FUJITSU

shaping tomorrow with you

次世代スパコン「京（けい）」(*)の言語処理系と性能評価

＊ 理化学研究所様が2010年7月に決定、発表した「次世代スーパーコンピュータ」の愛称

2010年10月20日

富士通株式会社
次世代テクニカルコンピューティング開発本部
ソフトウェア開発統括部

林正和

アジェンダ

FUJITSU

- 次世代スーパーコンピュータ「京」の現状
- プログラミングモデルと富士通のHPC向けアーキテクチャ
- 次世代スーパーコンピュータ「京」世代の言語処理系
- SPARC64VIIIfxの性能 （現時点）
- まとめ

FUJITSU

次世代スーパーコンピュータ「京」の現状

理化学研究所　計算科学研究機構様　施設

建物外観

理化学研究所様　ご提供

理化学研究所計算科学研究機構様　施設内部

冷却水パイプ

計算機フロア (1F)

空調設備

理化学研究所様　ご提供

次世代スーパーコンピュータ「京」初出荷

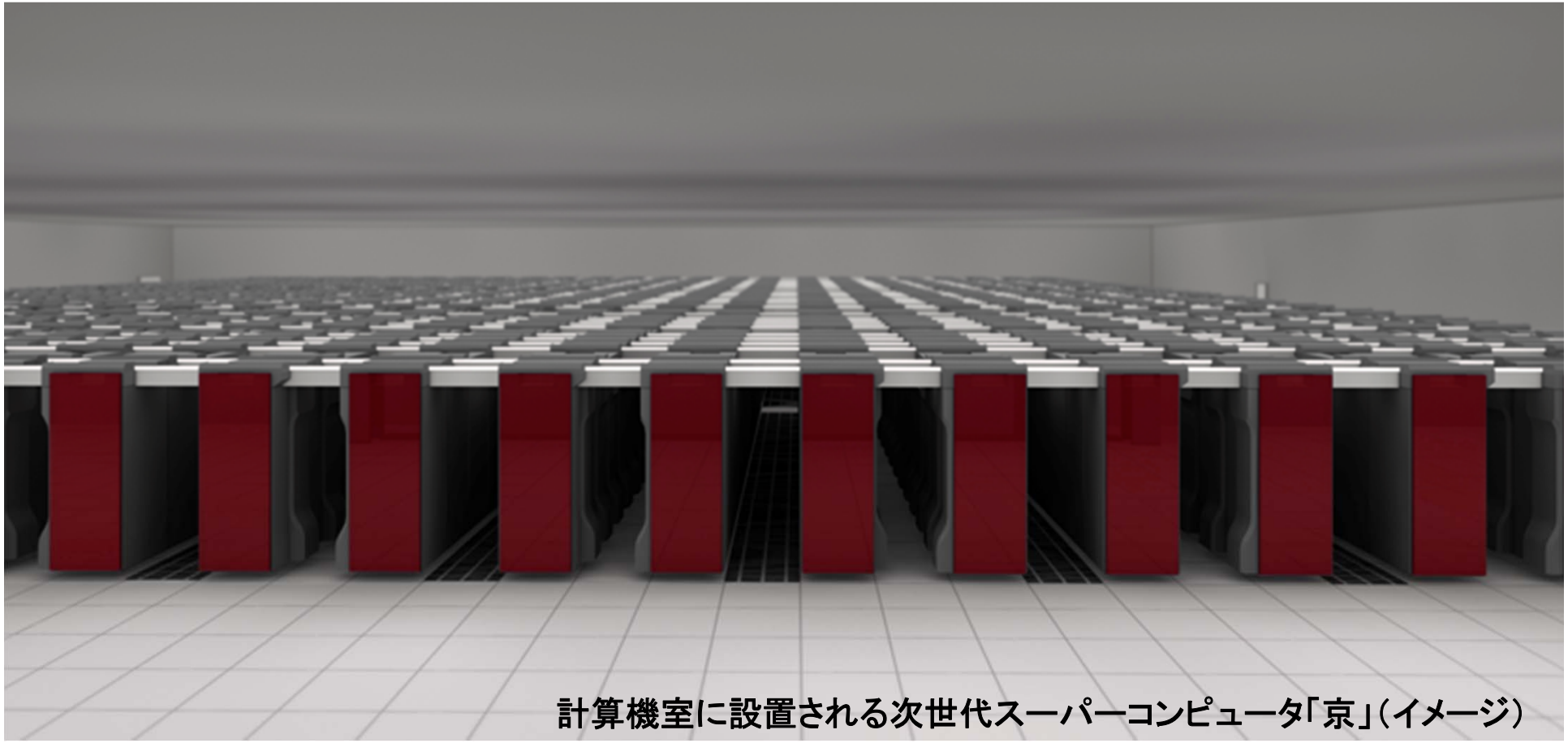
FUJITSU

■ 2010年9月28日　富士通ITプロダクツ（石川県かほく市）

理化学研究所 計算科学研究機構様 施設内部

FUJITSU

2012年の完成に向け
2010年9月29日より搬入開始



計算機室に設置される次世代スーパーコンピュータ「京」（イメージ）

FUJITSU

プログラミングモデルと富士通のHPC向けアーキテクチャ

HPCプログラミングモデルの課題

FUJITSU

- 背景
  - 動作周波数向上は頭打ち。
  - → コア数増加＋専用命令によって、性能を向上する。
    - → 数万～数十万コアの並列処理が必要。
- 課題：数万コア超の性能向上に限界

1）プロセス/スレッドのハイブリッド並列が必要

→ハイブリッド並列は敷居（プログラミング、チューニング）が高い。

2）数万超のネットワークが必要

→Fat Treeでの数万ノード構成は非現実的

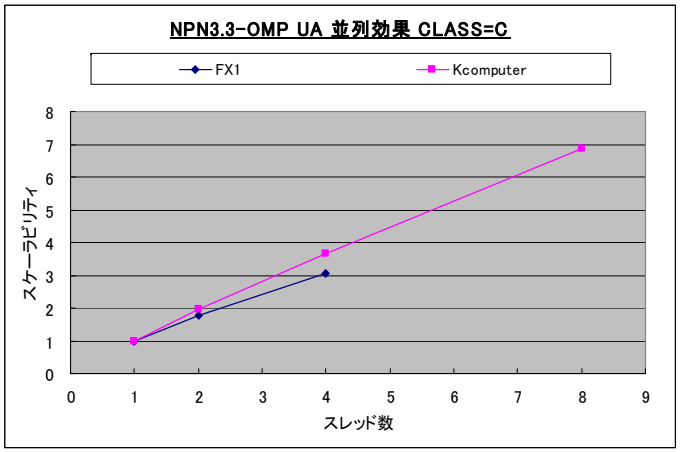
- 取り組み
  - ハイブリッド並列化を睨んだアーキテクチャ
  - HPCに向けてコアの強化（命令、バンド幅）
  - 新インターコネクト

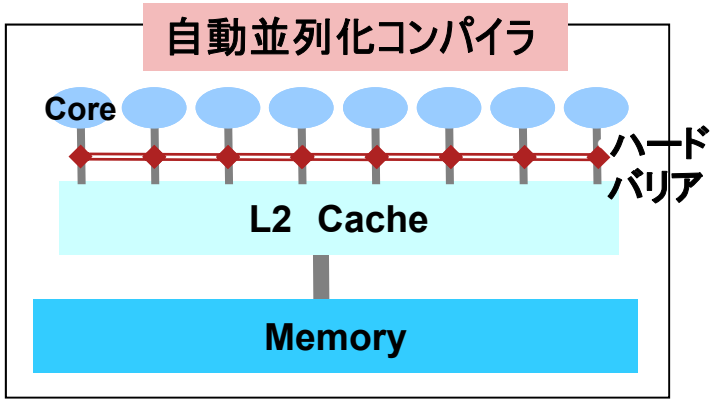
} 言語処理系の対応

ハイブリッド並列を睨んだアーキテクチャ

FUJITSU

- VISIMPACT
  - マルチコアを高速な1CPU化し、ハイブリッド並列を容易にする仕組み
  - ハード/ソフト技術が連携して実現
    - スレッド並列時に顕在化するフォルス シェアリングを抑止するコア間共有キャッシュ
    - スレッド並列時の制御オーバーヘッドの低減するハードバリア
    - 多くのプログラムを容易にスレッド並列化する自動並列化コンパイラ(自動ベクトル化を凌駕)
  - ベクトル型CPUが得意とする計算処理にも威力を発揮





弊社スパコン(FX1)で有効性を実証
→次世代スーパーコンピュータ「京」のベースアーキテクチャとして採用＆発展

次世代スーパーコンピュータ「京」に向けての機能

FUJITSU

コアの強化

理論性能のみではなくアプリケーションの実効性能を重視

- 汎用CPUをベースに、レジスタ数拡張、柔軟なSIMD演算器、ソフト制御可能なキャッシュ等の実行性能を高めるための機能を追加(HPC-ACE)
- 1CPU／1ノード構成により高メモリバンド幅を確保

  （メモリ　Peak　64GB/Sec　/ 実効　46.6GB/sec　[STREAM Triad性能])

新インターコネクト

6次元メッシュ/トーラス（ユーザービューは3次元トーラス）

- PCクラスタで使用されるFBB(*) のファットツリーでは、数万ノードの構成は非現実的
- 通常の3次元トーラスではできない高い運用性や対故障性を6次元メッシュ/トーラスでは実現可能（*: Full-Bisection Bandwidth）

集団通信アクセラレータ（Allreduce,バリア）

コアの強化（HPC-ACE詳細）

FUJITSU

(High Performance Computing - Arithmetic Computational Extensions)

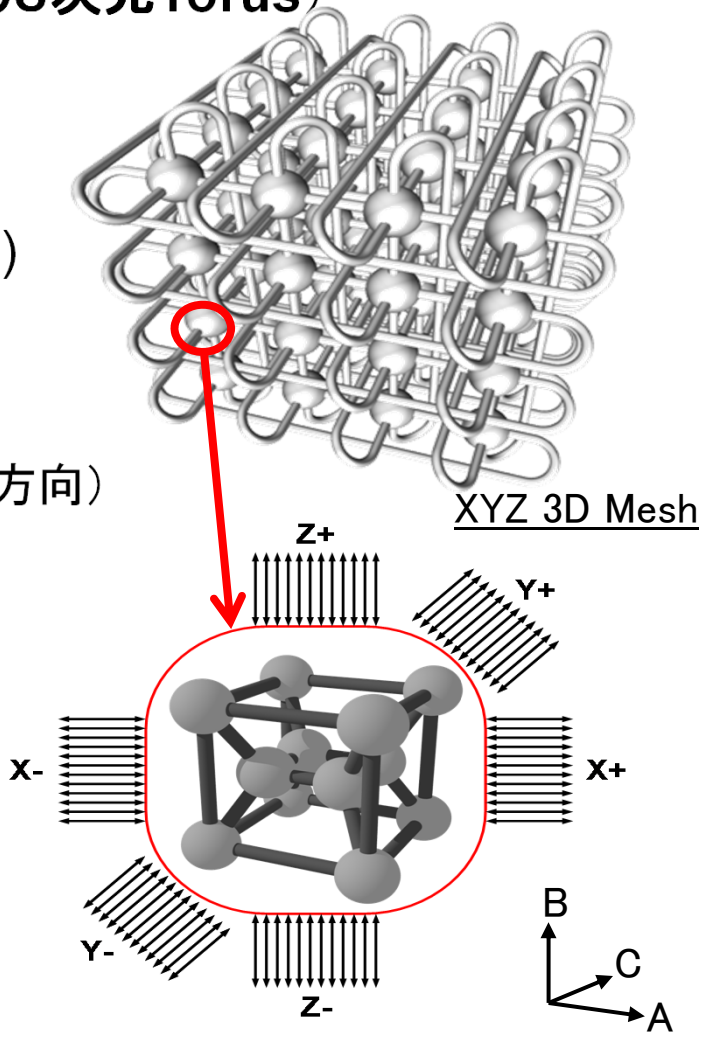
- SPARC64™ VIIIfxのISA (Instruction Set Architecture)
  - 準拠仕様
    - SPARC-V9 仕様
    - JPS (Joint Programmer's Specification): SPARC-V9拡張仕様
  - HPC-ACE: 富士通独自のHPC向け命令セット拡張
    - レジスタ拡張
    - セクターキャッシュ
    - SIMD (single instruction multiple data) 命令
    - マスク演算
    - 除算/平方根の逆数近似
    - 三角関数補助命令
    - 高機能prefetch/メモリアクセス制御機構

- 以下のSPARC64™ VIIIfx 関連文書は次のURLからダウンロードできます。
  http://jp.fujitsu.com/solutions/hpc/brochures/
  - The SPARC® Architecture Manual Version 9
  - SPARC® Joint Programming Specification (JPS1): Commonality
  - SPARC64™ VIIIfx Extensions

新インタコネクト

FUJITSU

- アーキテクチャ
  - 6次元 Mesh/Torus（アプリから見ると**階層なし3次元Torus**）
  - 外部スイッチのない直接網
- 特長
  - スケーラビリティ(80,000ノード超の規模に対応)
  - **耐故障性**(ノード故障時に運用継続可能)
  - 通信性能
    - ノード間100GB/s以上（リンク当たり5GB/s 双方向）
    - 4つの独立な通信エンジンによる高い実効スループット
    - 高速な**バリア・集合演算**



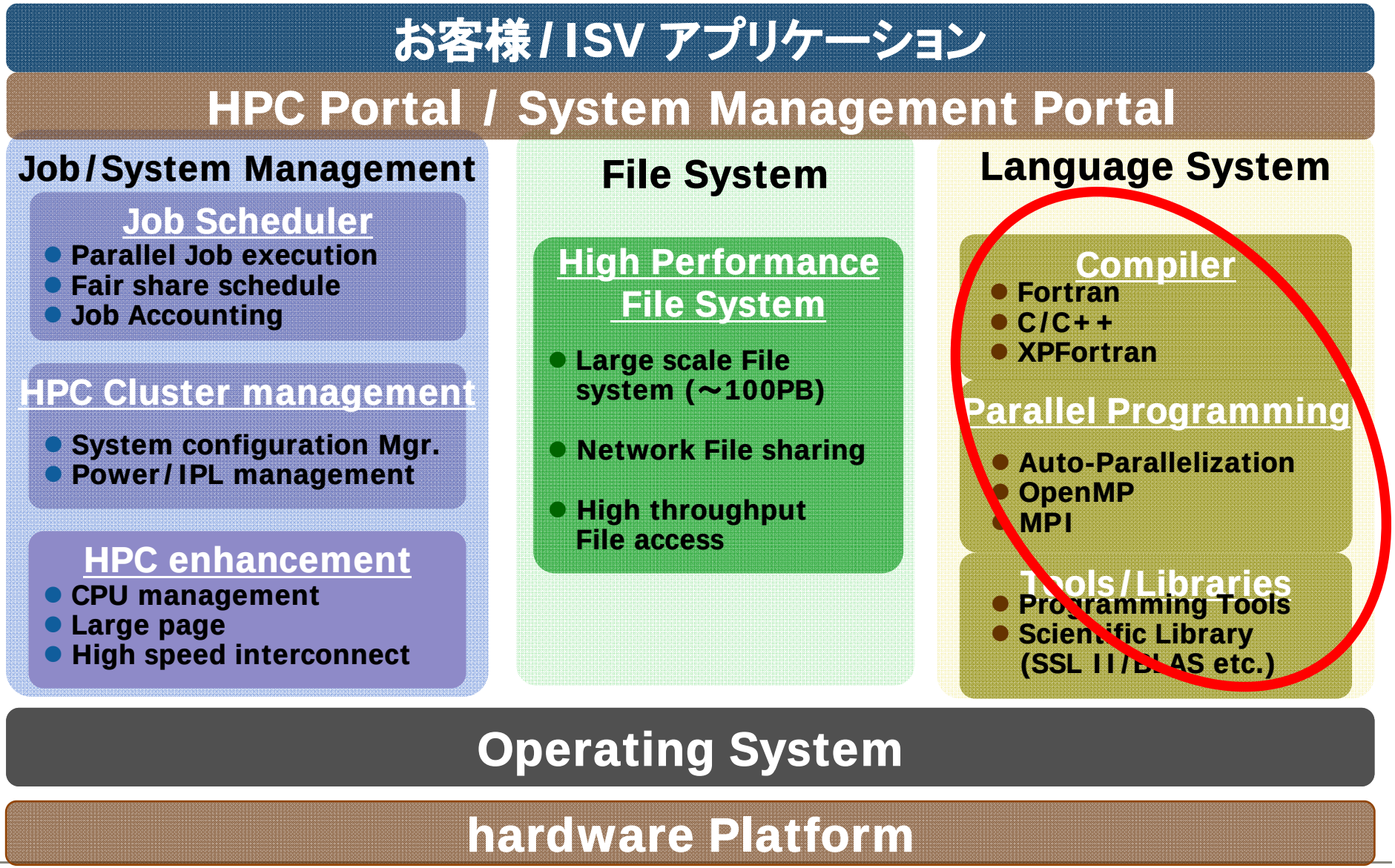
FUJITSU

次世代スーパーコンピュータ「京」の言語処理系について

注）言語処理系は開発中です。エンハンス予定の項目も入っています。

Fujitsu HPC ソフトウェアスタック

FUJITSU



プログラム開発環境の設計方針

FUJITSU

■超並列処理の実用化

➔ プロセス数を削減
⇒プロセス/スレッドのハイブリッド実行モデルを容易に記述

- コンパイラ:
  - OSSを翻訳できるデファクト言語仕様をサポート
  - ハイブリッド並列を容易に記述できる機能をサポート
    - マルチコアCPU(コア間共用キャッシュ)＋高速(ハードウェア)バリア
    - 上記CPUアーキテクチャを活かす自動並列化
  - HPC-ACEを活かす最適化
- ライブラリ
  - ネットワークの特徴を活かし，数万プロセス並列を実用化するMPI
  - システムにあわせてチューニングした数学ライブラリ
- 開発支援ソフトウェア
  - デバッグ機能
  - チューニング(プロファイラ)機能

言語仕様

FUJITSU

下記の標準規格、業界標準仕様をサポート

- 標準規格
  - Fortran: ISO/IEC 1539-1:2004（Fortran2003）
  - C :
    1999年規格 “JIS X 3010：2003”, “ISO/IEC 9899:1999”
    “JIS X 3010-1993”, “ISO/IEC 9899: 1990”（C89規格）
  - C++ :
    2003年規格 “JIS X 3014：2003”, “ISO/IEC 14882:2003”
- 業界標準仕様の実現
  - OpenMP Version3.0仕様
  - GNU C/C++拡張仕様
- 並列プログラミング
  - MPI-2.1
  - XPFortran

最適化機能エンハンス：HPC-ACEの利用

FUJITSU

- HPC-ACE機能を効果的に利用するコンパイラ最適化機能
  - SIMD命令の活用
    - 自動ベクトル化を応用したSIMD命令を自動生成
    - IF文を含むループのSIMD化（マスク付きSIMD化）
  - 拡張レジスタ（浮動小数点 256個，整数 64個）の利用
    - スピルコードの削減
    - 命令レベルの並列度の向上

実行命令数の削減。
ループ最適化の効果を高め、命令待ちを解消。

  - セクターキャッシュ（ソフトウェア制御可能なキャッシュメモリ）の利用
    - セクターキャッシュを使用者が意識して利用するためのディレクティブ
    - セクターキャッシュを考慮したプリフェッチ命令の自動生成

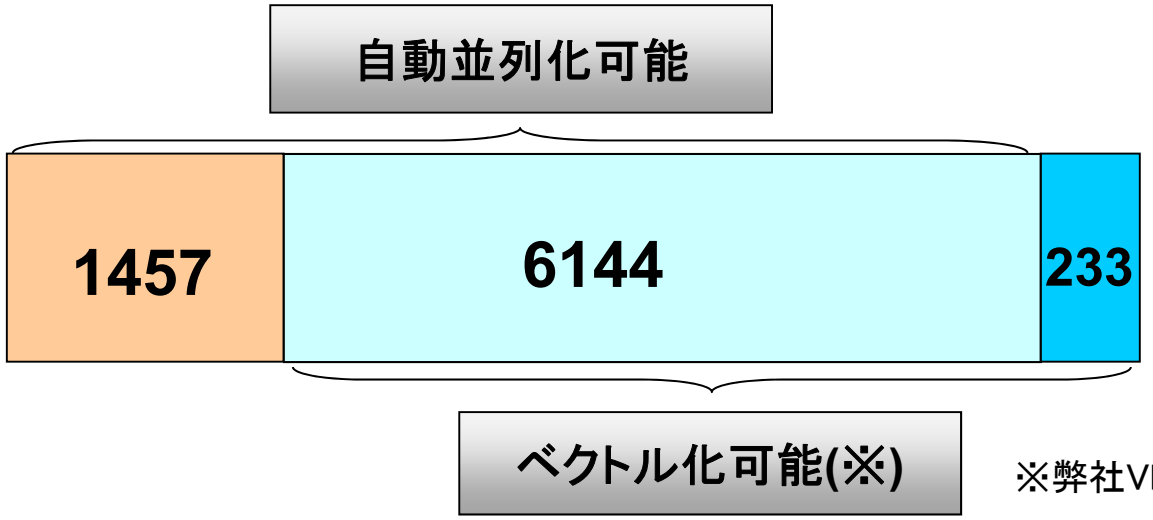
キャッシュミスの削減。

最適化機能エンハンス: 自動並列化強化

FUJITSU

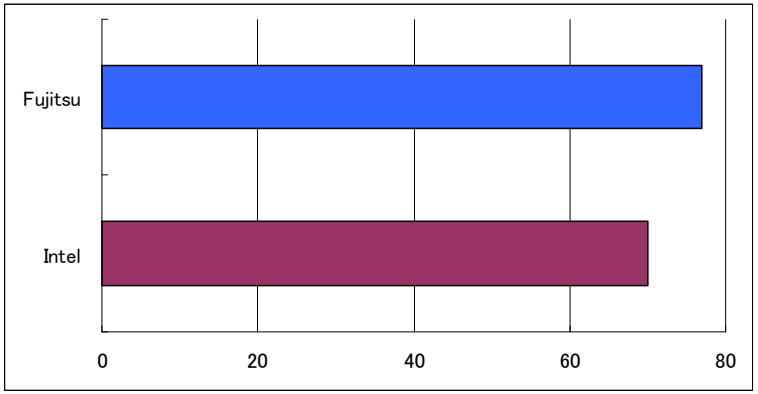
■ 自動並列化の強化（ベクトル凌駕/他社競合力強化）

実コードでの自動並列化およびベクトル化のループ数



自動並列は、高次元での変換が可能。
・最内ではベクトル化できないループを並列化できる。
↓
解析力の更なる強化を実施

※弊社VPPコンパイラでのベクトル化可能



PCクラスタ向けコンパイラでは、インテルコンパイラに対して10%並列化率が高い。(実コード)
（同じ最適化エンジン）
(Intel v11 vs Parallelnavi3.4)

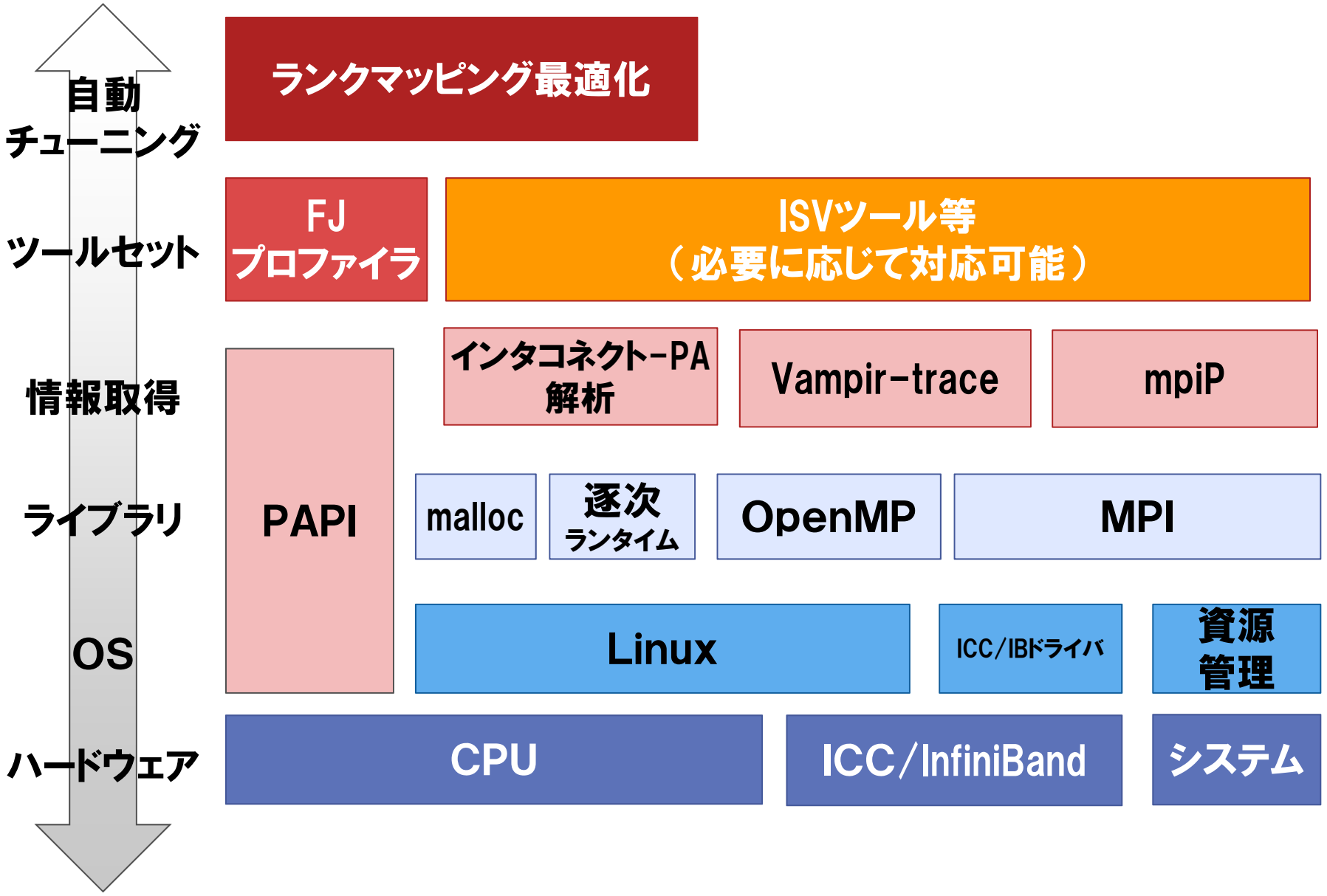
MPIライブラリ

FUJITSU

- 高性能を引き出す仕組み
  - 1対1通信
    - ソフトウェアの階層構造をバイパスする特別な低遅延経路設定
    - 新インタコネクトの性能を最大限に引き出せるように、送受信データの長さや配置に加え、ホップ数も考慮に加え、転送方式切替を最適化
  - 集団通信
    - 使用頻度高い関数(Bcast,Allgather,Allreduce,Alltoall等)について、1対1通信を利用せず，新インタコネクトの特徴を活かし，輻輳を抑える専用アルゴリズムを採用
    - 新インタコネクトの高機能バリア通信(ハード実装)を利用
  - PCクラスタにも共通技術は転用していく。

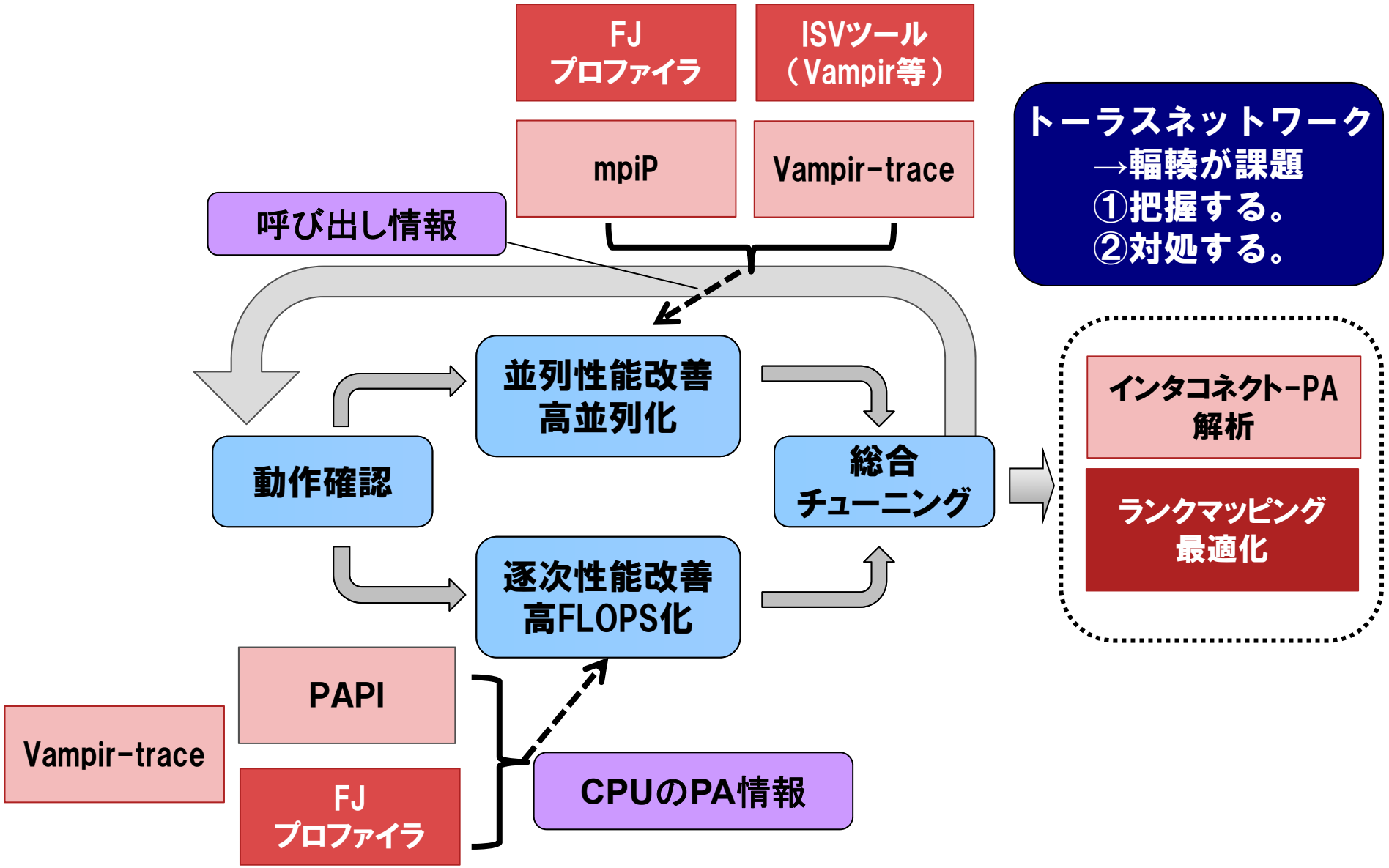
チューニングツールの構成（検討中含）

FUJITSU



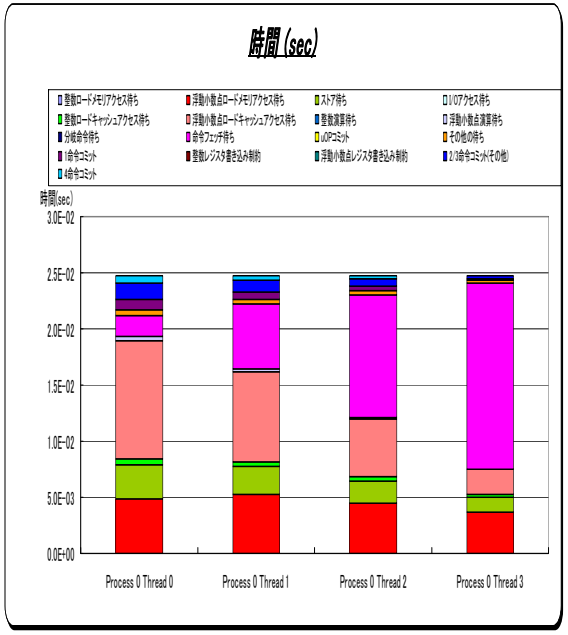
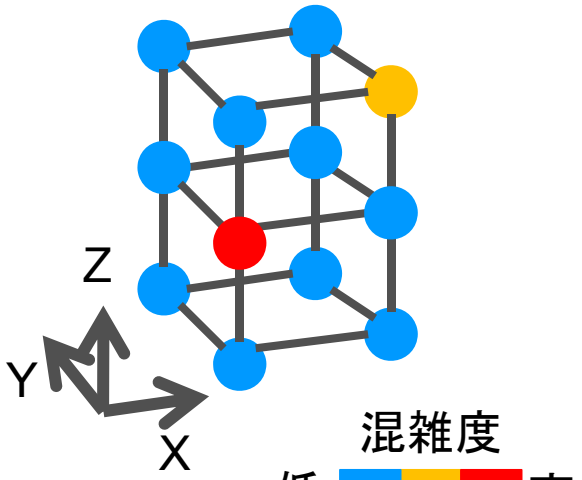
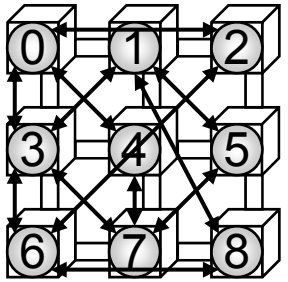
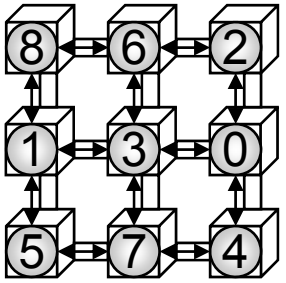
アプリケーション高性能化ステップとツール（検討中含）

FUJITSU



チューニング情報とランクマッピング

FUJITSU

CPU-PA	インタコネクト-PA	自動ランクマッピング
・アプリの実行の挙動把握 ■キャッシュスラッシング発見 ■スレッド並列のスレッドバランス確認,など	・通信混雑の発生により性能ボトルネックを見つける。 ■非効率な通信箇所発見 →まずは、「見える化」 →ノウハウを集める	・通信処理内容に応じて各ランクの実行ノードを適切に入れ替える。 ■通信処理時間短縮 ∑バイト長×ホップ数最小
	(イメージ図)  混雑度 低 高	 輻輳箇所:多 通信処理時間:長  輻輳箇所:少 通信処理時間:短

FUJITSU

次世代スーパーコンピュータ「京」の性能（現時点）

コンパイラは開発中のため、本日のデータは最終的なものではありません。

FUJITSU

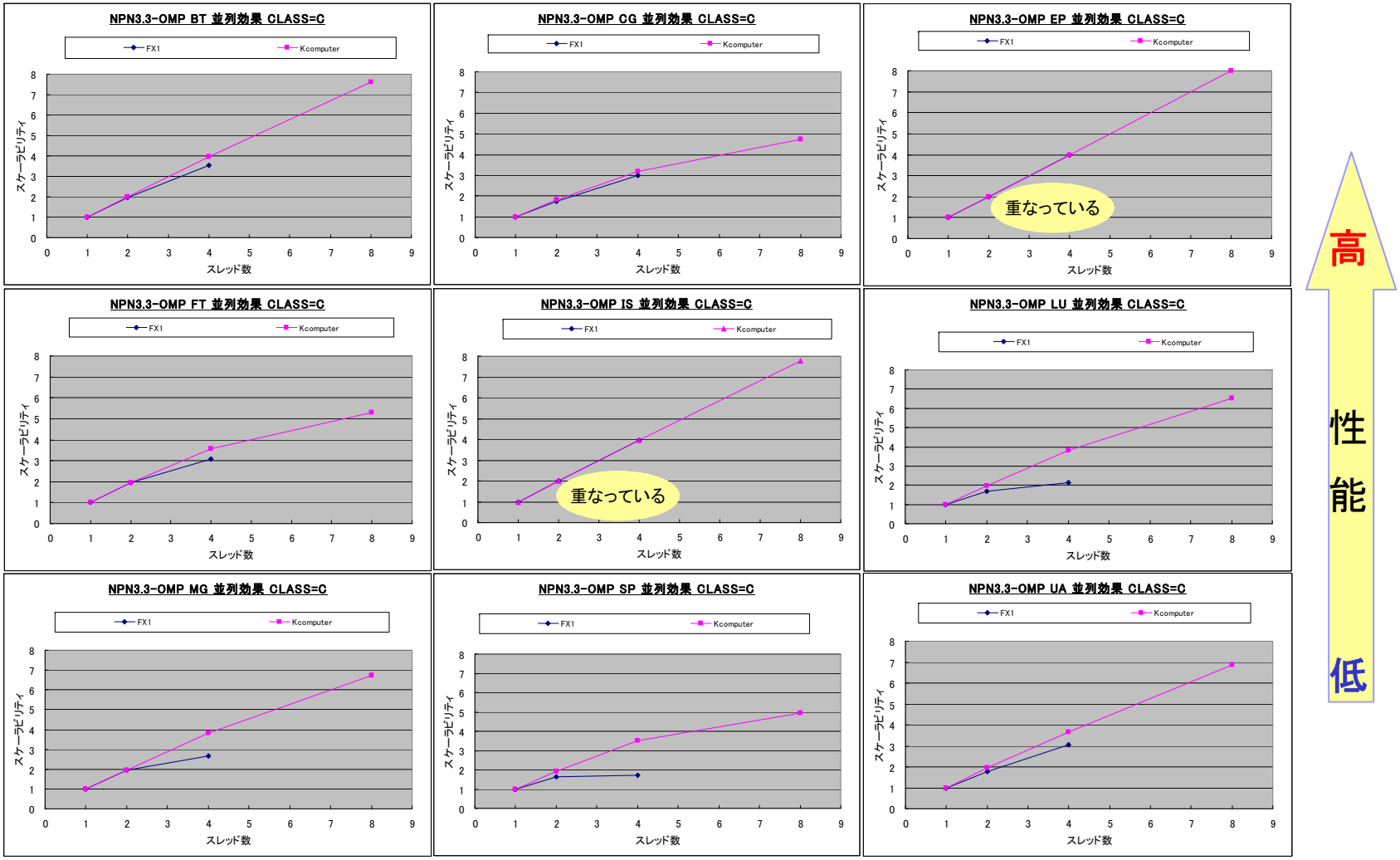
SPARC64VIIIfxのVISIMPACT性能

VISIMPACTの性能（NPB）

昨年度報告

FUJITSU

- FX1のVISIMPACTと比べて同等以上のスケーラビリティを達成



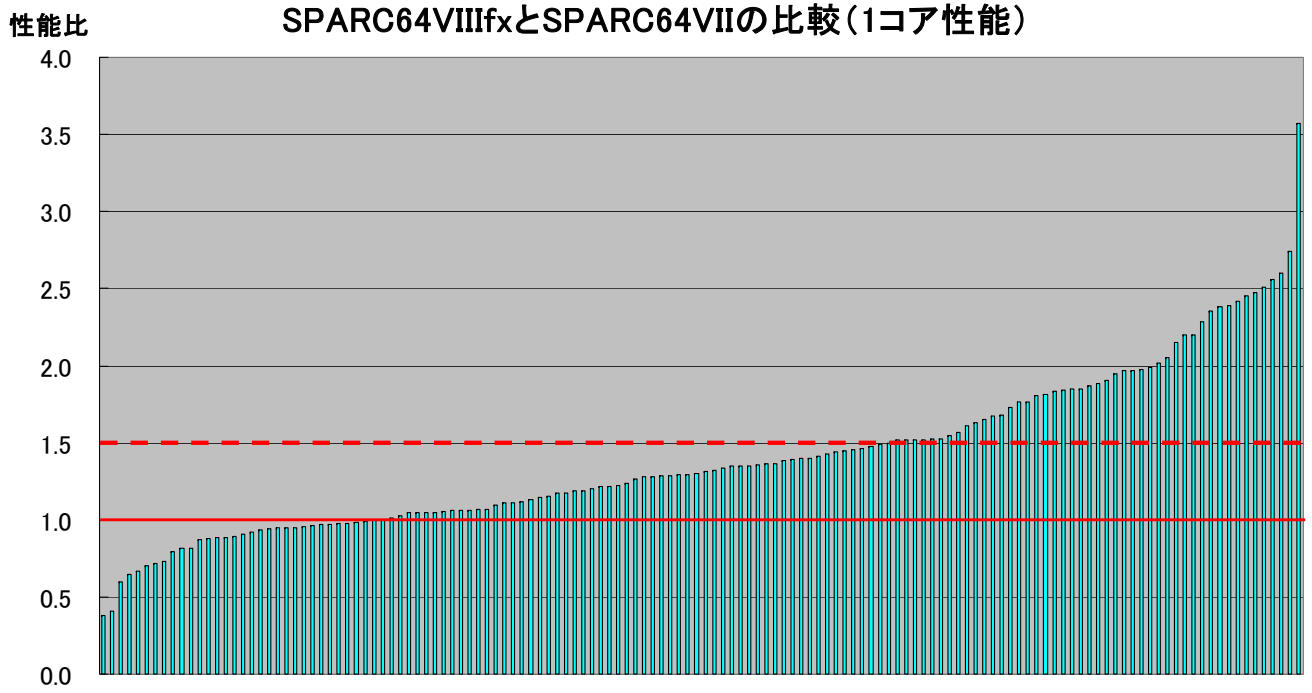
FUJITSU

SPARC64VIIとSPARC64VIIIfxの比較（1コア性能）

SPARC64VIIIfxの性能（実コードセット）

FUJITSU

- SPARC64VIIとSPARC64VIIIfxの1コア性能を比較
  - 周波数 0.8倍 × SIMD 2倍 → 1.6倍のハードピーク性能比に対して、HPC-ACEを活用して平均1.5倍の性能向上が目標

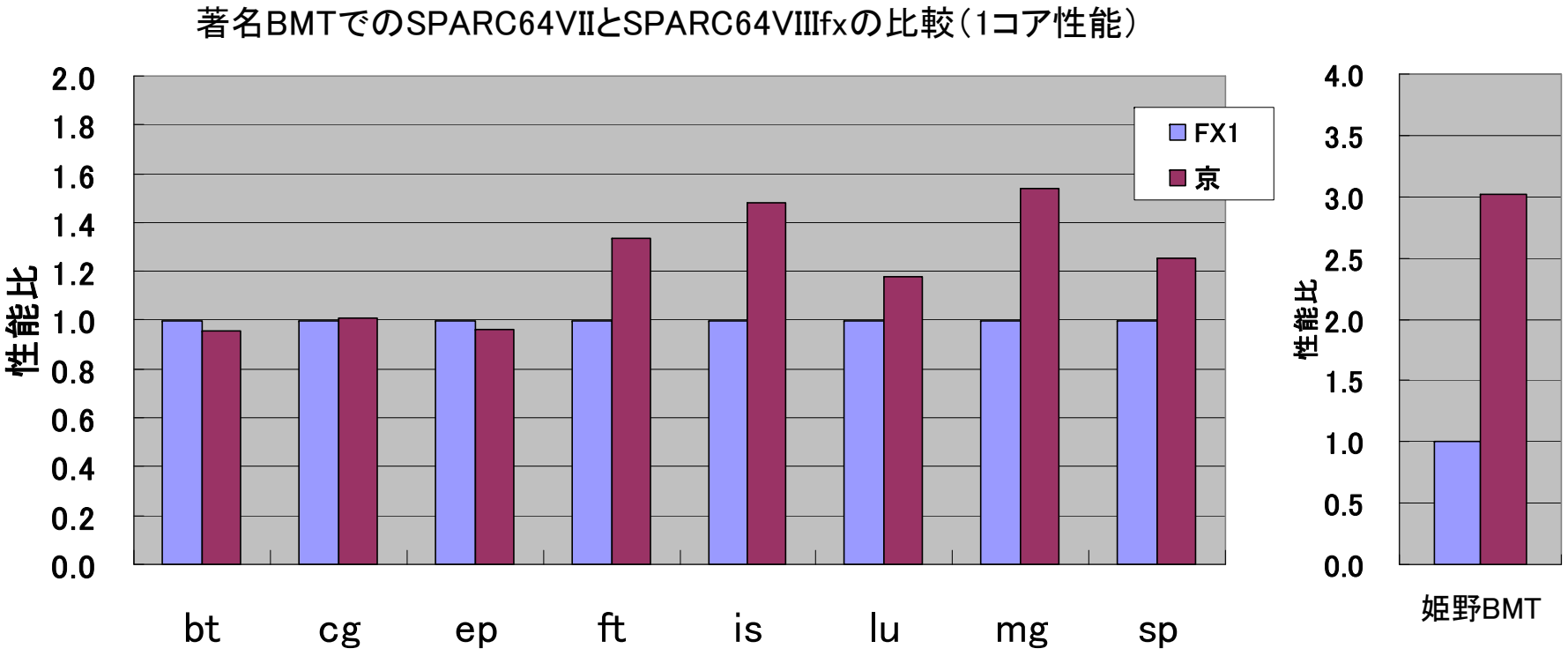


- 約140本の実コードで平均1.4倍の性能を達成
  - ベクトルマシン向けのコードで性能向上が顕著
  - コンパイラのエンハンス（チューニング）で更に向上

SPARC64VIIIfxの性能（ NPB、姫野BMT ）

FUJITSU

- SPARC64VIIとSPARC64VIIIfxの1コア性能を比較
  - 周波数 0.8倍 × SIMD 2倍 → 1.6倍のハードピーク性能比に対して、HPC-ACEを活用して1.5倍の性能向上が目標



- NPBで平均1.21倍、姫野BMTではHPC-ACEの効果でハードピーク性能比を大きく超える3倍の性能を測定

FUJITSU

浮動小数点レジスタの拡張効果について

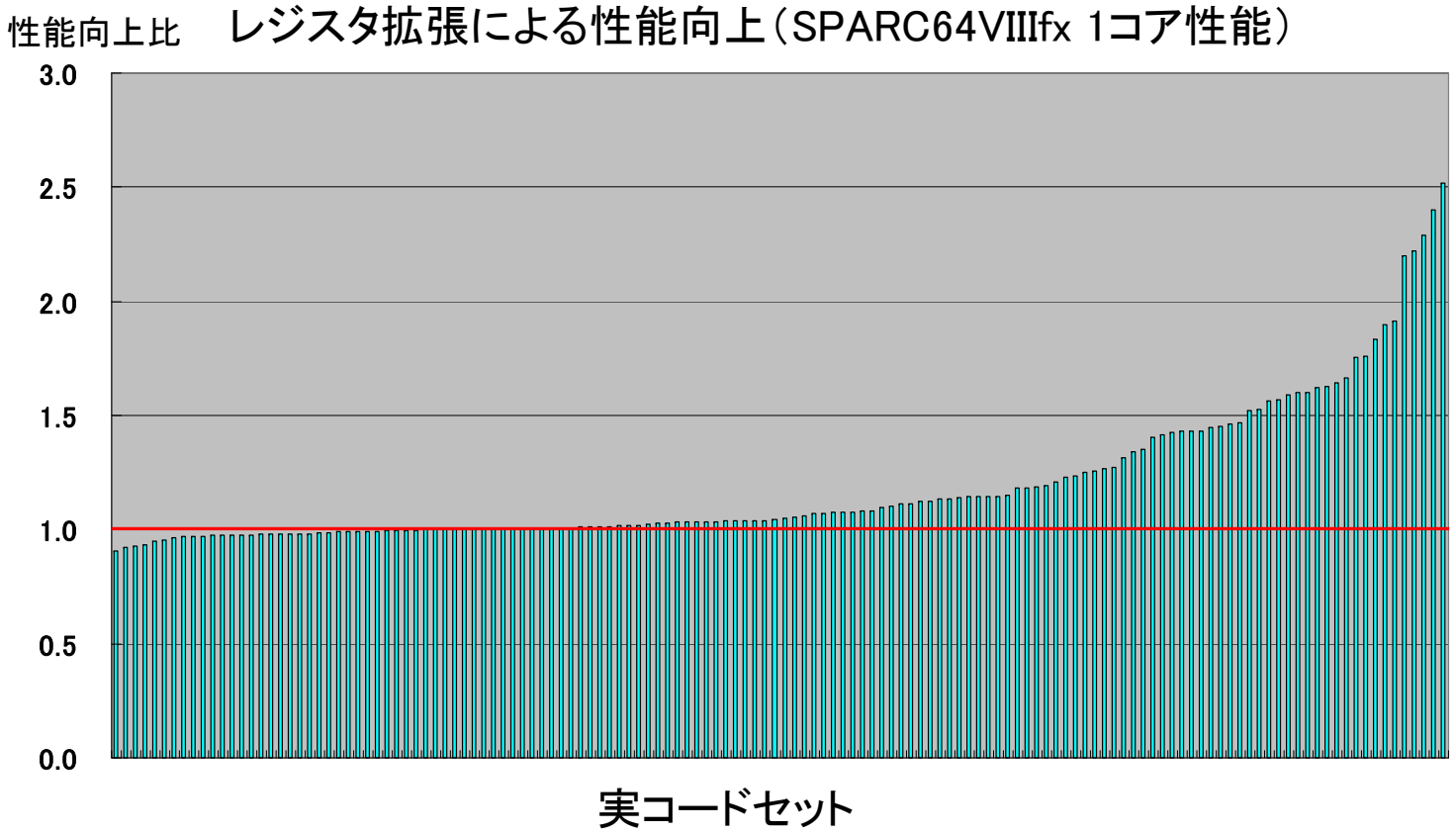
期待効果

1. ソフトウェアパイプライニング対象の拡大/アンローリング展開数の増加
   ➔ 命令並列度の向上による演算待ち時間の短縮
2. レジスタ退避のためのメモリアクセス(スピル)の削減
   ➔ メモリアクセス命令数の削減 / キャッシュミス削減

Fレジスタ拡張の効果（実コードセット）

FUJITSU

- HPC-ACEで、使用するFR数を32個と256個として性能を比較

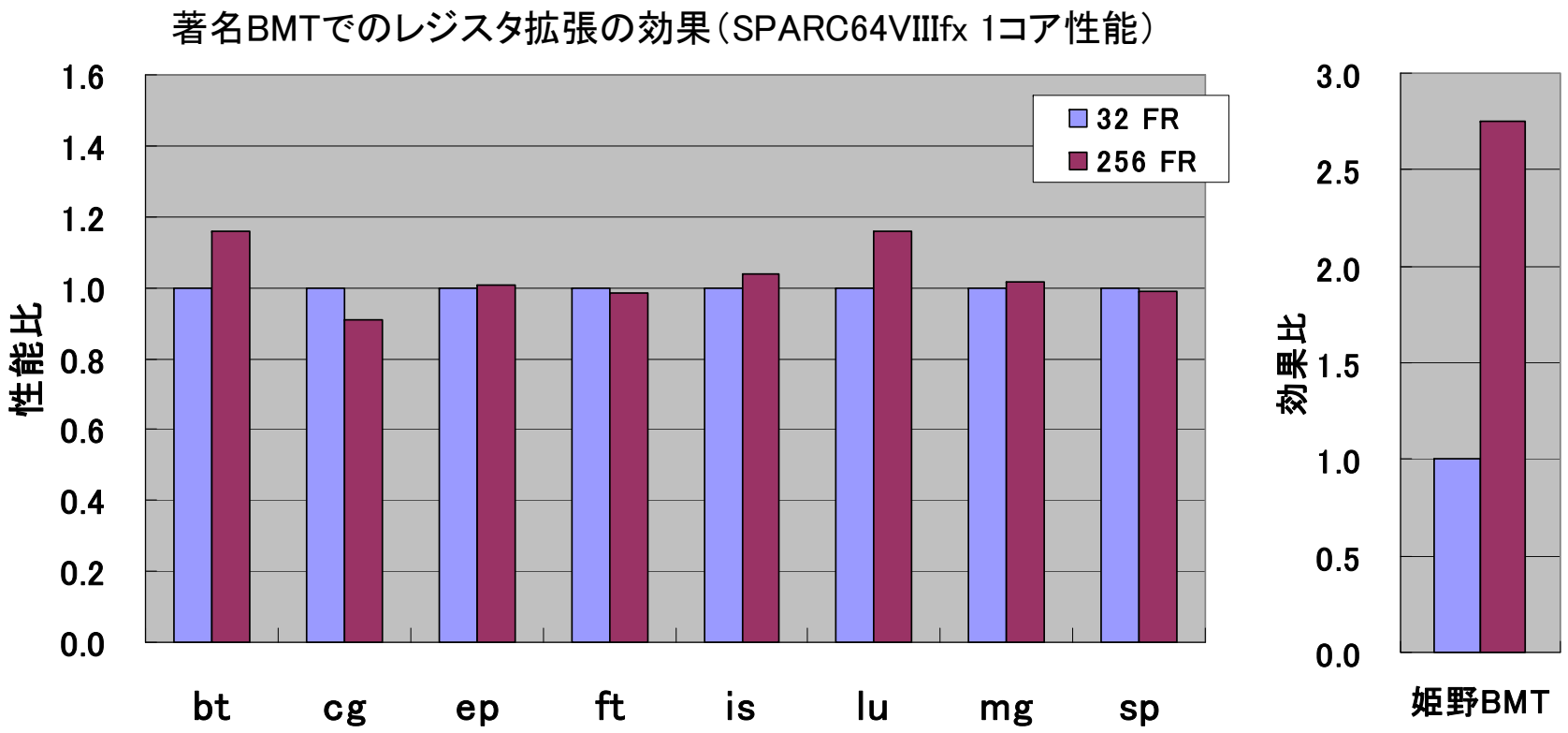


- 約140本の実コード中、73%のコードで効果を確認、平均で1.2倍の性能向上
  - 性能低下しているのはループの回転数が極端に少ないケース

Fレジスタ拡張効果（NPB、姫野BMT）

FUJITSU

- 著名BMT（NPB、姫野BMT）で、32個と256個のFRの性能を比較



- NPBで平均1.03倍、姫野BMTで2.75倍の効果
  - NPBで効果が低いのはループ中に分岐が多くスケジューリングできないため
    マスク付SIMD化で性能改善可能（現在、本機能開発＆テスト中）

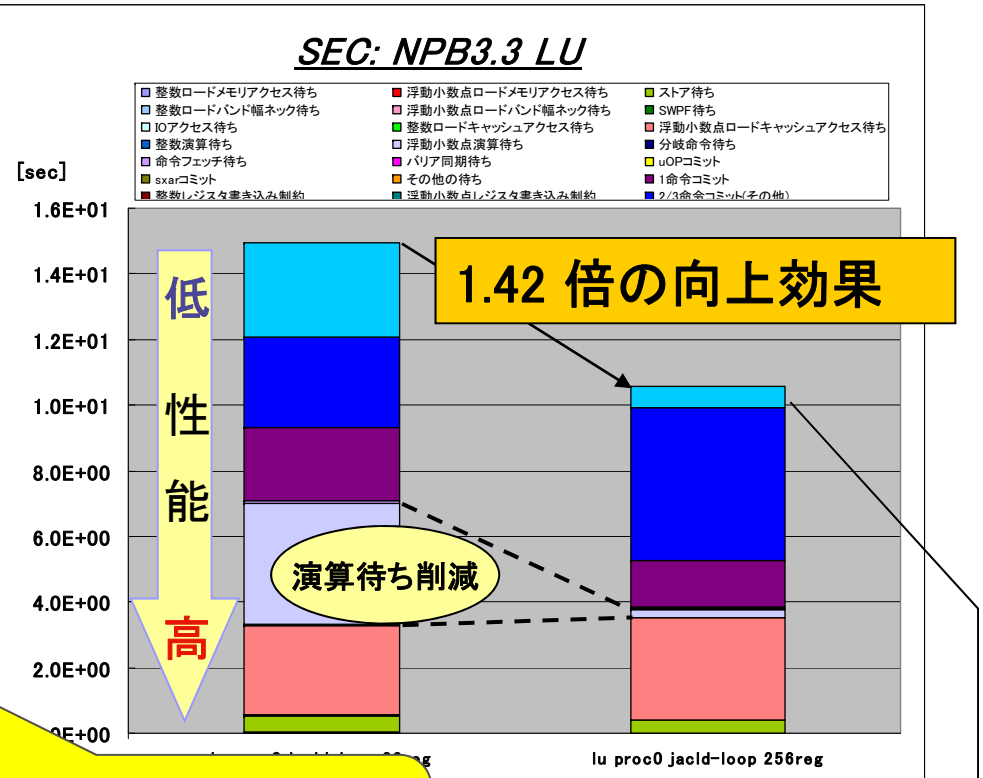
レジスタ拡張効果の分析（NPB3.3-LU）

FUJITSU

NPB3.3-LU 高コストループ(340行)

```
 39   1                do j = jst, jend
 40   2   2              do i = ist, iend
 41   2
 42   2          c-------------------------------------------------------------------
 43   2          c   form the block daigonal
 44   2          c-------------------------------------------------------------------
 45   2   2                tmp1 = 1.0d+00 / u(1,i,j,k)
 46   2   2                tmp2 = tmp1 * tmp1
 47   2   2                tmp3 = tmp1 * tmp2
 48   2
 49   2   2                d(1,1,i,j) =  1.0d+00
 50   2             >                   + dt * 2.0d+00 * (   tx1 * dx1
 51   2             >                                      + ty1 * dy1
 52   2             >                                      + tz1 * dz1 )
 53   2   2                d(1,2,i,j) =  0.0d+00
 54   2   2                d(1,3,i,j) =  0.0d+00
 55   2   2                d(1,4,i,j) =  0.0d+00
 56   2   2                d(1,5,i,j) =  0.0d+00
  :
  : ~~~~~~~~~~~
367   2   2                 c(5,3,i,j) = - dt * tx2
368   2             >         * ( - c2 * ( u(3,i-1,j,k)*u(2,i-1,j,k) ) * tmp2 )
369   2             >         - dt * tx1
370   2             >         * (  c34 - c1345 ) * tmp2 * u(3,i-1,j,k)
371   2   2                 c(5,4,i,j) = - dt * tx2
372   2             >         * ( - c2 * ( u(4,i-1,j,k)*u(2,i-1,j,k) ) * tmp2 )
373   2             >         - dt * tx1
374   2             >         * (  c34 - c1345 ) * tmp2 * u(4,i-1,j,k)
375   2   2                 c(5,5,i,j) = - dt * tx2
376   2             >         * ( c1 * ( u(2,i-1,j,k) * tmp1 ) )
377   2             >         - dt * tx1 * c1345 * tmp1
378   2             >         - dt * tx1 * dx5
379   2
380   2   2               end do
381   1                 end do
```

SPARC64™ VIIIfx：1コア実行



1.42 倍の向上効果

ループボディが大きい場合でもHPC_ACEならスケジューリング効果が期待できる

レジスタ不足を補うために生成されていたロード命令も削減された。

	32reg	256reg	比率
浮動小数点ロード命令	9.43E+10	9.68E+09	0.10

FUJITSU

SIMD効果について

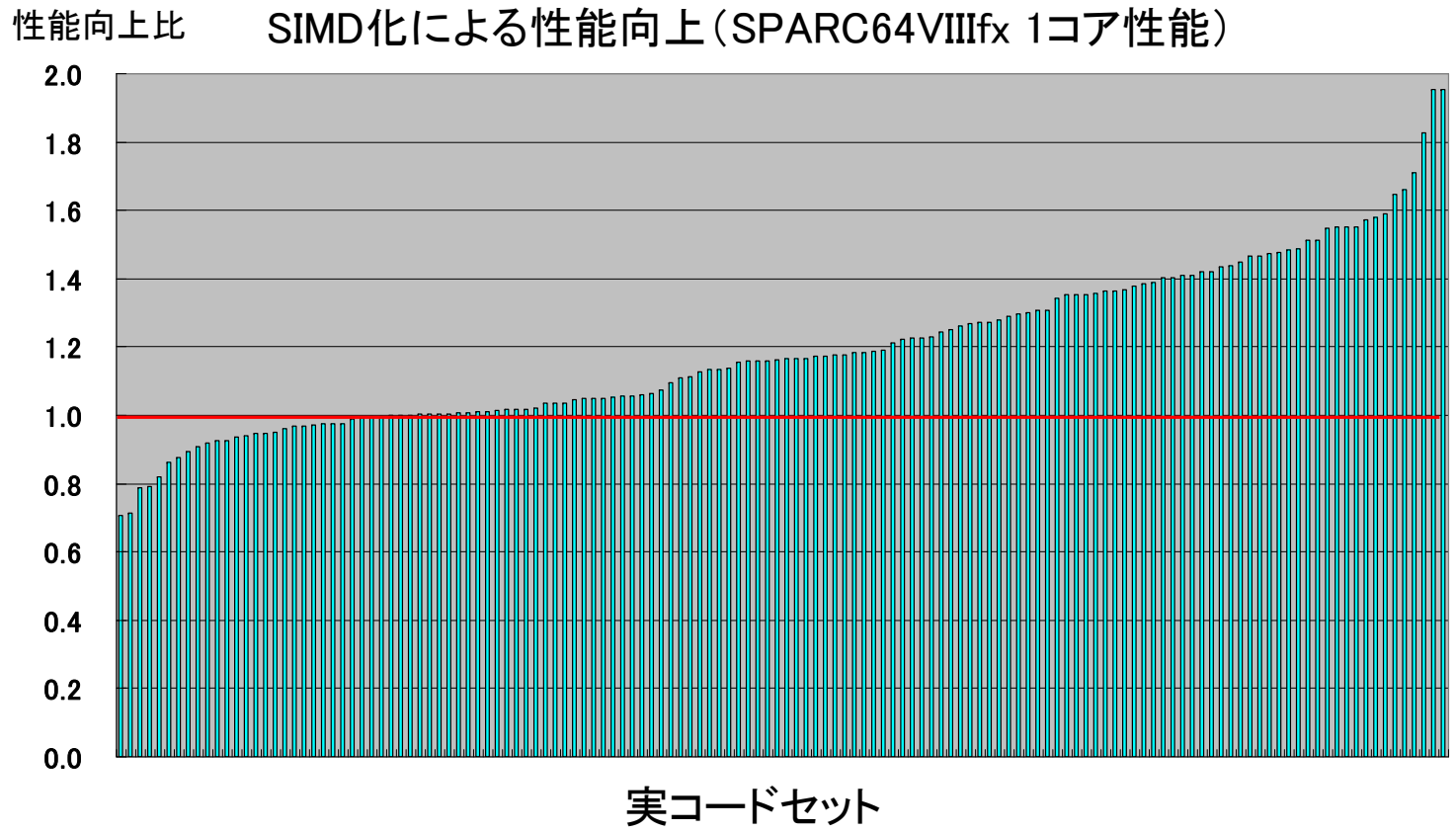
期待効果

1. 浮動小数点命令の2演算並列同時実行
   ➔浮動少数点命令の実行時間の削減
2. マスク付SIMD化で命令スケジューリング対象拡大
   ➔浮動少数点命令の演算待ち時間の削減

SIMD化の効果（実コードセット）

FUJITSU

- SIMDを使用する場合と使用しない場合の性能を比較

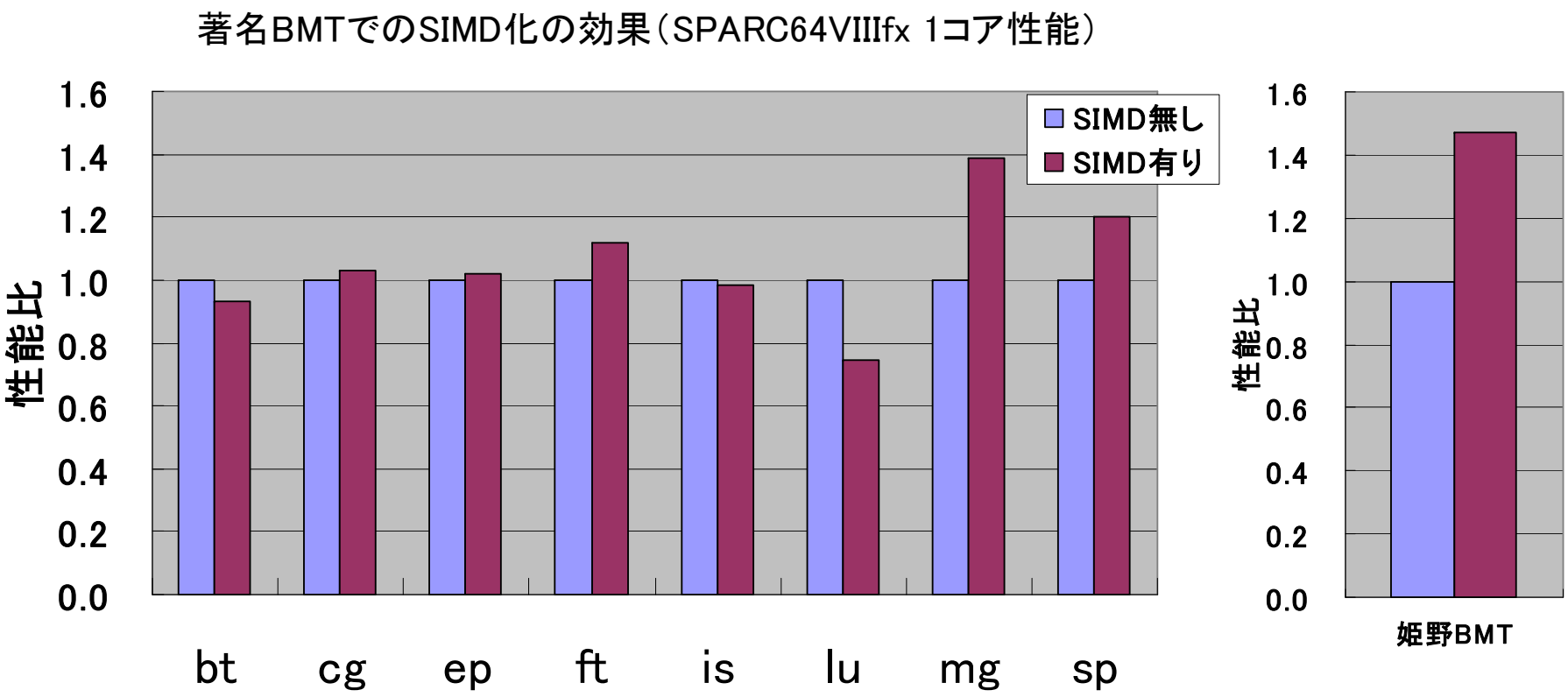


- 約140本の実コード中、80%のコードで効果を確認、平均で1.2倍の性能向上
  - コンパイラのSIMD化認識とSIMD命令生成に課題が残っており、SIMD化で性能低下する場合が残存

SIMD化の効果（NPB、姫野BMT）

FUJITSU

- 著名BMT（NPB、姫野BMT）で、SIMD化の有無の性能を比較



- NPBで平均1.05倍、姫野BMTで1.47倍の効果
  - NPBで効果が低いのはループ中に分岐が多いため、マスク付SIMD化により性能向上可能（現在、本機能は開発＆テスト中）

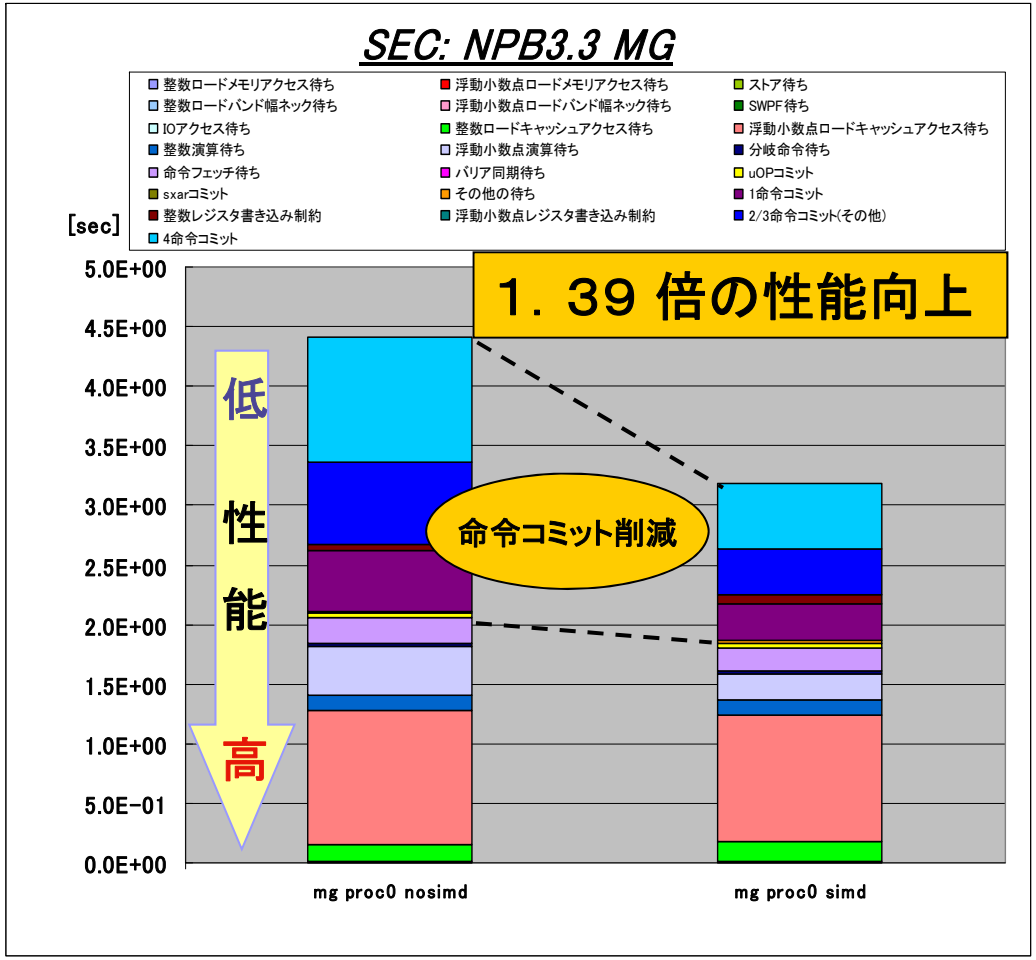
SIMD化の効果の詳細（NPB3.3 MG）

FUJITSU

- SIMD化で実行命令数が削減され、命令実行時間が短縮されて性能が向上

```
685   1              do i3=2,n3-1
686   2                do i2=2,n2-1
                    <<< Loop-information Start >>>
                    <<<  [OPTIMIZATION]
                    <<<    SIMD
                    <<<    SOFTWARE PIPELINING
                    <<< Loop-information  End >>>
687   3    4v              do i1=1,n1
688   3    4v                u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
689   3              >             + u(i1,i2,i3-1) + u(i1,i2,i3+1)
690   3    4v                u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
691   3              >             + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
692   3    4v              enddo
                    <<< Loop-information Start >>>
                    <<<  [OPTIMIZATION]
                    <<<    SIMD
                    <<<    SOFTWARE PIPELINING
                    <<< Loop-information  End >>>
693   3    6v              do i1=2,n1-1
694   3    6v                r(i1,i2,i3) = v(i1,i2,i3)
695   3              >                 - a(0) * u(i1,i2,i3)
 :
702   3              >                 - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
703   3              >                 - a(3) * ( u2(i1-1) + u2(i1+1) )
704   3    6v              enddo
705   2                  enddo
706   1                enddo
```

	NOSIMD	SIMD	比率
命令数	1.31E+10	7.24E+09	0.55



命令数削減 ➔ 命令コミット時間の削減

FUJITSU

セクタキャッシュ機能について

期待効果

1. キャッシュをローカルメモリ的に使用することが可能
   ➔ キャッシュミス = メモリアクセスコストの削減

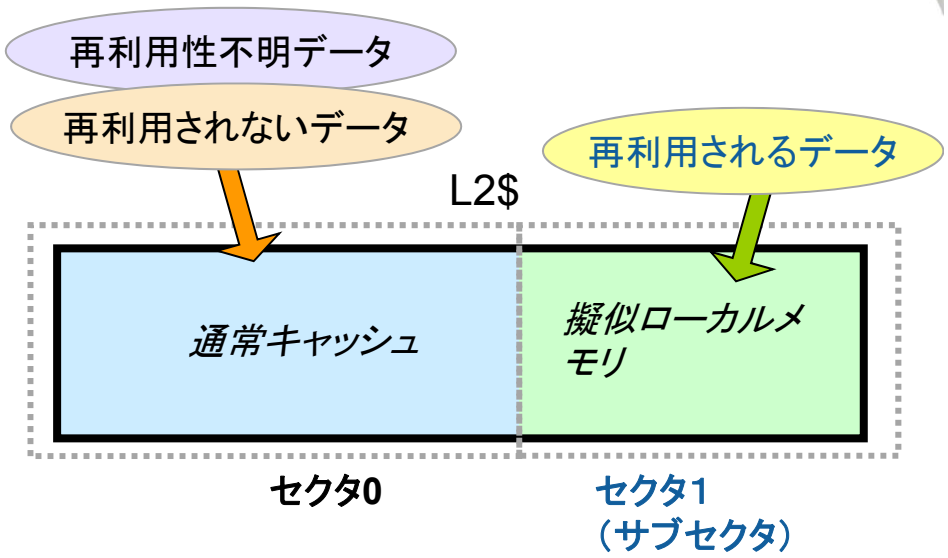
セクタキャッシュの利用イメージ

FUJITSU

●セクターキャッシュ: *擬似ローカルメモリ*

➔ソフトウエアが、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

- 再利用する配列
  ➔ セクタ1（サブセクタ）を使用
- その他 ➔ 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ない
- ユーザは指示行でセクタ1に載せる配列を指定できる




```
!ocl cache_sector_size (8,2)
!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
```

〈意図〉
ループ中で配列 b と配列 c のアクセスによって配列 a がキャッシュから追い出されない

セクタキャッシュ指定のコンパイラ指示行の使用例

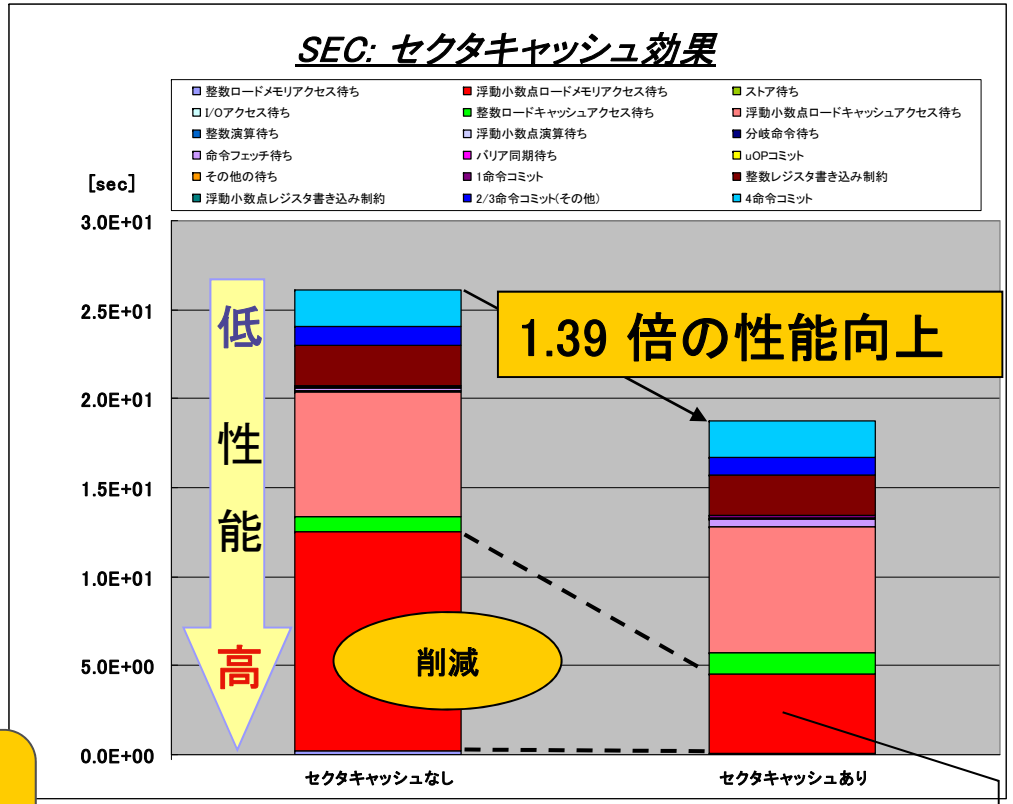
セクタキャッシュの効果

FUJITSU

- SPARC64VIIIfx 1ソケット（8コア）でセクタキャッシュの効果を検証

```
39                c----------------------------
40                !ocl cache_sector_size (3, 9)
41    1  s   s          do iter=1, itmax
42    1  s   s            call sub(a, b, c, s, n, m)
43    1  s   s          enddo
44                c----------------------------
:
~~~~~~~~~~~~
52                    subroutine sub(a, b, c, s, n, m)
53                    real*8    a(n), b(m), s
54                    integer*4 c(n)
55
56                !ocl cache_subsector_assign (b)
                  <<< Loop-information Start >>>
                  <<<  [PARALLELIZATION]
                  <<<    Standard iteration count: 728
                  <<<  [OPTIMIZATION]
                  <<<    SIMD
                  <<<    SOFTWARE PIPELINING
                  <<< Loop-information  End >>>
57    1 pp  4v          do i=1,n
58    1  p  4v            a(i) = a(i) + s * b(c(i))
59    1  p  4v          enddo
60
61                    end
```

配列bのサイズは4.5MB。
L2キャッシュに載るサイズ。



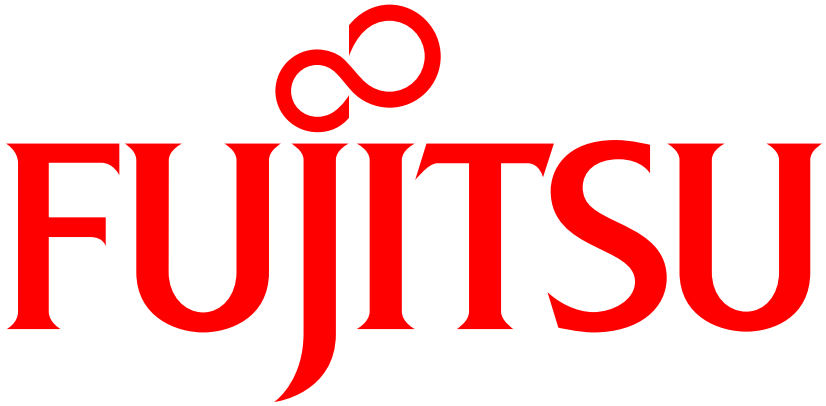
	セクタキャッシュなし	セクタキャッシュあり	比率
L2キャッシュミス数	6.57E+09	3.95E+09	1.66

セクタキャッシュにより、配列bがL2$にキープされてメモリアクセスコストが削減され、性能が向上

まとめ

FUJITSU

- 超並列処理の実用化に向けハイブリッドのプログラミングを容易にするアーキテクチャについて紹介。
  VISIMACT
  SPARC64™ VIIIfx（HPC-ACE)
  新インターコネクト
- このアーキテクチャを活かす言語処理系について説明。
- コンパイラ（開発中）での性能評価を実施した。残存課題はあるもののHPC-ACEの特徴を使い、コアでFX1比平均で1.4倍（Peak比1.6倍）出ている。→さらにエンハンスを継続し、1.5倍を目指す。
- 次世代スーパーコンピュータ「京」の2012年の運用に向け、着実に開発作業を実施していく。
- 富士通は、本アーキテクチャを基盤にし、HPCに向けたアーキテクチャ及びコンパイラの開発を継続していく。



shaping tomorrow with you