しない
- セクタ内の追い出しはLRU（最近最も使われていないデータを最初に捨てる）アルゴリズムで制御します。
- セクタ０、１の用途はアプリケーションで決めることが可能です。
  ただし、命令列はセクタ０に格納されます。
- 2次キャッシュはアシスタントコアが常時2wayを使用します。



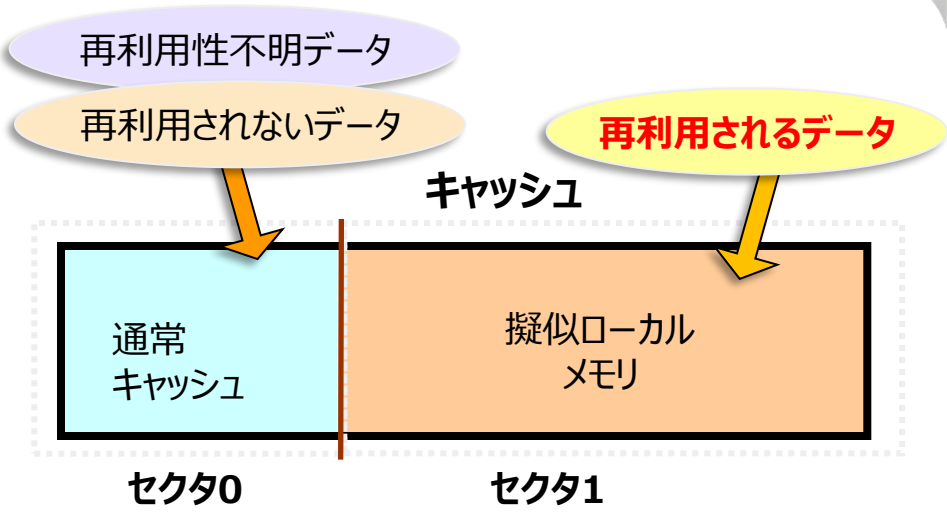
セクタキャッシュの使用方法（1/2）

FUJITSU

■ セクターキャッシュ：疑似ローカルメモリ

ソフトウェアが、データの再利用性に応じて
セクタを使い分けることが可能

- 再利用するデータ → セクタ1を使用
- その他のデータ → セクタ0を使用
- セクタ1上のデータは、他のデータによって
  追い出されることがない
- 指示行でセクタ1に載せる配列を指定できる



セクタキャッシュ指定のコンパイラ指示行の使用例

```
!OCL SCACHE_ISOLATE_WAY(L2=10)
!OCL SCACHE_ISOLATE_ASSIGN(a)
do j=1,m
   do i=1,n
      a(i) = a(i) + b(i,j)*c(i,j)
   enddo
enddo
!OCL END_SCACHE_ISOLATE_ASSIGN
!OCL END_SCACHE_ISOLATE_WAY
```

旧仕様の指定方法（京,FX100）

```
!OCL CACHE_SECTOR_SIZE(4,10)
!OCL CACHE_SUBSECTOR_ASSIGN(a)
do j=1,m
   do i=1,n
      a(i) = a(i) + b(i,j)*c(i,j)
   enddo
enddo
!OCL END_CACHE_SUBSECTOR
!OCL END_CACHE_SECTOR_SIZE
```

<狙い>
ループ中で配列 b と配列 c のアクセスによって
再利用性のある配列 a がキャッシュから追い出されないようにする

セクタキャッシュの使用方法（2/2）

FUJITSU

セクタキャッシュを使用する場合は、以下の最適化制御行を指定します。

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
SCACHE_ISOLATE_WAY(L2=n1[,L1=n2]) END_SCACHE_ISOLATE_WAY	1次キャッシュと2次キャッシュのセクタ1の最大way数を指示します。	○	×	○	×
SCACHE_ISOLATE_ASSIGN(array1[,array2]…) END_SCACHE_ISOLATE_ASSIGN	キャッシュのセクタ1に載せる配列を指示します。	○	×	○	×

■ 注意事項

- 2次キャッシュはアシスタントコアが常時2wayを使用します。そのため、n1およびn2に指定できる範囲は以下のようになります。
  0 ≦ *n1* ≦ “2次キャッシュの最大way数 - 2”
  0 ≦ *n2* ≦ “1次キャッシュの最大way数”

- アシスタントコアを含むCMGでは、L2キャッシュの一部(2way=1MiB分)をアシスタントコア用に使用します。そのため、アシスタントコアを含むCMGでは、2次キャッシュの最大way数は14, サイズは7MiBとなります。

A64FX諸元	
CMG数	4
L1Iキャッシュ・サイズ	64KiB / 4way
L1Dキャッシュ・サイズ	64KiB / 4way
L2キャッシュ・サイズ	32MiB / 16way (8MiB / CMG)

FUJITSU

高速ストア(zfill)

- 高速ストア(zfill)のイメージ
- 高速ストア(zfill)の動作条件

高速ストア(zfill)のイメージ

FUJITSU

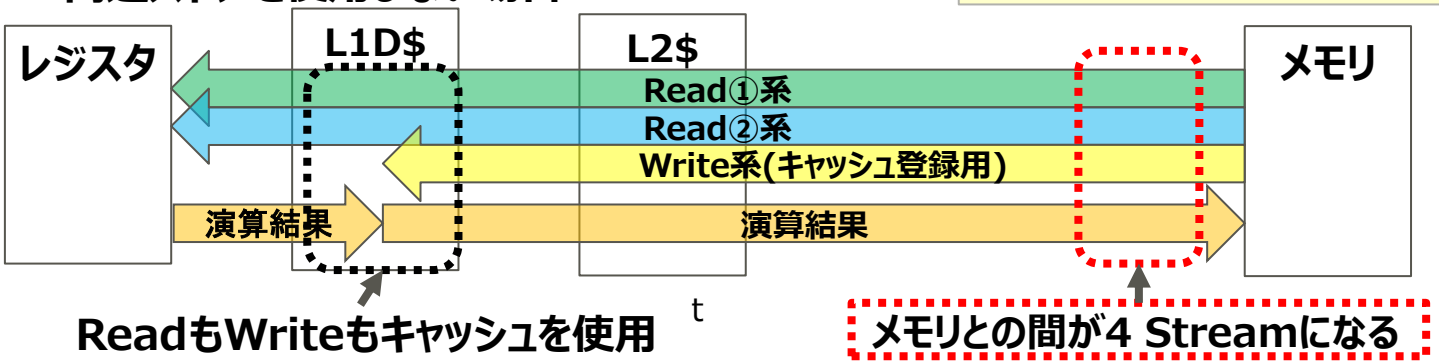
- 高速ストア（zfill）とは

  キャッシュ上に書き込み用のキャッシュラインを確保する機能です。これにより、メモリからのキャッシュラインの読み込みが削減できるので、メモリスループットがネックとなっているプログラムでは性能が改善します。

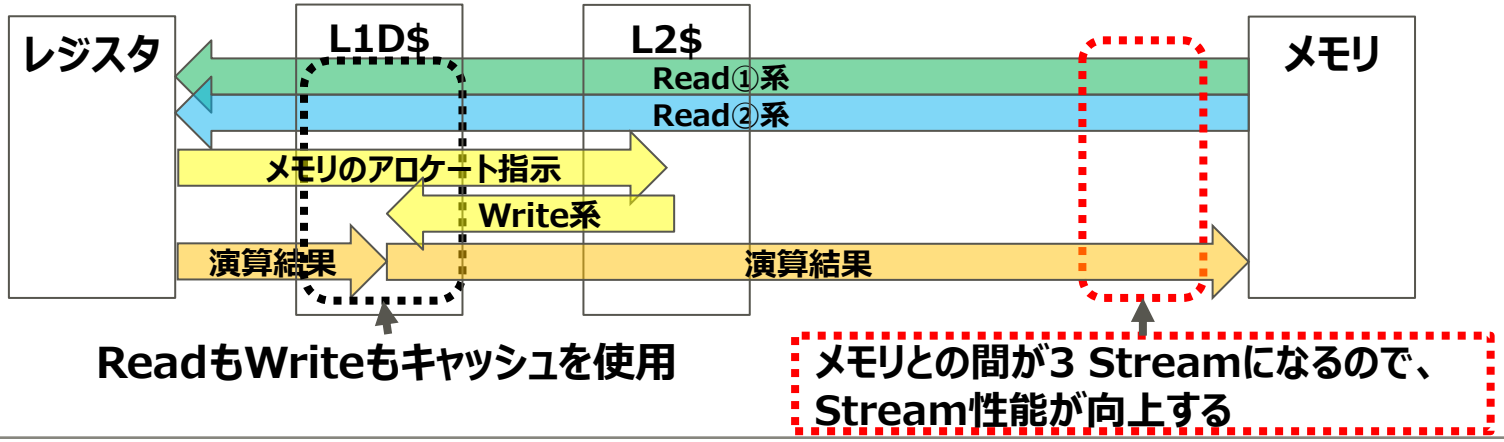
A64FXでは、DC ZVA命令を使用します。

- イメージ図 (Stream Triad ケースでのイメージ)

1．高速ストアを使用しない場合



2．高速ストアを使用した場合



高速ストア(zfill)の動作条件 (1/2)

FUJITSU

- 動作条件
  - ストア対象の配列がイタレーション間で依存が無いこと
  - 定義のある配列の参照が無いこと
  - 連続的なメモリアクセスであること
- 動作するケース

連続的なメモリアクセスのため zfill が動作する。

【Triad】

```
 real*4 y(n), x1(n), x2(n), c0
 do j = 1, iter
!$omp parallel do
   Do i = 1, n
     y(i)=x1(i) + c0 * x2(i)
   End Do
 enddo
```

iter = 3
n = 3145728

【ストリーム的演算カーネル pattern1】

```
do l = 1, lall
   !$OMP PARALLEL DO
   do k = 1, kall
   do g = 1, gall
      rho(g,k,l) = PROG(g,k,l,1) / metrics(g,k,l)
      vx (g,k,l) = PROG(g,k,l,2) / PROG(g,k,l,1)
      vy (g,k,l) = PROG(g,k,l,3) / PROG(g,k,l,1)
      vz (g,k,l) = PROG(g,k,l,4) / PROG(g,k,l,1)
      ein(g,k,l) = PROG(g,k,l,5) / PROG(g,k,l,1)
   enddo
   enddo
enddo
```

```
do iq = 1, qall
   do l  = 1, lall
   !$OMP PARALLEL DO
   do k  = 1, kall
   do g  = 1, gall
      q(g,k,l,iq) = PROGq(g,k,l,iq) / PROG(g,k,l,1)
   enddo
   enddo
   enddo
enddo
```

gall=16900、kall=96
lall=1、qall=6

高速ストア(zfill)の動作条件 (2/2)

FUJITSU

動作するケース

【ストリーム的演算カーネル pattern2】

```
do l = 1, lall
  !$OMP PARALLEL DO
  do k = 1, kall
  do g = 1, gall
    tendency(g,k,l,1) = tendency_0(g,k,l,1) + tendency_11(g,k,l) + tendency_21(g,k,l)
    tendency(g,k,l,2) = tendency_0(g,k,l,2) + tendency_12(g,k,l) + tendency_22(g,k,l)
    tendency(g,k,l,3) = tendency_0(g,k,l,3) + tendency_13(g,k,l) + tendency_23(g,k,l)
    tendency(g,k,l,4) = tendency_0(g,k,l,4) + tendency_14(g,k,l) + tendency_24(g,k,l)
    tendency(g,k,l,5) = tendency_0(g,k,l,5) + tendency_15(g,k,l) + tendency_25(g,k,l)
    tendency(g,k,l,6) = tendency_0(g,k,l,6) + tendency_16(g,k,l) + tendency_26(g,k,l)
  enddo
  enddo
enddo
```

zfill が動作できるコードだが、現在のコンパイラではストア対象の配列の依存が無いことを見切れない。改善の余地がある。
(ループ分割をすることで現在のコンパイラでもzfillを動作させることはできるため今回はループ分割し評価する)

動作しないケース

【ストリーム的演算カーネル pattern3】

```
do l = 1, lall
  !$OMP PARALLEL DO
  do k = 1, kall
  do g = 1, gall
    value(g,k,l,1) = value(g,k,l,1) + tendency(g,k,l,1) * fraction
    value(g,k,l,2) = value(g,k,l,2) + tendency(g,k,l,2) * fraction
    value(g,k,l,3) = value(g,k,l,3) + tendency(g,k,l,3) * fraction
    value(g,k,l,4) = value(g,k,l,4) + tendency(g,k,l,4) * fraction
    value(g,k,l,5) = value(g,k,l,5) + tendency(g,k,l,5) * fraction
    value(g,k,l,6) = value(g,k,l,6) + tendency(g,k,l,6) * fraction
  enddo
  enddo
enddo
```

定義のある配列の参照があるためzfill が動作しない

電力制御

- ブーストモードとは
- エコモードとは

ブーストモードとは

FUJITSU

- ブーストモードとは
  A64FXでは、性能と電力効率のバランスを考えて、ノーマルモードでは2.0GHzで動作させるが、アプリケーション（ジョブ）を少しでも速く動作させたい場合のことを考え、CPUの周波数を2.2GHzで動作させるブーストモードも用意した。ブーストモードでは、ノーマルモードより高い周波数で動作させるため、CPUの駆動電圧を昇圧する。
- ブーストモードでは、CPUの周波数は2.2GHzとなるが、HBMの周波数はノーマルモードと同じである。下表にブーストモードにおける性能向上が見込めるコード特性を示す。

コード特性	ブーストモードによる性能効果
L1バンド幅ネック	○
L2バンド幅ネック	○
メモリバンド幅ネック	×
演算ネック	○
アクセスレイテンシネック	○
Tofu通信バンド幅ネック	×
Tofu通信レイテンシネック	△　（CPU動作部分のみ）

エコモードとは (1/2)

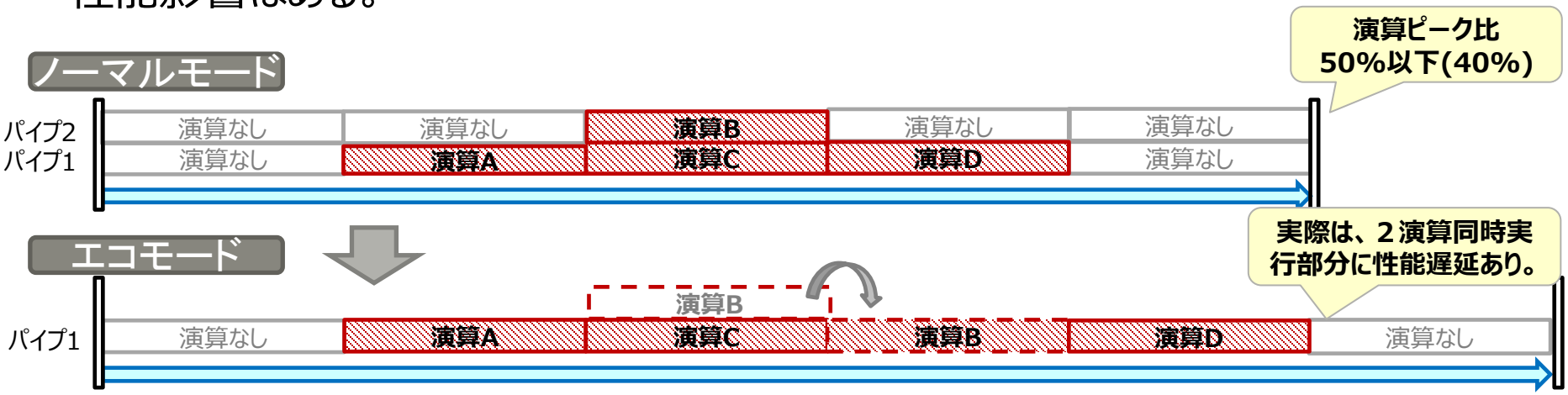
FUJITSU

- エコモードとは
  エコモードとは、FPU のパイプラインを1つのみ使用するパワーノブをONにするとともに、底上げ電力を浮動小数点演算1パイプ分のみと半減させるモードである。エコモードではピークの浮動小数点演算性能は低下するが、Run 状態およびStanby状態の電力削減に効果がある。

- ノーマルモードとエコモードの演算パイプラインの動き

  エコモードでは、演算パイプラインを１パイプ分のみと半減させるため、浮動小数点演算ピーク比、50%を超えるコードに関して確実に性能低下となる。

  浮動小数点ピーク比は50%を超えないケースでも、演算命令の動作タイミングにより性能影響はある。



エコモードとは (2/2)

FUJITSU

- 下表にエコモードにおける性能影響（低下）が見込まれるコード特性を示す。
- 電力に関しては、確実にノーマルモードよりもエコモードは削減される。

コード特性	エコモードによる性能影響
L1バンド幅ネック	なし
L2バンド幅ネック	なし
メモリバンド幅ネック	なし
演算ネック	あり
アクセスレイテンシネック	なし
ノード内バリアネック	なし
Tofu通信バンド幅ネック	なし
Tofu通信レイテンシネック	なし

FUJITSU

shaping tomorrow with you

A64FXプログラミングガイド 入門編

FUJITSU

プログラム開発環境概略

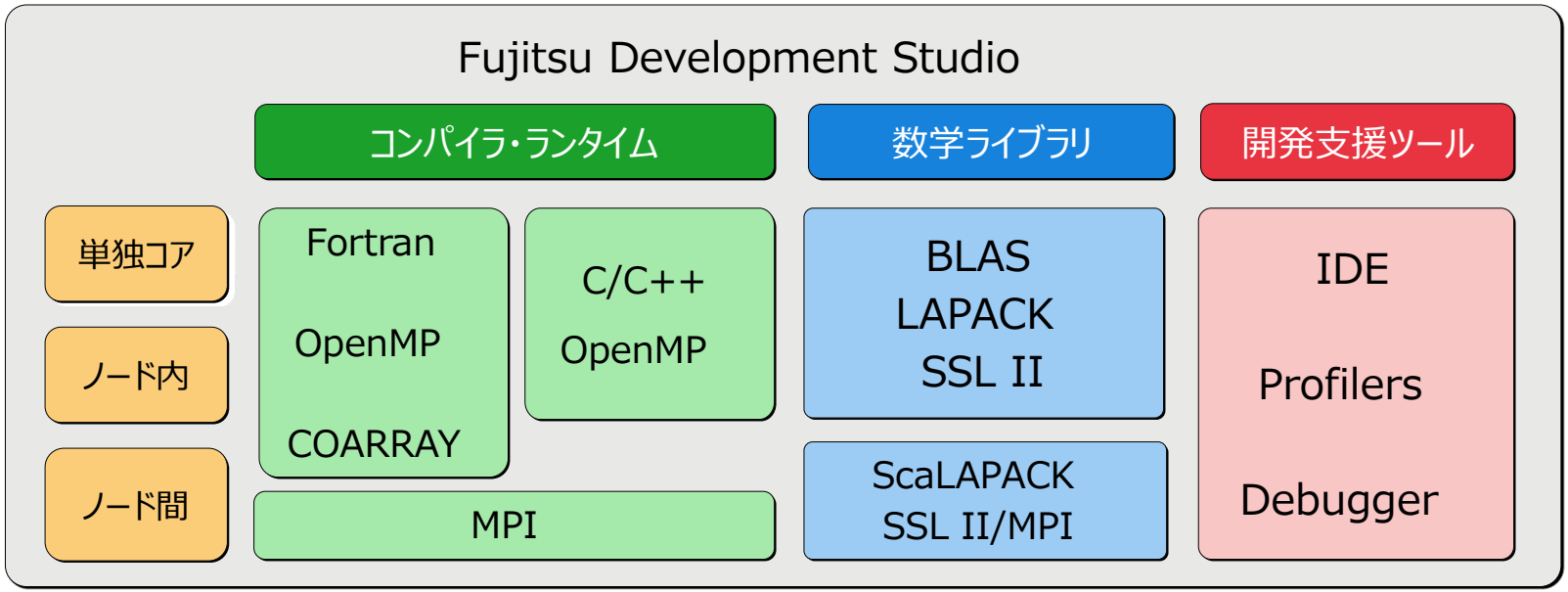
- プログラミング開発環境体系
- 言語仕様

プログラミング開発環境体系

FUJITSU

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

FUJITSU

- 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（仮称）の一部

FUJITSU

C/C++ tradモードと clangモード

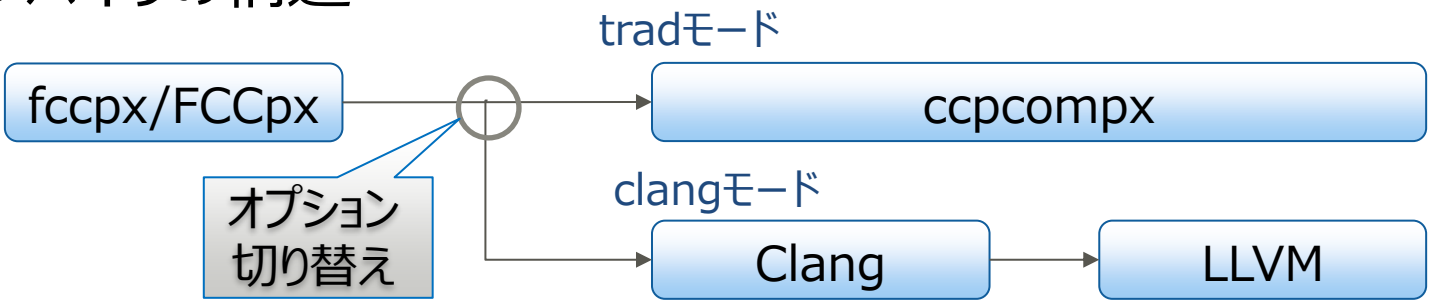
C/C++のtrad/clangモードの選択について

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



FUJITSU

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 (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で使われている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

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

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		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++	
				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。

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	–	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++	–	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	–	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++	–	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)

FUJITSU

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	−	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++	−	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

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
- 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）

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）

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）

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）

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）

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）

翻訳オプション	言語	意味 変更理由	旧仕様	新仕様
new -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）

翻訳オプション	言語	意味 変更理由	旧仕様	新仕様
new -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
new -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 new	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
new -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）

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} new	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

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
```

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

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	–
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	–
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	–
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	–
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	−
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	−
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

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)

FUJITSU

- 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)

FUJITSU

- 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)

FUJITSU

- 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
- 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

ラージページのページングポリシー

FUJITSU

- 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

FUJITSU

- 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

FUJITSU

CPU性能解析レポートを用いた性能チューニング

- CPU性能解析レポート：サイクルアカウンティングとは
- サイクルアカウンティングを活用したチューニング方法
- チューニングマップ：分類と状態
- CPU性能解析レポートの特徴
- CPU性能解析レポート 表示イメージ

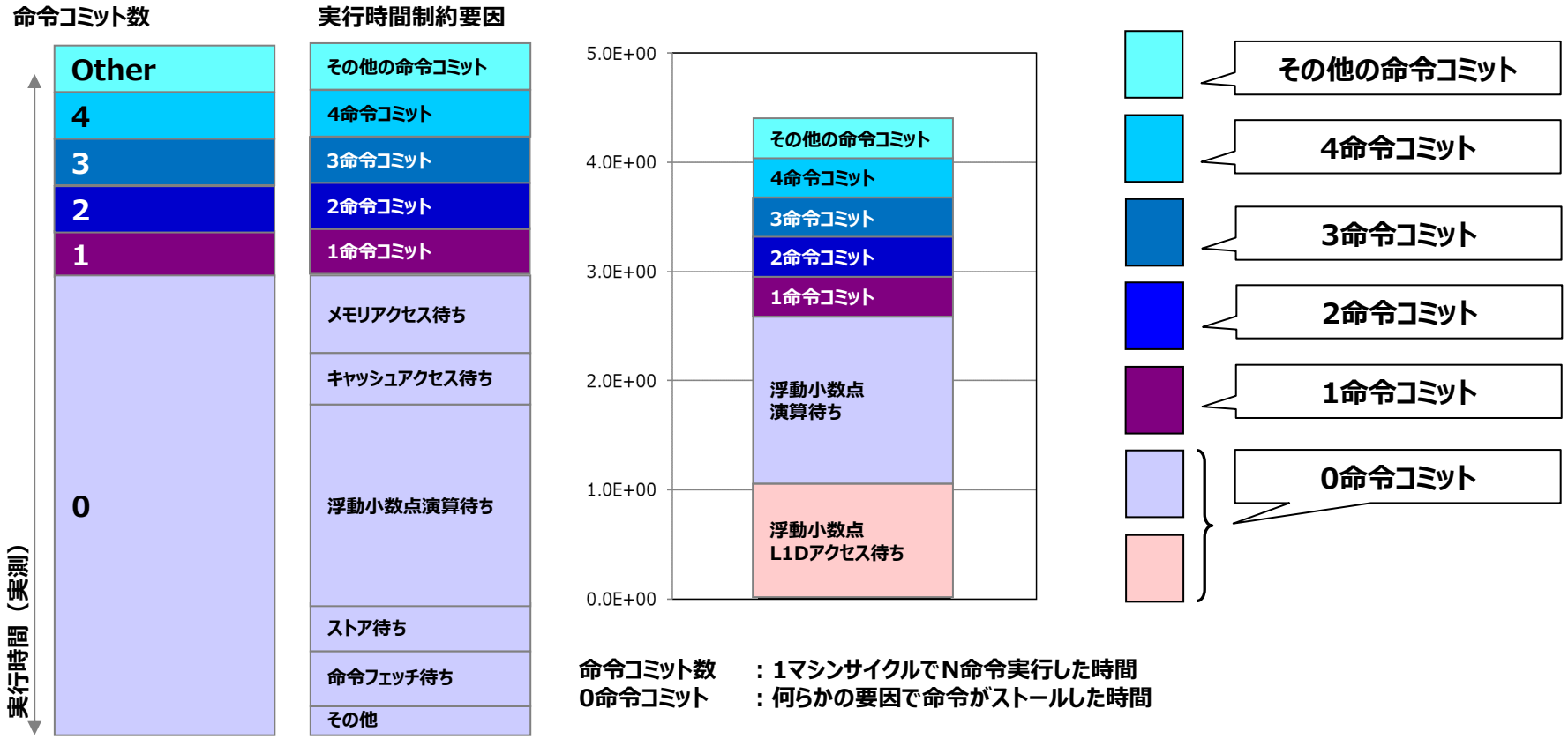
CPU性能解析レポート：サイクルアカウンティングとは

FUJITSU

サイクルアカウンティングとは、性能ボトルネックの要因分析手法です。

サイクルアカウンティング情報は、CPU性能解析レポートの右上部に掲載されています。

サイクルアカウンティングでは、あるアプリケーションプログラムを実行するためにかかった総時間（CPUサイクル数）をCPUの動作状態で分類してグラフ化します。そのグラフからCPU内のボトルネックが把握できるので、詳細な性能分析やチューニングを行うことができます。

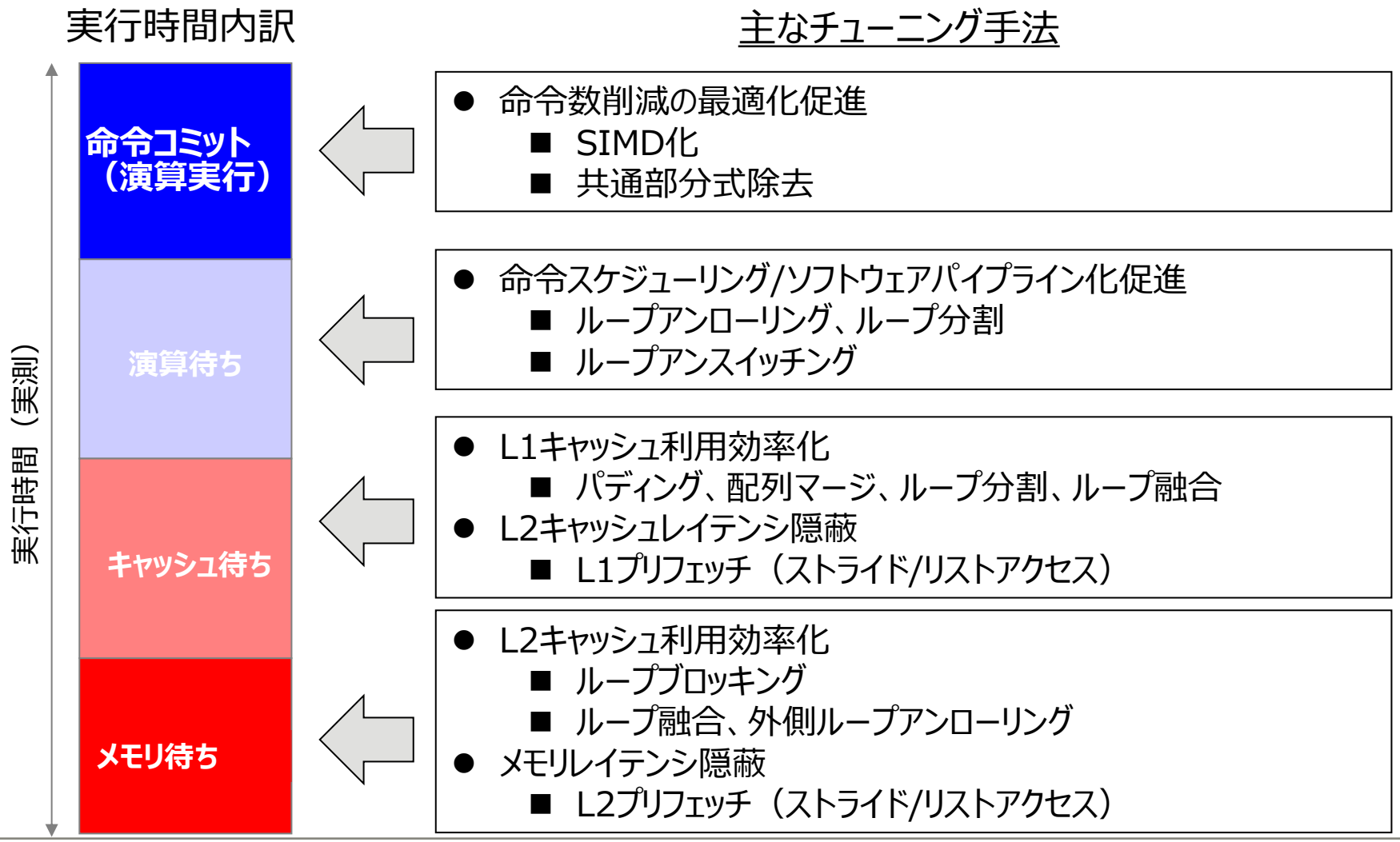


サイクルアカウンティングを活用したチューニング方法

FUJITSU

■ サイクルアカウンティングの結果から、チューニング手法を選択します。

次に主なチューニング手法を示します。
より詳細な内容はチューニングマップを参照してください。



チューニングマップ：分類と状態

ボトルネックの分類	PAグラフから見える高コスト	PA情報から見える高コスト	状態
メモリネック	浮動小数点ロードメモリアクセス待ち	-	メモリレイテンシがボトルネックになっています。
	整数ロードメモリアクセス待ち	-	
	ストア待ち	-	ストア命令のコストがボトルネックになっています。
	メモリ・キャッシュビジー待ち	-	メモリスループットがボトルネックになっています。
	-	メモリビジー率が高い	
	-	L2ミス率が高い L2ミスdm率が高い	メモリレイテンシがボトルネックになっています。
L2キャッシュネック	浮動小数点ロードL2アクセス待ち	-	L2キャッシュレイテンシがボトルネックになっています。
	整数ロードL2アクセス待ち	-	
	-	L2ビジー率が高い	L2キャッシュスループットがボトルネックになっています。
	-	L1Dミス率が高い L1Dミスdm率が高い	L2キャッシュレイテンシがボトルネックになっています。
L1キャッシュネック	浮動小数点ロードL1Dアクセス待ち	-	L1キャッシュレイテンシがボトルネックになっています。
	整数ロードL1Dアクセス待ち	-	
	-	L1ビジー率が高い	L1キャッシュスループットがボトルネックになっています。
スケジューリングネック	浮動小数点演算待ち	-	演算命令のレイテンシがボトルネックになっています。
	整数演算待ち	-	
	分岐命令待ち	-	分岐命令がボトルネックになっています。
並列化ネック	バリア同期待ち	-	スレッド並列化されていない部分がボトルネックになっています。
ロードインバランスネック	バリア同期待ち	命令バランスのmax-min差が大きい	スレッド間のロードインバランスがボトルネックになっています。
TLBネック	-	mDTLBミス率が高い	TLBミスやTLBスラッシングがボトルネックになっています。
	-	uDTLBミス率が高い	TLBミスがボトルネックになっています。
命令フェッチ	命令フェッチ待ち	-	命令キャッシュミスやスラッシングがボトルネックになっています。
命令数ネック	命令コミット：その他の命令コミット	-	命令数がボトルネックになっています。
	命令コミット：4命令コミット		
	命令コミット：3命令コミット		
	命令コミット：2命令コミット		
	命令コミット：1命令コミット		
その他	その他の待ち	-	PAが正しく採取できていない可能性があります。

CPU性能解析レポートの特徴

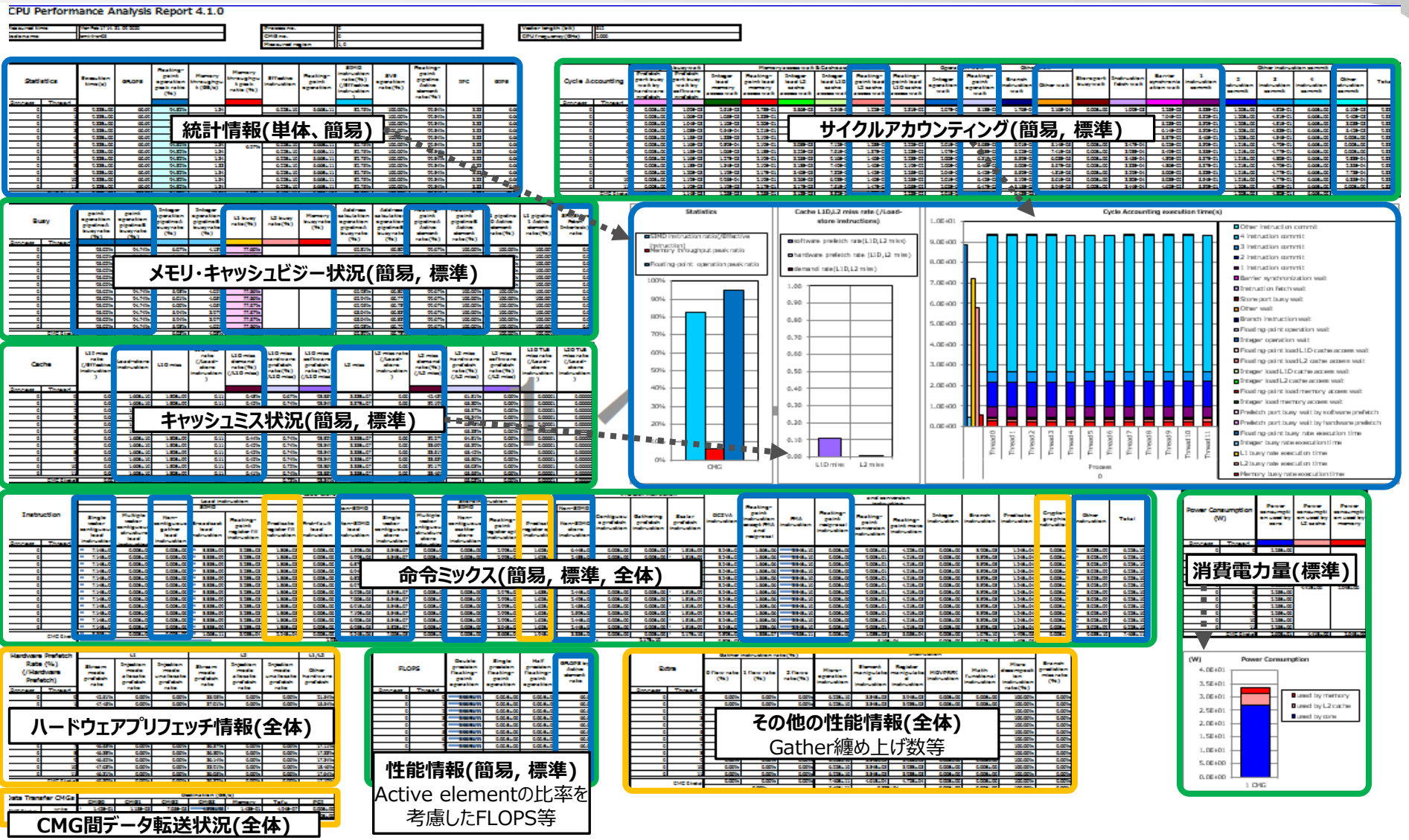
- PMUイベントに基づきCPUの性能情報をエクセルで表示
- 主要データは5回の計測でも採取した性能情報を表示可能
- 推奨は11回の計測で性能情報を表示(FX100相当)
- 追加計測することで、性能情報を詳細化可能

計測回数に対する採取可能な性能情報

性能情報のレベル	単体レポート	簡易レポート	標準レポート	詳細レポート
計測回数	1	5	11	17
統計情報	✓	✓※	←	←
サイクルアカウンティング		✓	✓※	←
メモリ・キャッシュビジー状況		✓	✓※	✓※
キャッシュミス状況		✓	✓※	←
命令ミックス		✓	✓※	✓※
負荷不均衡		✓	←	←
消費電力量			✓	←
ハードウェアプリフェッチ情報				✓
CMG間データ転送状況				✓
その他の性能情報				✓

(※) レベルが高いほど情報量が増加

CPU性能解析レポート 表示イメージ (全体)

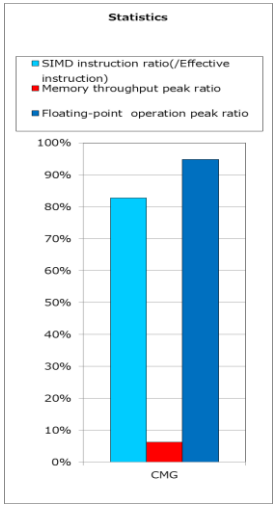


データ収集の回数によって、単体、簡易, 標準, 全体の順に表・グラフが詳細化します。
（下記のような各色の枠は実際のCPU性能解析レポートにはありません）

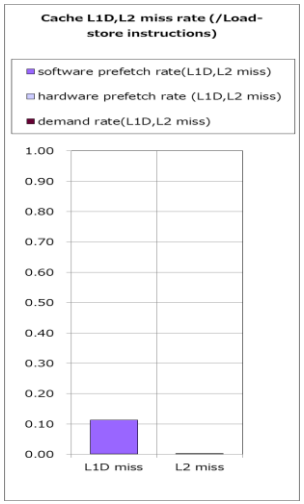
簡易　標準　詳細

CPU性能解析レポート 表示イメージ (グラフ)

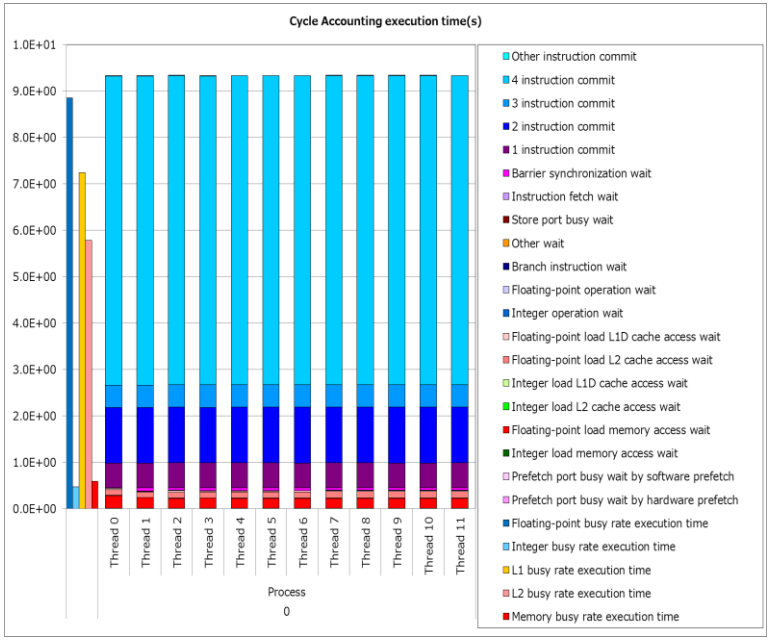
- 下記情報を見るだけで大まかな性能状態の把握が可能
  - 演算ピーク・スループット ピーク・SIMD比



  - キャッシュ状況



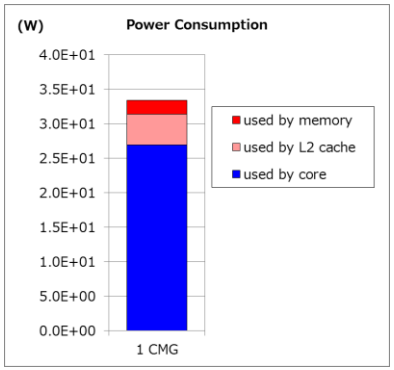
  - サイクルアカウンティング



  - CMG間データ転送状況

Data Transfer CMGs		Destination (GB/s)						
		CMG0	CMG1	CMG2	CMG3	Memory	Tofu	PCI
CMG 0 total	write	1.42E-01	1.15E-03	7.05E-05	4.89E+00	1.43E-01	4.94E-07	0.00E+00
	read					4.53E+00	5.49E-07	0.00E+00

  - 消費電力量



CPU性能解析レポート 表示イメージ (表1)

FUJITSU

統計情報

Statistics		Execution time (s)	GFLOPS	Floating-point operation peak ratio (%)	Memory throughput (GB/s)	Memory throughput peak ratio (%)	Effective instruction	Floating-point operation	SIMD instruction rate (%) (/Effective instruction)	SVE operation rate (%)	Floating-point pipeline Active element rate (%)	IPC	GIPS
Process	Thread												
0	0	9.33E+00	60.69	94.82%	1.34		6.2	5.66E+11	82.75%	100.00%	99.84%	3.33	6.66
0	1	9.33E+00	60.69	94.82%	1.36						99.84%	3.33	6.66
0	2	9.33E+00	60.69	94.82%	1.35						.84%	3.33	6.66
0	3	9.33E+00	60.69	94.82%	1.33						.84%	3.33	6.66
0	4	9.33E+00	60.69	94.82%	1.33						.84%	3.33	6.66
0	5	9.33E+00	60.69	94.82%	1.34						.84%	3.33	6.66
0	6	9.33E+00	60.69	94.82%	1.34	6					99.84%	3.33	6.66
0	7	9.33E+00	60.69	94.82%	1.34		6.22E+10	5.66E+11	82.75%	100.00%	99.84%	3.33	6.66
0	8	9.33E+00	60.69	94.82%	1.33		6.22E+10	5.66E+11	82.75%	100.00%	99.84%	3.33	6.66
0	9	9.33E+00	60.69	94.82%	1.34		6.22E+10	5.66E+11	82.75%	100.00%	99.84%	3.33	6.66
0	10	9.33E+00	60.69	94.82%	1.34		6.22E+10	5.66E+11	82.75%	100.00%	99.84%	3.33	6.66
0	11	9.33E+00	60.69	94.82%	1.34		6.22E+10	5.66E+11	82.75%	100.00%	99.84%	3.33	6.66
	CMG 0 total	9.33E+00	728.23	94.82%	16.06	6.27%	7.46E+11	6.79E+12	82.75%	100.00%	99.84%	3.33	79.93

実行時間、浮動小数点演算数、
メモリスループット、SIMD命令率など
主要な統計情報が表示される

サイクルアカウンティング

Cycle Accounting		Prefetch port busy wait		Memory access wait & Cache access wait						Operation wait		Other wait		Store port busy wait	Instruction fetch wait	Barrier synchronization wait	1 instruction commit	Other instruction commit				Total
		Prefetch port busy wait by hardware prefetch	Prefetch port busy wait by software prefetch	Integer load memory access wait	Floating-point load memory access wait	Integer load L2 cache access wait	Integer load L1D cache access wait	Floating-point load L2 cache access wait	Floating-point load L1D cache access wait	Integer operation wait	Floating-point operation wait	Branch instruction wait	Other wait					2 instruction commit	3 instruction commit	4 instruction commit	Other instruction commit	
Process	Thread																					
0	0	0.00E+00	1.09E-03	2.51E-03	2.78E-01	3.56E-03	2.34E-03	1.22E-01	2.31E-02	2.07E-03	5.18E-03	1.76E-04	2.16E-04	0.00E+00	1.09E-03	2.25E-03	5.38E-01	1.20E+00	4.82E-01	6.66E+00	6.10E-03	9.33E+00
0	1	0.00E+00	1.06E-03	1.05E-03	2.33E-01	2.65E-03	6.65E-04	1.20E-01	8.50E-03	1.88E-03	5.08E-03	5.14E-0		0.00E+00	3.78E-04	7.04E-02	5.32E-01	1.20E+00	4.81E-01	6.66E+00	9.40E-03	9.33E+00
0	2	0.00E+00	1.04E-03	1.10E-03	2.20E-01	2.93E-03	7.80E-04	1.22E-01	3.67E-02	2.11E-03	5.32E-03	5.60F			3.47E-04	5.23E-02	5.39E-01	1.20E+00	4.81E-01	6.66E+00	3.03E-03	9.33E+00
0	3	0.00E+00	1.08E-03	9.54E-04	2.21E-01	2.89E-03	7.74E-04	1.26E-01	2.25E-02	2.01E-0											E-03	9.33E+00
0	4	0.00E+00	1.15E-03	1.33E-03	2.19E-01	2.96E-03	7.15E-04	1.31E-01	2.22E-02	2.02E-											+00	9.33E+00
0	5	0.00E+00	1.16E-03	9.89E-04	2.19E-01	3.08E-03	7.12E-04	1.28E-01	2.22E-02	2.01E-											+00	9.33E+00
0	6	0.00E+00	1.15E-03	1.06E-03	2.18E-01	3.22E-03	7.51E-04	1.37E-01	2.23E-02	1.97E-											+00	9.33E+00
0	7	0.00E+00	1.16E-03	1.27E-03	2.19E-01	3.25E-03	9.16E-04	1.40E-01	2.23E-02	2.00E-											-04	9.33E+00
0	8	0.00E+00	1.18E-03	1.34E-03	2.19E-01	3.18E-03	7.49E-04	1.40E-01	2.19E-02	2.00E-											-04	9.33E+00
0	9	0.00E+00	1.20E-03	1.19E-03	2.17E-01	3.40E-03	7.32E-04	1.42E-01	2.25E-02	2.04E-											-04	9.33E+00
0	10	0.00E+00	1.19E-03	9.95E-04	2.19E-01	3.26E-03	6.93E-04	1.40E-01	2.25E-02	2.01E-0											E-04	9.33E+00
0	11	0.00E+00	1.19E-03	1.19E-03	2.17E-01	3.17E-03	7.51E-04	1.47E-01	2.05E-02	2.02E-03	6.47E-03	5.15E-05	5.94E-05	0.00E+00	3.44E-04	4.62E-02	5.39E-01	1.20E+00	4.80E-01	6.66E+00	0.00E+00	9.33E+00
	CMG 0 total	0.00E+00	1.14E-03	1.25E-03	2.25E-01	3.13E-03	8.82E-04	1.33E-01	2.23E-02	2.01E-03	5.93E-03	6.43E-05	7.01E-05	0.00E+00	4.14E-04	5.05E-02	5.37E-01	1.20E+00	4.80E-01	6.66E+00	2.22E-03	9.33E+00

メモリ・キャッシュビジー待ち時間、浮動小数点演算待ち時間、
バリア同期待ち時間、命令コミット時間など
積み上げグラフの数値(秒)が表示される

CPU性能解析レポート 表示イメージ (表2)

メモリ・キャッシュビジー情報

Busy		Floating-point operation pipeline A busy rate (%)	Floating-point operation pipeline B busy rate (%)	Integer operation pipeline A busy rate (%)	Integer operation pipeline B busy rate (%)	L1 busy rate (%)	L2 busy rate (%)	Memory busy rate (%)	Address calculation operation pipeline A busy rate (%)	Address calculation operation pipeline B busy rate (%)	Floating-point pipeline A Active element rate (%)	Floating-point pipeline B Active element rate (%)	L1 pipeline 0 Active element rate (%)	L1 pipeline 1 Active element rate (%)	SFI(Store Fetch Interlock) rate
Process	Thread														
0	0	95.02%	94.74%	6.07%	4.17				62.81%	60.80%	99.67%	100.00%	100.00%	100.00%	0.00
0	1	95.02%	94.74%												
0	2	95.02%	94.74%												
0	3	95.02%	94.74%												
0	4	95.02%	94.74%												
0	5	95.02%	94.74%												
0	6	95.02%	94.74%	5.98%	4.02%	77.56%	61.93%	6.27%	62.98%	60.80%					
0	7	95.02%	94.74%	6.01%	4.05%	77.56%			62.94%	60.77%					
0	8	95.02%	94.74%	6.00%	4.05%	77.57%			62.95%	60.78%					
0	9	95.02%	94.74%	5.94%	3.97%	77.57%			63.04%	60.83%					
0	10	95.02%	94.74%	5.94%	3.97%	77.57%			63.04%	60.83%					
0	11	95.02%	94.74%	5.98%	4.02%	77.56%			62.98%	60.79%					
CMG 0 total		95.02%	94.74%	6.03%	4.08%	77.56%	61.93%	6.27%	62.89%	60.78%					

L1ビジー率、L2ビジー率、メモリビジー率、
浮動小数点演算・整数演算パイプラインビジー率など

- メモリビジー率について
  - 京・FX100のPAシートから、分母としている数値が変更(1CMGあたりのメモリバンド幅理論性能値(256GB/s))となった。
  - 80%程度でビジーと考える

キャッシュミス状況

Cache		L1I miss rate (/Effective instruction)	Load-store instruction	L1D miss	L1D miss rate (/Load-store instruction)	L1D miss demand rate (%) (/L1D miss)	L1D miss hardware prefetch rate (%) (/L1D miss)	L1D miss software prefetch rate (%) (/L1D miss)	L2 miss	L2 miss rate (/Load-store instruction)	L2 miss demand rate (%) (/L2 miss)	L2 miss hardware prefetch rate (%) (/L2 miss)	L2 miss software prefetch rate (%) (/L2 miss)	L1D TLB miss rate (/Load-store instruction)	L2D TLB miss rate (/Load-store instruction)
Process	Thread														
0	0	0.00	1.60		0.11	0.48%	0.67%	98.85%	3.33E+07	0.00	42.43%	61.81%	0.00%	0.00001	0.00000
0	1								07	0.00	39.19%	65.30%	0.00%	0.00001	0.00000
0	2								07	0.00	38.98%	65.37%	0.00%	0.00001	0.00000
0	3								07	0.00	39.04%	65.34%	0.00%	0.00001	0.00000
0	4								07	0.00	38.74%	65.48%	0.00%	0.00001	0.00000
0	5	0.00	1.60E+10	1.80E+09	0.11	0.42%	0.74%	98.84%	3.35E+07	0.00	38.94%	65.38%	0.00%	0.00001	0.00000
0	6	0.00	1.60E+10	1.80E+09	0.11	0.44%	0.74%	98.82%	3.35E+07	0.00	39.27%	64.81%	0.00%	0.00001	0.00000
0	7	0.00	1.60E+10	1.80E+09	0.11	0.42%	0.74%	98.84%	3.35E+07	0.00	38.69%	65.39%	0.00%	0.00001	0.00000
0	8	0.00	1.60E+10	1.80E+09	0.11	0.42%	0.74%	98.84%	3.35E+07	0.00	38.51%	65.42%	0.00%	0.00001	0.00000
0	9	0.00	1.60E+10	1.80E+09	0.11	0.42%	0.74%	98.84%	3.35E+07	0.00	38.53%	65.50%	0.00%	0.00001	0.00000
0	10	0.00	1.60E+10	1.80E+09	0.11	0.42%	0.72%	98.86%	3.35E+07	0.00	39.17%	65.03%	0.00%	0.00001	0.00000
0	11	0.00	1.60E+10	1.80E+09	0.11	0.41%	0.74%	98.85%	3.35E+07	0.00	38.46%	65.55%	0.00%	0.00001	0.00000
CMG 0 total		0.00	1.93E+11	2.16E+10	0.11	0.43%	0.73%	98.84%	4.02E+08	0.00	39.16%	65.03%	0.00%	0.00001	0.00000

ロード・ストア数、L1Dミス率、L2Dミス率、DTLBミス率
L2ミス内訳（dmミス率、hwpfミス率、swpfミス率）など

CPU性能解析レポート 表示イメージ (表3)

命令ミックス

Instruction		Load-store instruction																Prefetch instruction				Floating-point instruction			Floating-point move and conversion instruction						
		Load instruction								Store instruction																					
		SIMD							Non-SIMD	SIMD					Non-SIMD																
Process	Thread	Single vector contiguous load instruction	Multiple vector contiguous structure load instruction	Non-contiguous gather load instruction	Broadcast load instruction	Floating-point register fill instruction	Predicate register fill instruction	First-fault load instruction	Non-SIMD load instruction	Single vector contiguous store instruction	Multiple vector contiguous structure store instruction	Non-contiguous scatter store instruction	Floating-point register spill instruction	Predicate register spill instruction	Non-SIMD store instruction	Contiguous prefetch instruction	Gathering prefetch instruction	Scalar prefetch instruction	DCZVA instruction	Floating-point instruction except FMA and reciprocal	FMA instruction	Floating-point reciprocal instruction	Floating-point conversion instruction	Floating-point move instruction	Integer instruction	Branch instruction	Predicate instruction	Crypto-graphic instruction	Other instruction	Total
0	0	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	1.59E+06	5.84E+07	0.00E+00	0.00E+00	2.99E+03	1.62E+03		0.00E+00	0.00E+00	1.81E+09	8.24E+02	1.56E+06	3.54E+10	0.00E+00	9.00E+01	4.22E+03	0.00E+00	8.90E+08	1.24E+04	0.00E+00	8.03E+09	6.22E+10
0	1	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	6.99E+05	5.84E+07	0.00E+00	0.00E+00	2.99E+03	1.62E+03																
0	2	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	6.87E+05	5.84E+07	0.00E+00	0.00E+00																		
0	3	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	6.94E+05	5.84E+07	0.00E+00	0.00E+00																		
0	4	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	6.82E+05	5.84E+07	0.00E+00	0.00E+00																		
0	5	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	6.97E+05	5.84E+07	0.00E+00	0.00E+00																		
0	6	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	6.92E+05	5.84E+07	0.00E+00	0.00E+00																		
0	7	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	7.00E+05	5.84E+07	0.00E+00	0.00E+00																		
0	8	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	6.91E+05	5.84E+07	0.00E+00	0.00E+00																		
0	9	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	7.19E+05	5.84E+07	0.00E+00	0.00E+00	2.99E+03	1.62E+03	2.59E+05	0.00E+00	0.00E+00	1.81E+09	8.24E+02	1.56E+06	3.54E+10	0.00E+00	9.00E+01	4.21E+03	0.00E+00	8.89E+08	1.24E+04	0.00E+00	8.02E+09	6.22E+10
0	10	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	6.90E+05	5.84E+07	0.00E+00	0.00E+00	2.99E+03	1.62E+03	2.44E+05	0.00E+00	0.00E+00	1.81E+09	8.24E+02	1.56E+06	3.54E+10	0.00E+00	9.00E+01	4.21E+03	0.00E+00	8.89E+08	1.24E+04	0.00E+00	8.02E+09	6.22E+10
0	11	7.14E+09	0.00E+00	0.00E+00	8.85E+09	3.28E+03	1.86E+03	0.00E+00	6.95E+05	5.82E+07	0.00E+00	0.00E+00	3.04E+03	1.62E+03	2.44E+05	0.00E+00	0.00E+00	1.81E+09	8.24E+02	1.33E+06	3.54E+10	0.00E+00	9.00E+01	4.21E+03	0.00E+00	8.89E+08	1.24E+04	0.00E+00	8.02E+09	6.22E+10
CMG 0 total		8.56E+10	0.00E+00	0.00E+00	1.06E+11	3.93E+04	2.24E+04	0.00E+00	9.24E+06	7.00E+08	0.00E+00	0.00E+00	3.60E+04	1.94E+04	3.35E+06	0.00E+00	0.00E+00	2.17E+10	9.89E+03	1.85E+07	4.25E+11	0.00E+00	1.08E+03	5.05E+04	0.00E+00	1.07E+10	1.49E+05	0.00E+00	9.63E+10	7.46E+11

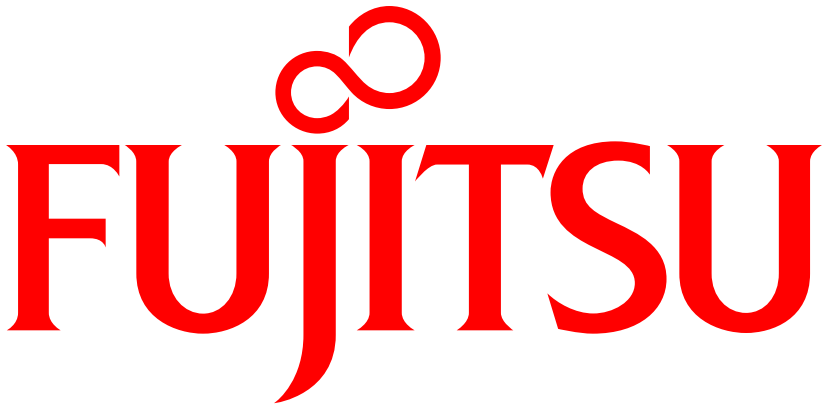
ロード・ストア命令、プリフェッチ命令、浮動小数点演算命令、整数演算命令、分岐命令 など

性能情報

FLOPS		Double precision floating-point operation	Single precision floating-point operation	Half precision floating-point operation	GFLOPS by Active element rate
Process	Thread				
0	0		0.00.E+00	0.00.E+00	60.59
0					
0					
0					
0	4	5.66.E+11	0.00.E+00	0.00.E+00	60.59
0	5	5.66.E+11	0.00.E+00	0.00.E+00	60.59
0	6	5.66.E+11	0.00.E+00	0.00.E+00	60.59
0	7	5.66.E+11	0.00.E+00	0.00.E+00	60.59
0	8	5.66.E+11	0.00.E+00	0.00.E+00	60.59
0	9	5.66.E+11	0.00.E+00	0.00.E+00	60.59
0	10	5.66.E+11	0.00.E+00	0.00.E+00	60.59
0	11	5.66.E+11	0.00.E+00	0.00.E+00	60.59
CMG 0 total		6.79.E+12	0.00.E+00	0.00.E+00	727.03

型別浮動小数点演算率(半精度・単精度・倍精度)、Predicate マスク情報を使ったGFLOPS値

- **型別浮動小数点演算率**
  測定区間で行われた浮動小数点演算のうち、各型(半精度、単精度、倍精度)を%で表示
- **Predicate マスク情報を使ったGFLOPS値**
  Predicateマスク情報を使用して算出したGFOPS値
  (Predicateマスク情報が演算のみの数値ではないため参考値)



shaping tomorrow with you