THE POSSIBILITIES ARE INFINITE FUJITSU

# 3.4. プログラミング指針

## ーフラットMPIとハイブリッド並列についてー

富士通株式会社　青木 正樹、鈴木 清文

## 目次

FUJITSU

- プログラミングモデル
- VISIMPACT
  （性能とプログラミングの容易性）
- VISIMPACTプログラミングの指針
- フラットMPIの性能特性
- 効果例
- まとめ
- 【補足資料】高機能スイッチの位置づけと効果



## 参考資料

FUJITSU

- 【補足資料】高機能スイッチの位置づけと効果
  - 「効果例」で使用されている高機能スイッチ（高機能SWあるいはISWと記述）についての解説です。
- SMPクラスタWG成果報告書
  「スカラ並列プログラミング チューニング・ガイド」
  http://www.ssken.gr.jp/MAINSITE/activity/workinggroup/smpc/guide.html
  - スカラSMP計算機(富士通PRIMEPOWERシリーズ)上でプログラムの性能評価作業およびチューニングを行う際のガイドです。『SMPクラスタWG』の成果として作成されました。(2003/12/25発行)

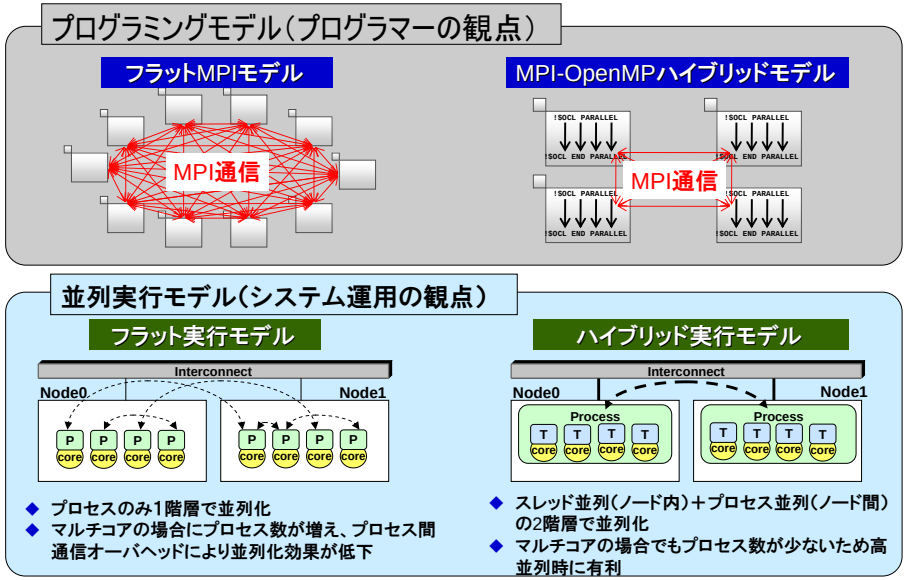


FUJITSU

**プログラミングモデル**

# プログラミングモデルと並列実行モデル

FUJITSU




- プロセスのみ1階層で並列化
- マルチコアの場合にプロセス数が増え、プロセス間通信オーバヘッドにより並列化効果が低下

- スレッド並列（ノード内）＋プロセス並列（ノード間）の2階層で並列化
- マルチコアの場合でもプロセス数が少ないため高並列時に有利



# 並列プログラミング環境

FUJITSU

| プログラミングモデル/プロセス並列手法 | | ノード間 | ノード内 |
|---|---|---|---|
| ハイブリッドモデル | データ並列 | XPFortran | OpenMP, 自動並列（スレッド並列） |
| | メッセージパッシング | MPI | |
| フラットモデル | データ並列 | XPFortran | |
| | メッセージパッシング | MPI | |



# プログラミングモデル選択のポイント

FUJITSU

| モデル | メリット | デメリット |
|---|---|---|
| ハイブリッド並列（＋SIMD化） | プロセス数増加を抑える | スレッド並列性能に依存（OpenMP化も考慮） |
| フラットMPI（＋SIMD化） | 1階層のみの考慮でよい | プロセス数増加の弊害(通信オーバーヘッド増大) |

結局のところ、

フラットMPIの性能特性　VS　スレッド並列の”プログラミング”×”性能”

がプログラミングモデル選択のポイントとなる。



FUJITSU

**VISIMPACT**
**（性能とプログラミングの容易性）**

# VISIMPACTとは(1)

FUJITSU

（VISIMPACT：
VIrtual Single processor by Integrated Multi-core Parallel ArChiTecture）

●コンセプト

- マルチコアCPU内の高効率なスレッド並列を実現する技術
- 高効率なハイブリッド並列実行モデルの実現を支援

*MPI + スレッド並列処理(自動並列 /OpenMP)*

●狙い

- マルチコアCPUを高速な一つのCPUを扱うことで……
  - ◆MPI プロセス数を $1/n_{core}$ に減らす
    - ➔並列処理効率を向上
  - ◆メモリアクセスを軽減

●技術的挑戦

- コア間のスレッドレベル並列処理のオーバヘッドをどう減らすか?

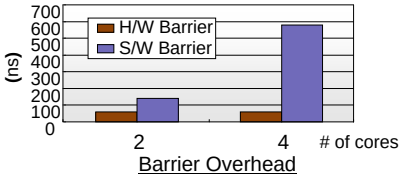


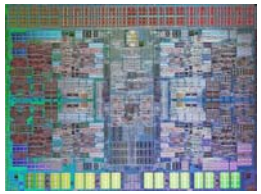
# VISIMPACTとは(2)

FUJITSU

■CPU 技術

- コア間のハードバリア機能
  - ➔ソフトウェアバリアと比較して10倍の高速化
  - ➔コア数に依存せずに一定のオーバヘッドを実現



**SPARC64™ VII**
*Real quad-core CPU for Technical Computing*
(2.5 GHz, 40 GFlops/chip)

- 共有L2キャッシュメモリ (6 MB)
  - ➔キャッシュ間のメモリ交換を軽減
  - ➔キャッシュメモリの利用を効率化

●コンパイラ技術

- ベクトル化の技術を適用した、高効率なスレッド並列(自動並列、OpenMP) を実現

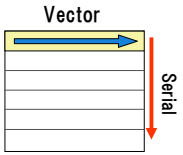


# VISIMPACTループ処理の考え方

FUJITSU

■Vector処理

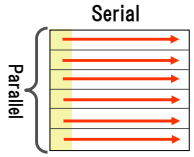
```
  DO J=1,N
V DO I=1,M
V   A(I,J)=A(I,J+1)*B(I,J)
V END
  END
```




- 適用範囲は広い
- 同期は頻繁だが低コスト

●従来の並列処理（疎粒度並列）

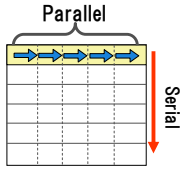
```
P DO J=1,N
P DO I=1,M
P   A(I,J)=A(I,J)*B(I,J)
P END
P END
```




- 適用範囲は狭い(広範囲な分析が必要)
- 同期は少ない

●今回の並列処理（細粒度並列）

```
  DO J=1,N
P DO I=1,M
P   A(I,J)=A(I,J+1)*B(I,J)
P END
  END
```




- 適用範囲は広い
- 同期は頻繁
- キャシュ間のデータの取り合いが頻繁に発生

VISIMPACT が対処



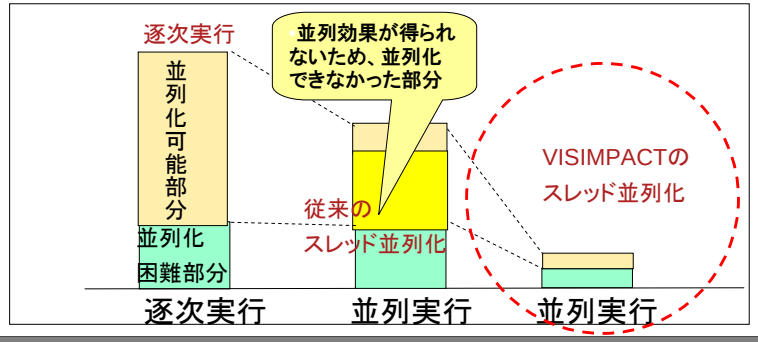
# VISIMPACTの効果

FUJITSU

**従来のスレッド並列化の問題**
**プロセス並列やベクトル化に比べ並列オーバヘッドが大きい。**
**そのため細粒度並列では並列効果が低くスレッド並列が利用しずらかった。**

⇩⇩ ⇩⇩VISIMPACTにより解決⇩⇩⇩⇩

**並列処理のオーバヘッド（同期、フォルスシュア）削減の結果、コンパイラによる細粒度並列化が可能となった。**

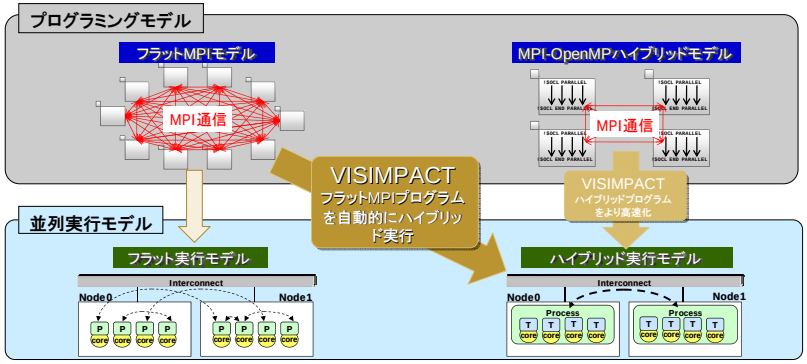


# 容易なハイブリッド実行の実現

FUJITSU

◆VISIMPACT
高効率なCPUチップ内(コア間)自動スレッド並列で、複数コアの仮想単一CPU化を実現

◆VISIMPACTによるフラットMPIプログラムのハイブリッド化

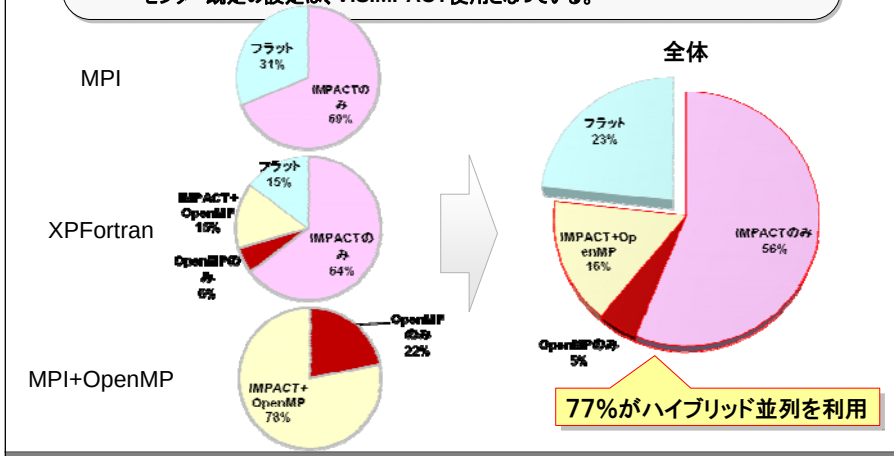





# 参考：JAXAセンターでのハイブリッド並列の利用状況

FUJITSU

- コンパイル時ログの情報から利用状況を判断
  - 情報採取期間：2009.1～2009.9、コンパイル実施者数：145人
  - 情報数に重複あり（1人の利用者が異なるスタイルでコンパイルした場合）
  - センター既定の設定は、VISIMPACT使用となっている。

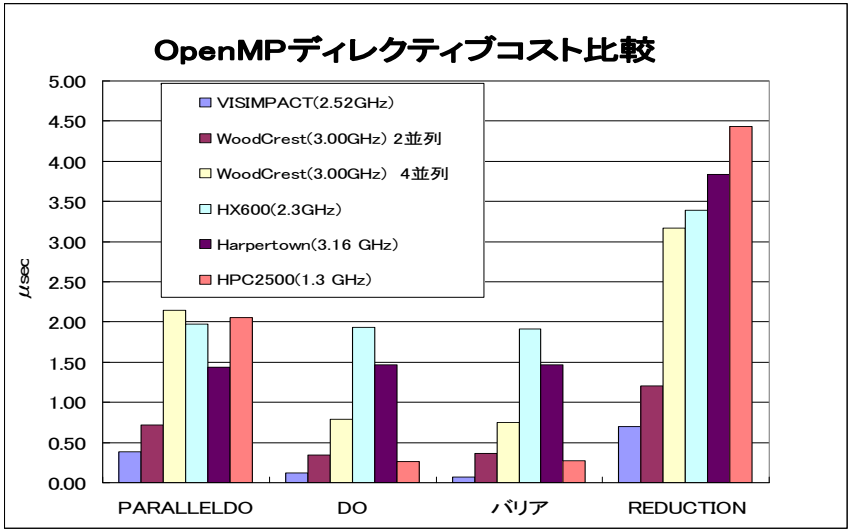





# 検証：スレッド並列のオーバーヘッド

FUJITSU

- スレッド並列のオーバーヘッドでFX1は他システムを圧倒

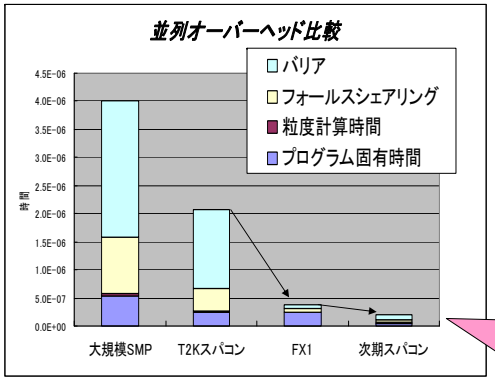





# 検証：並列化のオーバヘッド実測と推定

FUJITSU

- 大規模SMP(HPC2500),T2Kスパコン(HX600),FX1(SPARC64Ⅶ)で並列化のオーバヘッドの詳細を実測（4スレッド）。




検証用プログラム(n=900)

```
        t0 = wclock()
!$omp parallel private(j,jsw)
        Do j = 1, nrep
          jsw = jswitch( krit, indj, j )
!$omp do
          Do i = 1, n
            y(i,jsw) = y(i,jsw) + c0*x1(i)
          End Do
!$omp end do
        End Do
!$omp end parallel
        t1 = wclock() - t0
```

これからのアプリ開発者は、スレッドへ並列のオーバヘッドを気にする必要なし！

## スレッド並列化プログラミングの容易性

FUJITSU

**高い並列化能力(*)を持つ自動並列化コンパイラ（＋OpenMP指示行）がプログラミングを支援。**

■ 従来自動並列化
（疎粒度並列）
外側ループで
並列化を行ってきた。

■ VISIMPACTの自動並列化
（細粒度/疎粒度並列）
細粒度でも並列化効果あるため内側ループ(*)でも外側ループでも並列化可能。

＊ベクトル化と同様に内側ループを並列化（細粒度）
→技術的に確立しているベクトル化並みの
高い自動並列化率が可能



## 自動並列化能力

FUJITSU

ハードウェアのサポート（VISIMPACT）により、ベクトル化より広い範囲に適用！

| | | ベクトル化 | VISIMPACTスレッド並列（計画） | 現行スレッド並列 |
|---|---|---|---|---|
| 範囲 | | ○（最内ループ） | ◎（最内ループ＋手続き） | △（外側ループ） |
| データ型 | | ○（4/8バイト型） | ◎（全て） | ◎（全て） |
| データ依存関係 | 依存なし | ○ | ○ | ○ |
| | 順方向依存 | ○ | ○ | × |
| | 逆方向依存 | × | ○ | × |
| ループの演算内容 | 四則演算 | ○ | ○ | ○ |
| | リダクション演算 | ○ | ○ | ○ |
| | 収集・拡散 | ○ | ○（計画） | × |
| | DOブランチ | ○ | ○（計画） | × |
| 粒度 | | ○（ループ長数＋） | ○（ループ長数＋） | ×（ループ長数＋） |

ANLベクトル化コンテストプログラム（全135ループ）を用いて、各種ループの解析能力を比較

| | ベクトル化 | VISIMPACTレッド並列（計画） | 現行スレッド並列 |
|---|---|---|---|
| ベクトル化、並列化可能なループ数 | 86ループ | 89ループ以上 | 67ループ |



## スレッド並列化支援：スレッド並列化の確認情報

FUJITSU

言語開発環境において以下を支援。

■実行時情報：スレッド並列化率

■ソースコード情報：並列化（Pソース）表示

表示例　PPマーク:並列化対象のループ　Pマーク:並列化対象の文

```
(line-no.)(nest)(optimize)

  7     1     p          do i=2,n
  8     2    pp 8           do j=2,n
  9     2     p 8             a(i,j)=a(i,j)+b(i,j)*c
 10     2     p 8           enddo
 11     1     p          enddo
```

備考：最適化情報として「SIMD化」「ループ交換」「一重化」「prefetch情報」等もあり



## スレッド並列化支援：OpenMP指示行の自動生成機能

FUJITSU

翻訳オプションに-Qmを指定することで、コンパイルリストに加え、自動並列化の状況をOpenMP指示行により表現した原始プログラムを生成します。（**プラットフォーム無依存コードを開発するための支援機能**）

frt -Kparallel **-Qm** sample.f90
↓
sample.omp.f90（生成されるソースファイル）

以下に、生成されるソースの例を示します。

例：オリジナルソース (sample.f90)

```
parameter      (imax=124, jmax=124)
real      *4   array( imax, jmax)

    do 10  j = 1, jmax
    do 10  i = 1, imax
      array(i,j) = array(i,j) + (i*j)
 10  enddo
    end
```

例：生成されるソース（sample.omp.f90）

```
parameter      (imax=124, jmax=124)
real      *4   array( imax, jmax)

!$OMP PARALLEL DO PRIVATE(i) !FRT
    do 10  j = 1, jmax
    do 10  i = 1, imax
      array(i,j) = array(i,j) + (i*j)
 10  enddo
    end
```

## スレッド並列化（まとめ）：容易に高性能プログラミングが可能

FUJITSU

- プログラミングの容易性

自動ベクトル化　≒　VISIMPACTの自動並列化

ベクトル化プログラミングは容易：万人が認知。

なぜ、ベクトル化プログラミングは容易なのか？

①細粒度のループを対象

②対象ループは、ループ繰り返しによるデータ依存性なし

VISIMPACTの自動並列化の基本ﾌﾟﾛｸﾞﾗﾐﾝｸﾞもﾍﾞｸﾄﾙ化に類似

- 演算性能向上への相乗効果

コア内の演算器を効率良く使うためには、ILP(*)を最大限に高める必要がある。

ILP：命令レベルの並列性(Instruction-level parallelism)

利用者は、細粒度の並列性を意識すること（上記②）により

スカラCPUの演算性能を最大限引き出すことが可能

→ループ繰り返しによるデータ依存なくなり

ループ中の命令スケジューリングやSIMD化が促進



FUJITSU

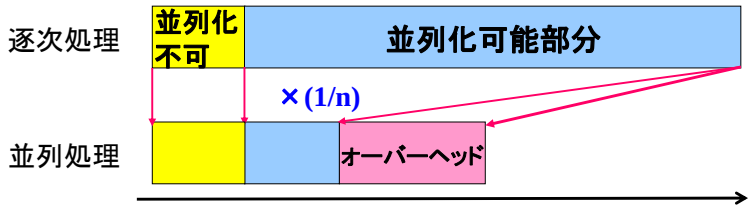
**VISIMPACTプログラミングの指針**
**（自動並列化プログラミング）**



## 自動並列化プログラミングのポイント

FUJITSU

- 並列化率の向上
- 大きな並列化粒度
- メモリアクセス競合の削減
- CPU負荷の分散





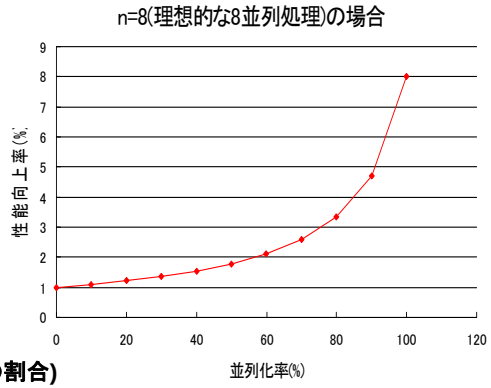

## 並列化率の向上

FUJITSU

■アムダールの法則

プログラム全体の性能向上率

$$= \frac{1}{(1-p)+\frac{p}{n}}$$




- p: 並列化率(並列化可能部分の割合)
- n: 並列化された部分の速度向上率

**⇒並列化率の向上が重要！**

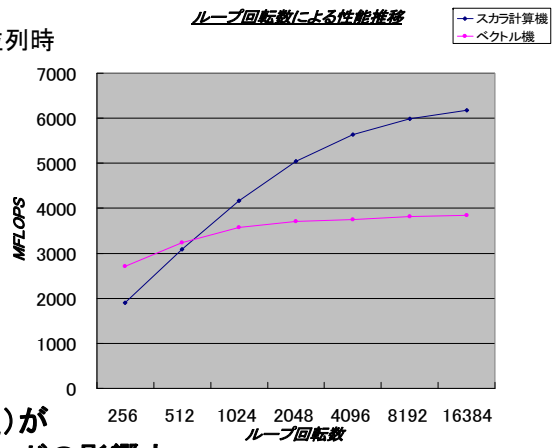
## 並列化の粒度を大きく

FUJITSU

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

評価ループソース

EXP関数を25演算と仮定




一般に、ループ長（粒度）が
小さいと並列化オーバヘッドの影響大

⇒粒度を大きく（外側で並列化）

⇒ただし、VISIMPACTでは影響小



## メモリアクセス競合の削減

FUJITSU

### ■メモリアクセス（キャッシュ）競合の発生例と改善策

```
real(8)
    a(256,256),b(256,256),
    c(256,256),d(256,256),
    e(256,256),f(256,256)

do j=1,256
  do i=1,256
    a(i,j)=b(i,j)+c(i,j)*d(i,j)
          +e(i,j)*f(i,j)
  enddo
enddo
```

```
real(8)
    a(257,256),b(257,256),
    c(257,256),d(257,256),
    e(257,256),f(257,256)

do j=1,256
  do i=1,256
    a(i,j)=b(i,j)+c(i,j)*d(i,j)
          +e(i,j)*f(i,j)
  enddo
enddo
```

キャッシュ機構の特性から2のべき乗のデータサイズを持つ大きなデータ間でキャッシュの競合が発生しやすい。

改善策：

配列の１次元目の大きさ変更（＋1）

（または、-Karraypad系オプション指定）



## CPU負荷の分散

FUJITSU

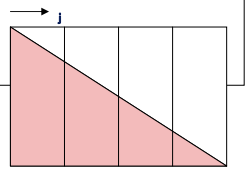
### ■CPU負荷の分散

特定のCPUの負荷（実行時間）が大きいと、全体の実行時間がそのCPUの実行時間にひきづられる。

⇒各CPUの負荷（分担）は均等になるのが望ましい

【三角ループ】不均等例1

```
DO J=1,M   ←均等分割で並列化
DO I=1,J   ★Jの値より処理量が変化
A(I,J)=…
ENDDO
ENDDO
```




【IF文を含むループ】不均等例2

```
DO J=1,M   ←並列化
DO I=1,N
IF (L(I,J)) THEN   ★L(I,J)の値により
…                  処理量が変化
ENDIF
ENDDO
ENDDO
```



FUJITSU

**フラットMPIの性能特性**
**（プロセス数増加時の課題）**

## フラットMPIの性能特性

FUJITSU

フラットMPIで実行した場合のプロセス数増加における性能面への課題は以下。

- 通信のオーバヘッド（時間）増加
- アプリケーションが利用できるメモリ量減少
- OSジッタの影響

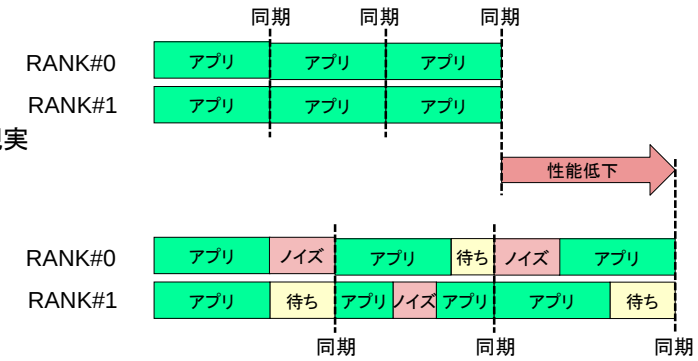
＋　最内ループに対しては、やはりSIMD化のプログラミングが必要。



## OSジッターについて

FUJITSU

- ノード毎にバラバラに発生するOSノイズ(システム処理)が、ノード間の同期処理によって、アプリ全体として累積し、実行性能を低下させる現象
  - 理想
  - 現実




➔ハイブリッド並列化によってプロセス並列数を減少させることで、OSジッターの影響を低減させることが可能



FUJITSU

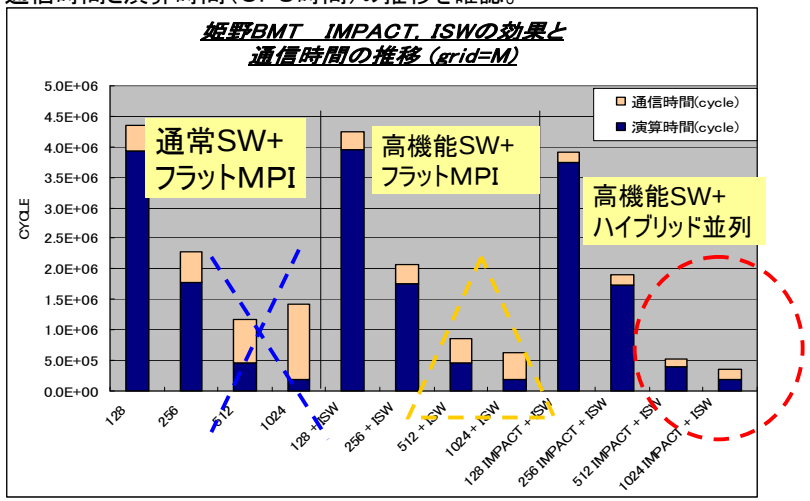
**効果例**



## 姫野BMT：FX1での性能検証

FUJITSU

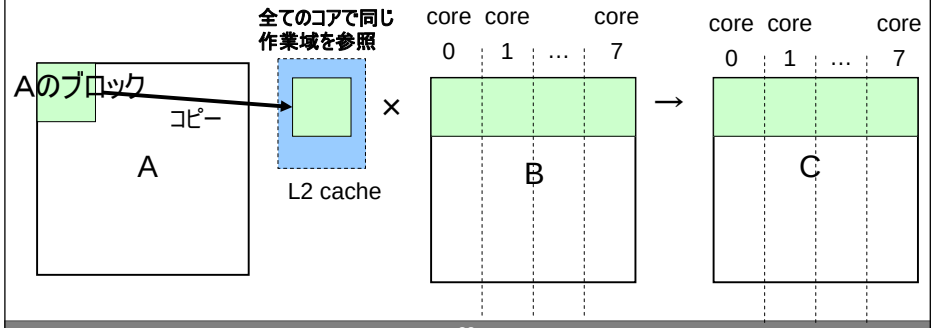
- MPI多次元分割、通信はMPI_allreduce使用、最内ループを自動並列。
  通信時間と演算時間（CPU時間）の推移を確認。



## LINPACKのDGEMM：共有キャッシュを利用

FUJITSU

- 行列積AB→Cの計算では、行列をブロックに分けて計算
  - 行列Aのブロックを L2 cacheに乗せる。ブロックが大きいほうが高効率
  - ➔フラットMPIではL2 cacheサイズがブロックが実質 1/コア数 となる→効率Down
  - ➔スレッド並列では、ブロックを共有してL2 cache 全体を活用→効率Up
- 効果
  - DGEMMで～2%の性能向上を推定






FUJITSU

**まとめ**



## プログラミングモデル まとめ

FUJITSU

- **富士通は、ハイブリッド並列プログラミングを推奨。**
  **➔ハードウェア/ソフトウェアの両面で利用者を支援**
  狙い：MPIでのみ記述されたフラットMPI実行アプリを自動でハイブリッド並列化へ（自動化できない部分のみOpenMPディレクティブで記述）
- **フラットMPIについても否定するものではない。**
  **ただし、通信オーバーヘッドに注意する必要あり。**



FUJITSU

# 【補足資料】
# 高機能スイッチの位置づけと効果

2009.5.23（2010.1.12改版）
富士通株式会社　清水 俊幸

# 高機能スイッチの位置づけと効果

FUJITSU

- 高並列アプリケーションの実行性能を高めるために，高機能スイッチを開発した
  - 高機能スイッチの概要については，2008年2月18日にPSIシンポジュームのプロトタイプの発表を参照

    http://www.psi-project.jp/images/event/toshiyuki_shimizu_20080218.pdf
- 高機能スイッチの位置づけと効果について考察した
  - 実装方式と将来性について分析し，ハード量は少なく、将来の超並列システムへの埋め込みが容易であることを確認した
  - 演算誤差について高機能スイッチが採用した1パス演算方式は最も誤差が小さいことを確認した
  - 実行速度のバラツキについて，ソフトによる1対1通信による実現の測定結果より，メッセージハンドリングを多数伴うソフトインプリは実行速度バラツキが大きいことを確認した
- 今後の取り組み
  - 大規模アプリケーションでの効果の確認などを行う



# コレクティブ通信高速化方法比較

FUJITSU

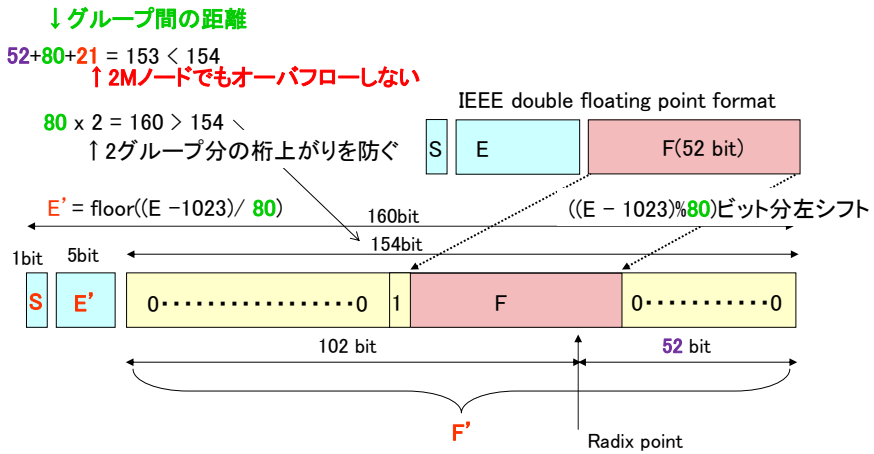
| | Host上のソフト | NIC上のマイクロ | 高機能スイッチ |
|---|---|---|---|
| 中継回数 | O(log(N)) | O(log(N)) | O(1) |
| OS処理による通信処理中断の影響 | あり | なし | なし |
| 課題 | レイテンシ削減<br>上記影響軽減 | レイテンシ削減 | ハードコスト低減<br>マルチユーザ対応 |
| 備考 | ハード追加なし | ハード追加なし<br>Myrinetなどでの実装例 | 専用ハード<br>BlueGeneで実現（2パス） |

- PSI 2-1では、新アルゴリズム考案により、原理的に従来方式の2倍高速化を実現
- 高機能スイッチ実現に必要なハード量は少なく、将来の超並列システムへの埋め込みが容易



# 2Mノードまでサポートするデータフォーマット

FUJITSU




- S: 符号
- E': 変換後の指数部（グループを表現）
- F': 変換後の仮数部（固定小数点として計算可能）



# 演算方式による誤差（有効ビット数）

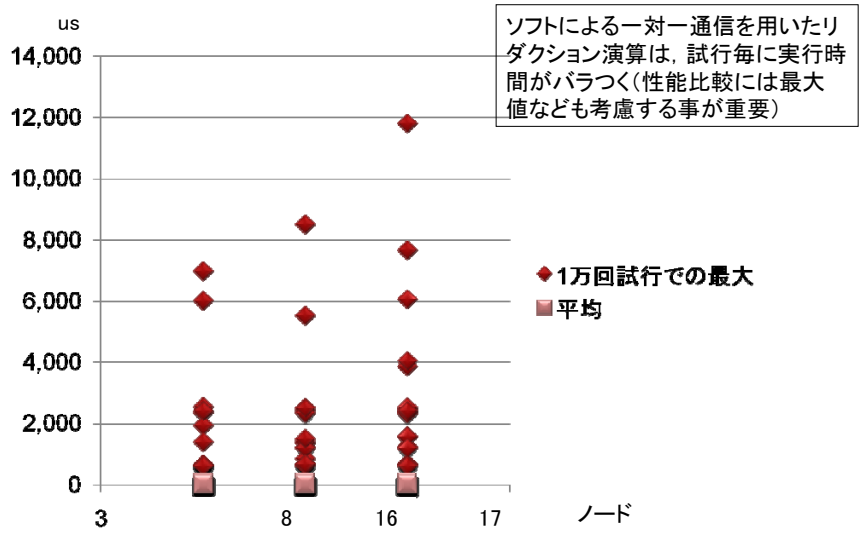
FUJITSU

| データ数 | 2K | 16K | 128K | 1M | 式 |
|---|---|---|---|---|---|
| IEEE倍精度 | 50 | 50 | 49 | 49 | 54-log(logN) |
| 2パス方式 | 43 | 40 | 37 | 34 | 54-log(N-1) |
| 1パス方式 | 54 | 54 | 54 | 54 | Min(81-log(N-1),54) |

- 有効ビット数は、1パス方式＞IEEE倍精度＞2パス方式となる
- 2パス方式では、2k/16k/128k/1Mデータについてそれぞれ64/67/70/73ビットの計算を実施すると想定
- 1パス方式では、154ビットの計算を実施すると想定（2Mノードまで適用可能）

# ソフトリダクションのバラツキ

FUJITSU




ソフトによる一対一通信を用いたリダクション演算は，試行毎に実行時間がバラつく（性能比較には最大値なども考慮する事が重要）



# 参考：オーバーヘッドのオーダー(BG/L)

FUJITSU

| 事象 | 時間のオーダー | 備考 |
|---|---|---|
| キャッシュミス | 100ns | |
| TLB ミス | 100ns | |
| IO 割り込み | 1us | ネットワーク含む |
| PTE ミス | 1us | “minor fault” |
| タイマー割り込み | 1us | |
| ページフォールト | 10us | “major fault” |
| スワップイン | 10ms | DISK IO有り |
| プリエンプション | 10ms | プロセス切り替え |

■出典：“Exploring Petascale Linux Operating Systems – ZeptoOS”
http://www.cs.unm.edu/~fastos/06meeting/ZeptoOS-PImtg-BostonUsenix-2006.pdf
■ZeptoOS は、Argonne National Lab. が開発中の BG/L ハード上で動作する軽量Linux