# 「K-scope」Fortran向けプログラム解析ツール
# 富士通性能解析ツール

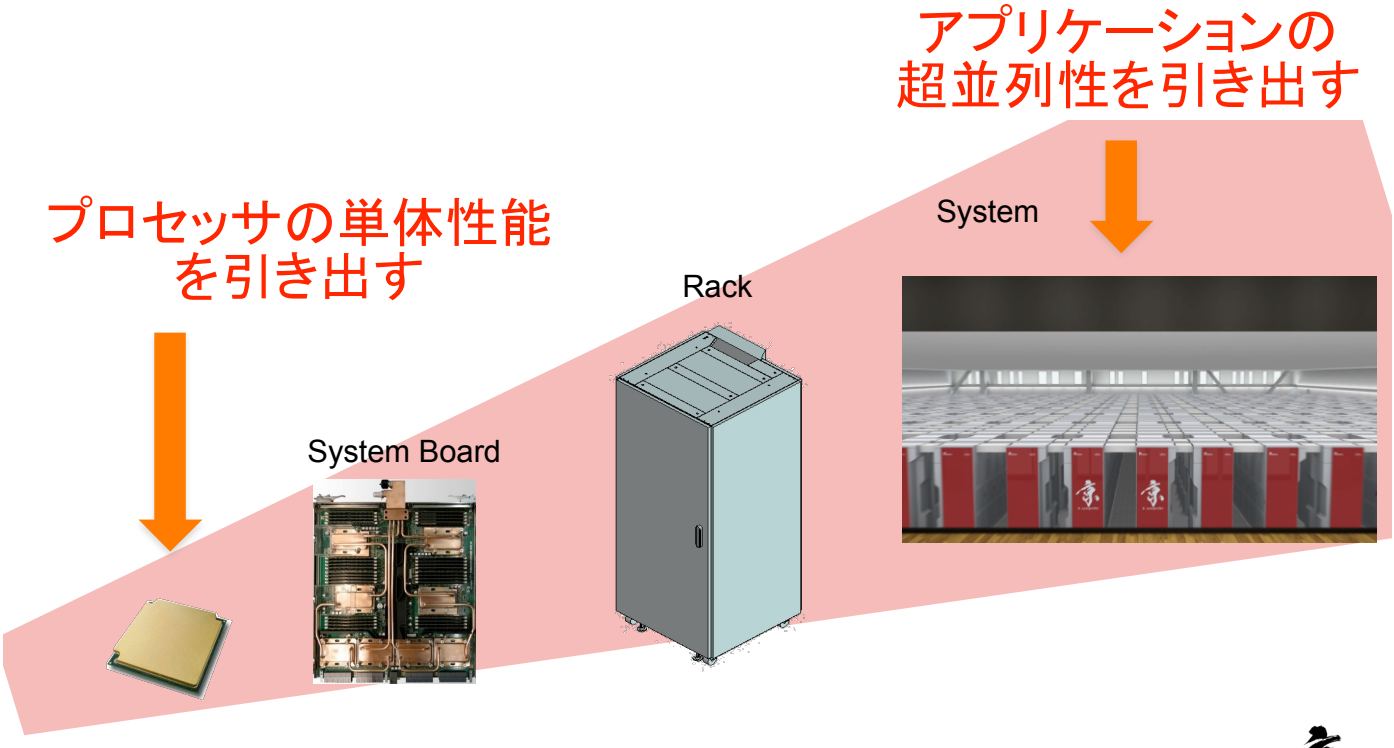
2014年3月

独立行政法人理化学研究所
計算科学研究機構　運用技術部門
ソフトウェア技術チーム　チームヘッド

南　一生
minami_kaz@riken.jp



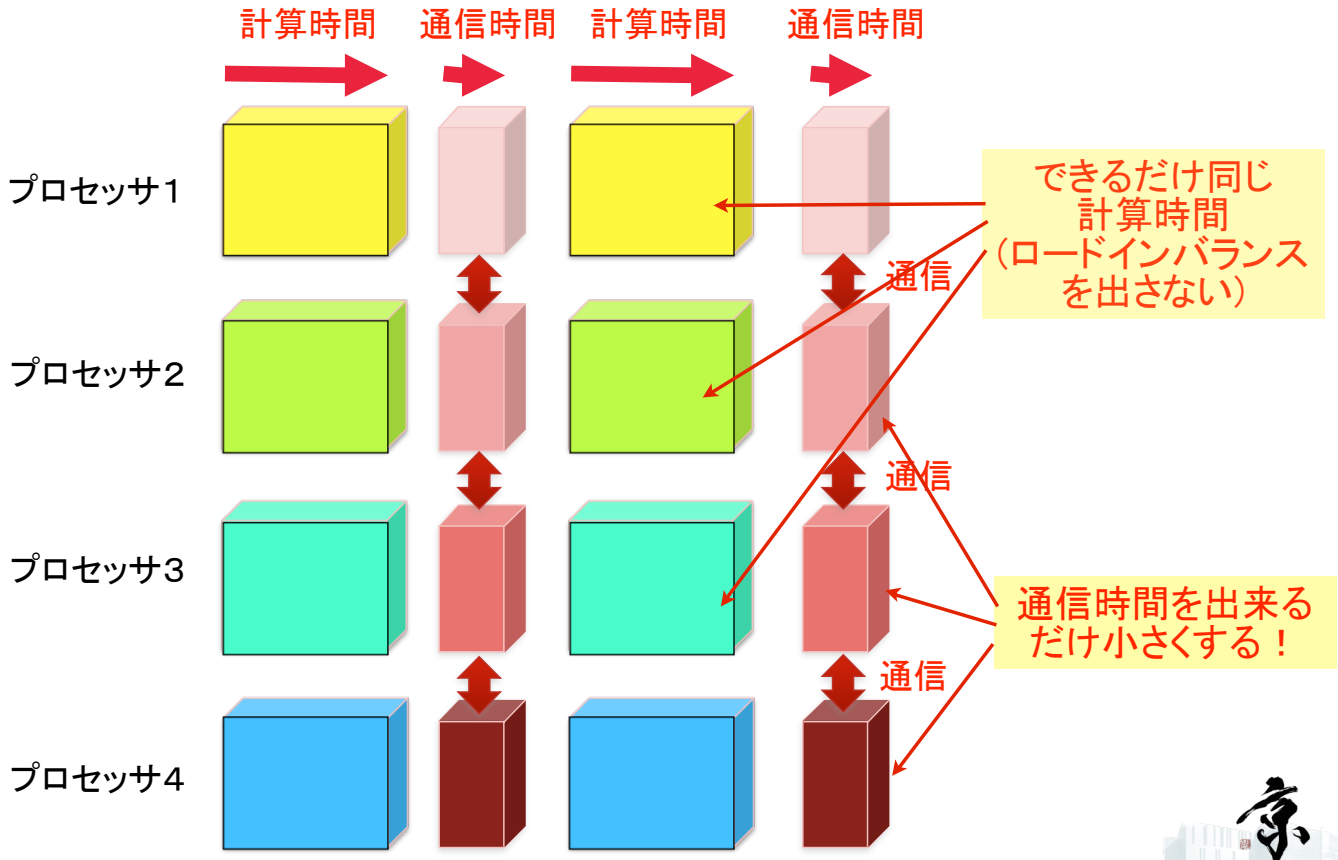
## 現代のスパコン利用の難しさ



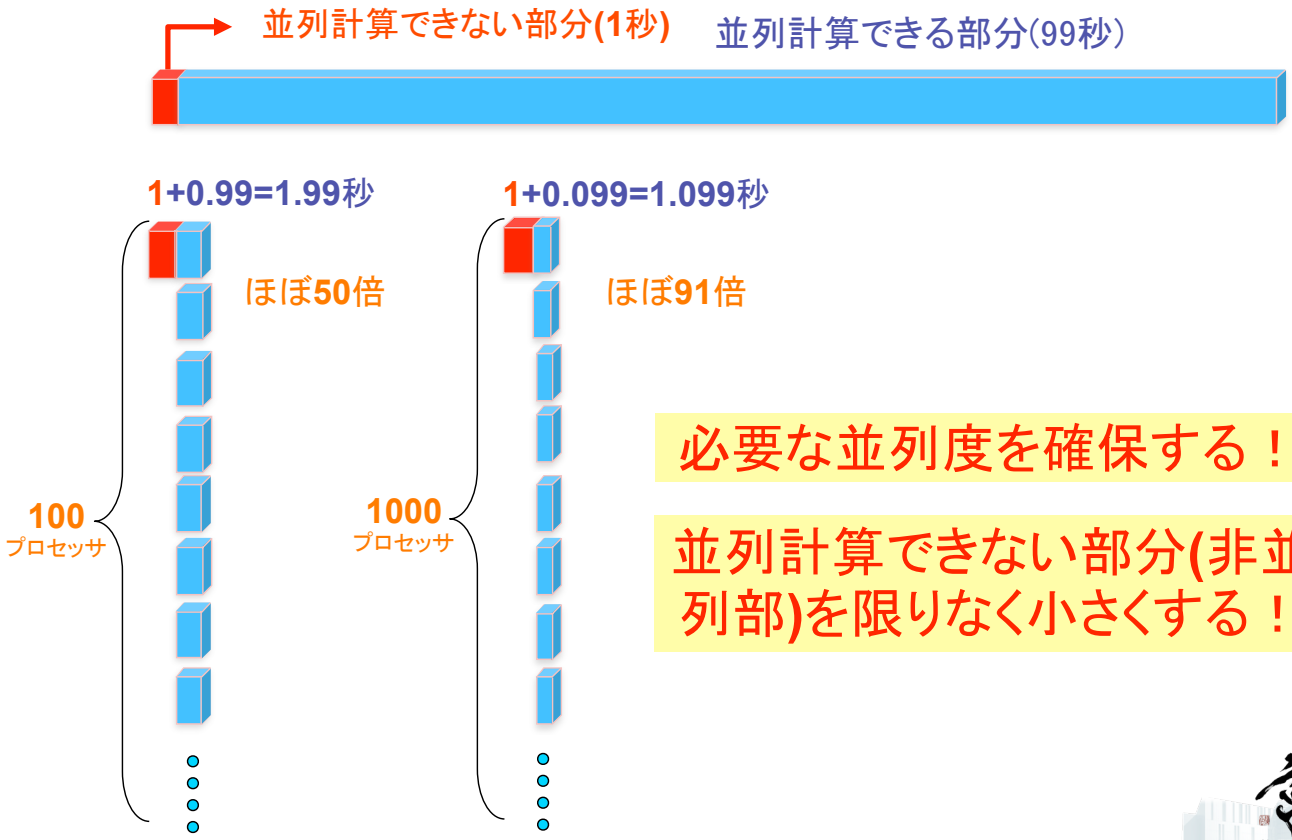
# 高並列化のための重要点



## 並列性能を引き出す(1)



# 並列性能を引き出す(2)




必要な並列度を確保する！

並列計算できない部分(非並列部)を限りなく小さくする！



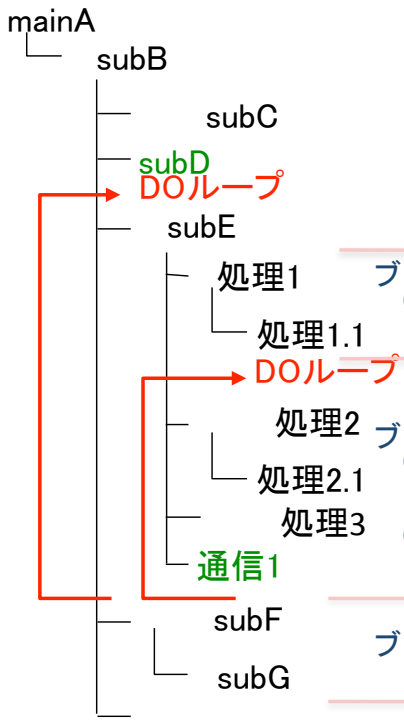
## アプリケーションを高並列にするためには どうしたら良いか

- ✓ 十分な並列度を得る並列化手法を採用する
- ✓ 非並列部分を最小化する
- ✓ 通信時間を最小化する
- ✓ ロードインバランスを出さない

最初に

アプリケーションの高並列阻害する要因を洗い出す事(並列特性分析)が重要

# 並列特性分析



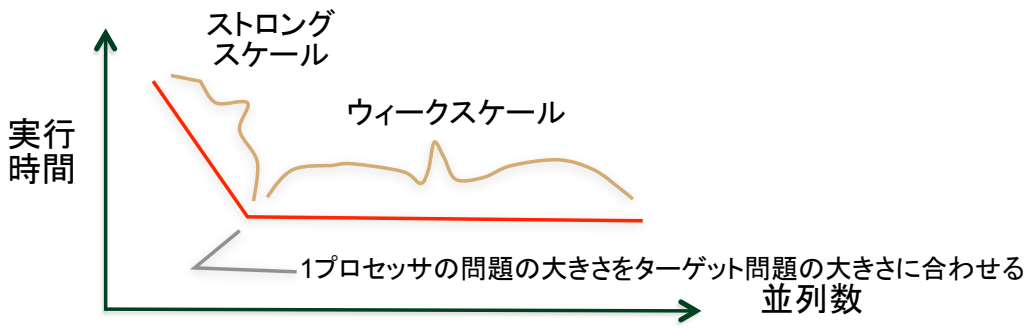

| | 実行時間・スケーラビリティ | 物理的処理内容 | 演算・通信特性 | 演算・通信見積り | カーネル |
|---|---|---|---|---|---|
| ブロック1（計算） | | | 部分並列 | Nに比例 | |
| ブロック2（計算） | | | 完全並列 | N**3に比例 | ○ |
| （通信） | | | 隣接通信 | 隣接面に比例 | ○ |
| ブロック3 | | | | | |



## 超高並列を目指した場合の留意点–ブロック毎に以下を評価する

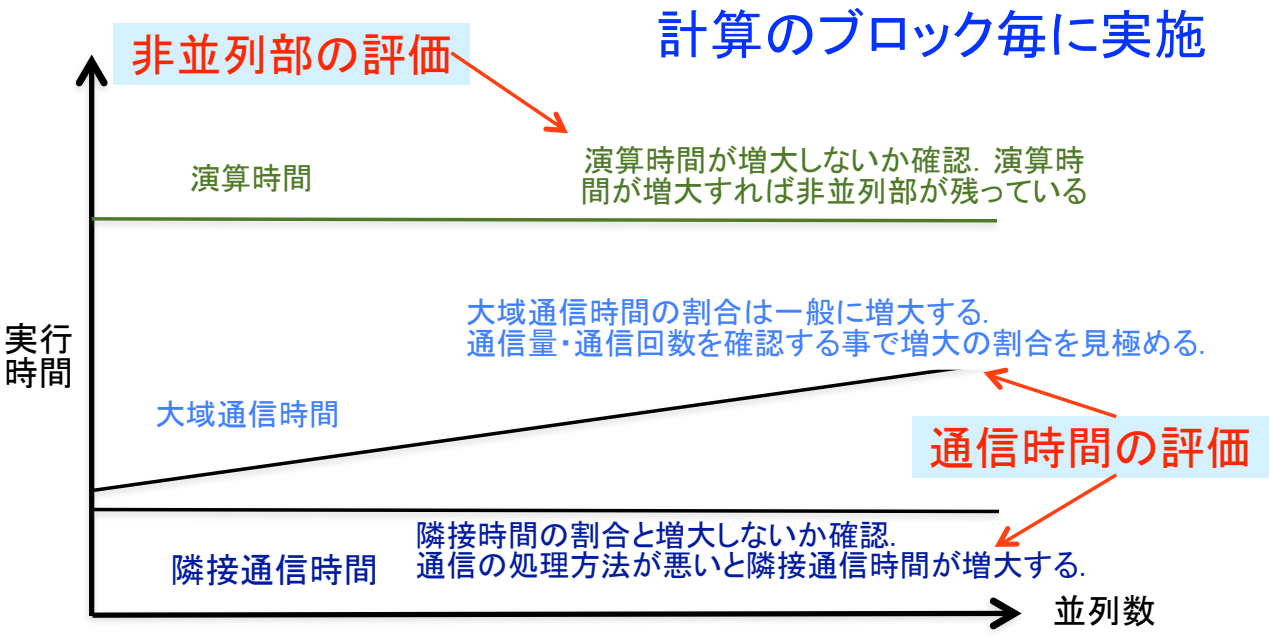
- 非並列部が残っていないか？残っている場合に問題ないか？
- 隣接通信時間が超高並列時にどれくらいの割合を占めるか？
- 大域通信時間が超高並列時にどれくらい増大するか？
- ロードインバランスが超高並列時に悪化しないか？

→ これらの評価が重要




ストロングスケーリング：全体の問題規模を一定にして並列数を増やし測定する方法
ウィークスケーリング　：1プロセッサで実行する問題規模を一定にし並列数を増やし測定する方

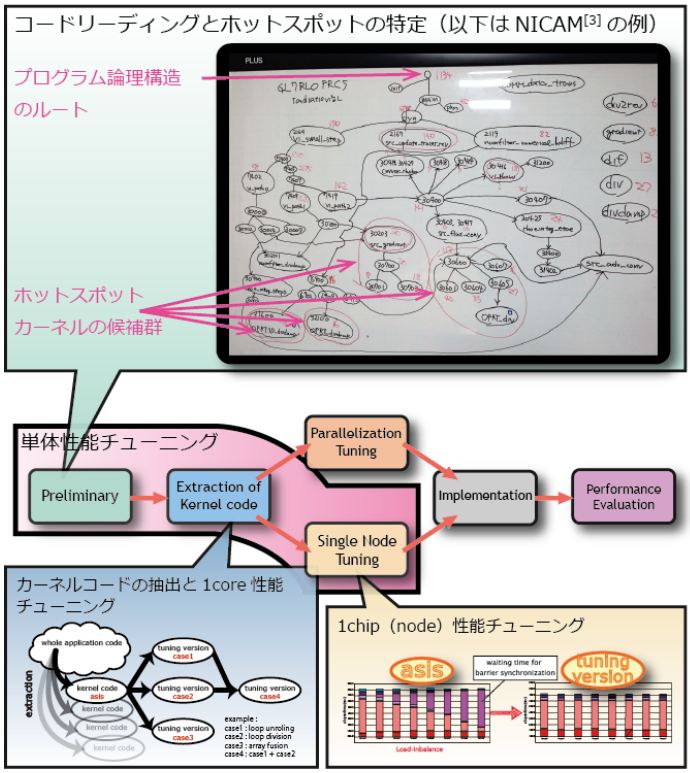
# ウィークスケーリング評価




Kscopeを使用した解析例

(NPB MGを題材に実際の解析例を示す)

# K-scopeとは？

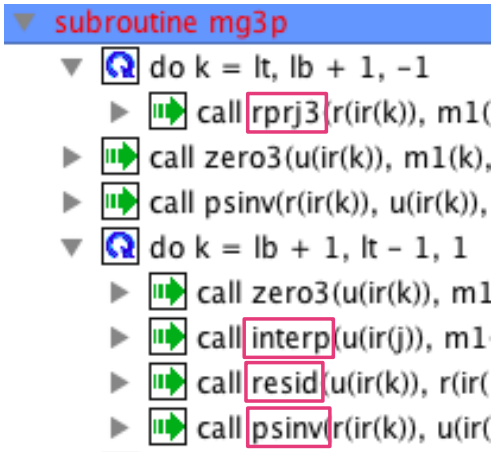



一般的に、プログラム構造を俯瞰的に理解する場合、ツリー構造に基づくトップダウンなアプローチが多用されます。いわゆるボトルネックとなるカーネルはそのツリー構造の末端に現れることが多く、本当に重要なルーチンかどうかはプロファイラのコスト情報等を用いて決定します。

“K-scope”は、Fortran 90 及び FORTRAN 77 コードのチューニングにおいて、ボトルネックとなり易いループ、分岐、およびプロシージャ呼び出しに代表されるプログラムの論理構造の可視化を軸とし、さらに「京」プロファイルデータに対応した、静的および動的解析に基づくプログラム構造解析支援ツールです。



# 並列特性分析(結果)



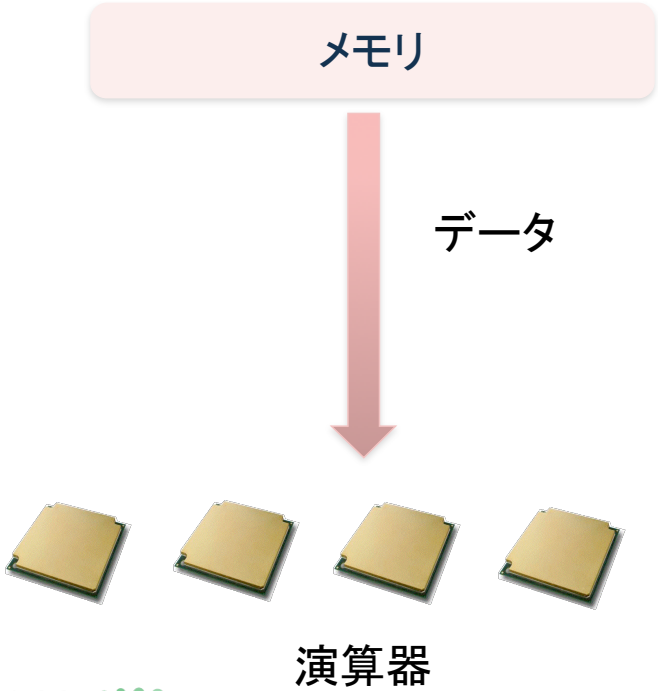

| 実行時間・スケーラビリティ | 物理的処理内容 | 演算・通信特性 | 演算・通信見積り | カーネル |
|---|---|---|---|---|
| 良好 | 制限補間 | 完全並列 | Nに比例 | ○ |
| | | | | |
| 良好 | 延長補間 | 完全並列 | Nに比例 | ○ |
| 良好 | 残差計算 | 完全並列 | Nに比例 | ○ |
| 良好 | 簡易求解 | 完全並列 | Nに比例 | ○ |

# 単体性能向上のための重要点



## プロセッサの単体性能を引き出す(1)

- かつては研究者やプログラマーは物理モデル式に忠実に素直にプログラミングすることが一般的であった



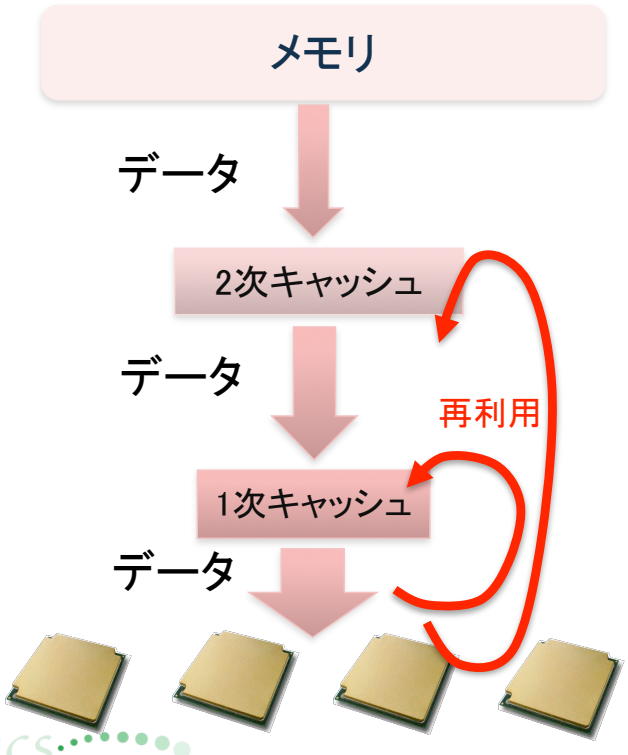

### メモリウォール問題

✓ 昔の計算機はメモリのデータ供給能力と演算器の能力がバランスしていた

- 現代の計算機は演算器の能力が高くなりメモリのデータ供給能力が相対的に不足している

# プロセッサの単体性能を引き出す(2)

## メモリウォール問題への対処




・データ供給能力の高いキャッシュを設ける
・キャッシュに置いたデータを何回も再利用し演算を行なう
・こうする事で演算器の能力を十分使い切る

「アプリケーションが要求するByte/Flop値が低い」タイプの計算(*1)

例えば行列行列積
(2N$^2$個のデータで2N$^3$個の演算可)

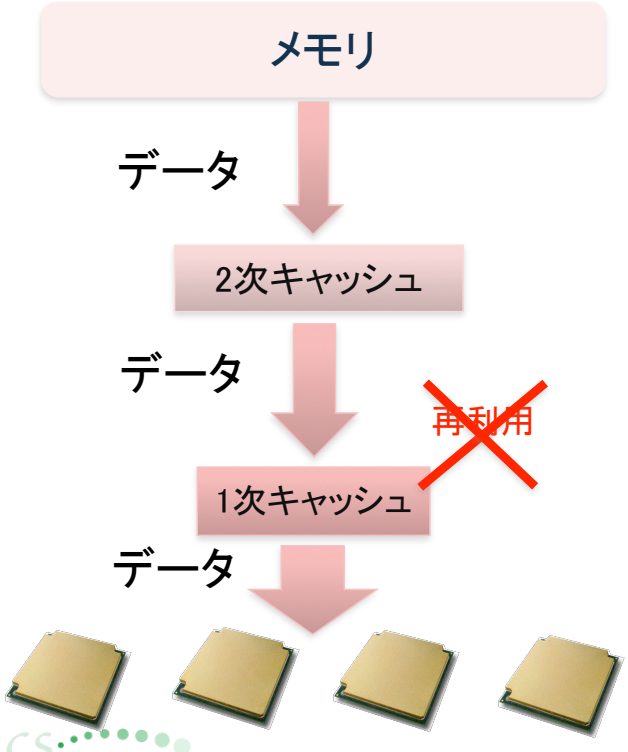
(*1)演算量(Flop) に比べデータの移動量(Byte)が小さい計算

キャッシュの有効利用



# プロセッサの単体性能を引き出す(3)

## とは言っても・・・・




・再利用出来ない問題もある
・こうなる演算器の能力を十分使い切る事が出来ない

「アプリケーションが要求するByte/Flop値が高い」タイプの計算(*2)

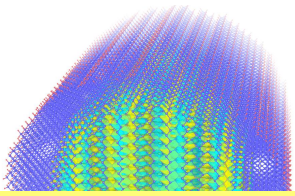
(*2)演算量(Flop) に比べデータの移動量(Byte)が大きい計算

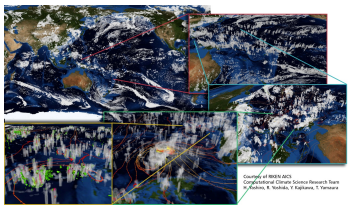
キャッシュの有効利用が難しい

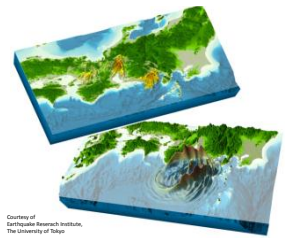
# 単体性能向上に向けての
# チューニング手順



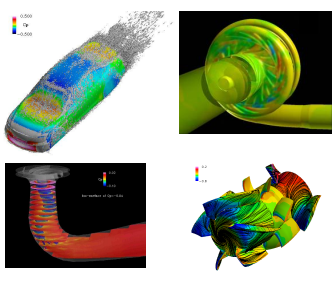
## アプリケーションの高性能化から分かったこと

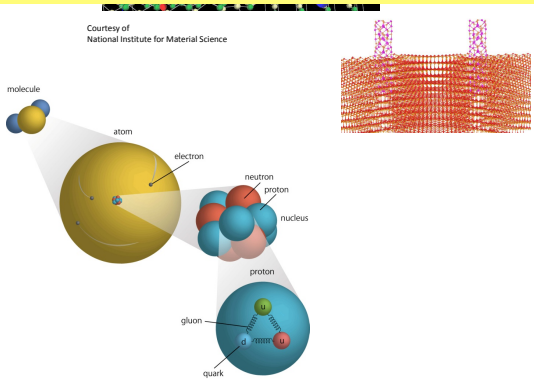






## CPU単体性能から見たアプリケーションの分類





# CPU単体性能上の分類

Hight

A Processor Performance

Low

| | |
|---|---|
| 1 | 行列行列積に書き換え可能 |
| 2 | キャッシュブロッキング可能 |
| 3 | 要求B/F値が低くループボディがシンプル |
| 4 | 要求B/F値が低いがループボディが複雑 |
| 5 | 要求B/F値が高い |
| 6 | 要求B/F値が高くリストアクセスを使用 |



# CPU単体性能の目安

DGEMMに書換え可能

30%～50%以上

キャッシュブロックに書換え可能

30%～50%以上

要求B/F値が低いステンシル計算は良い性能を得られる

30%～

要求B/F値が低い計算でもループボディが大きい計算は命令スケジューリング等の問題で思ったような性能が出ない場合が多い

いろいろで目安は云えない

要求B/F値が高いステンシル計算でもきちんとチューニングすればある程度キャッシュの有効利用もできメモリバンド幅ギリギリの性能が得られる

10%程度(詳細な見積り方法は後ほど)

要求B/F値が高くリストアクセスがある有限要素法のような疎行列とベクトルの積でもデータのリオーダリングによりベクトルを1次キャッシュに載せることによりメモリバンド幅ギリギリの性能が得られる

数%程度

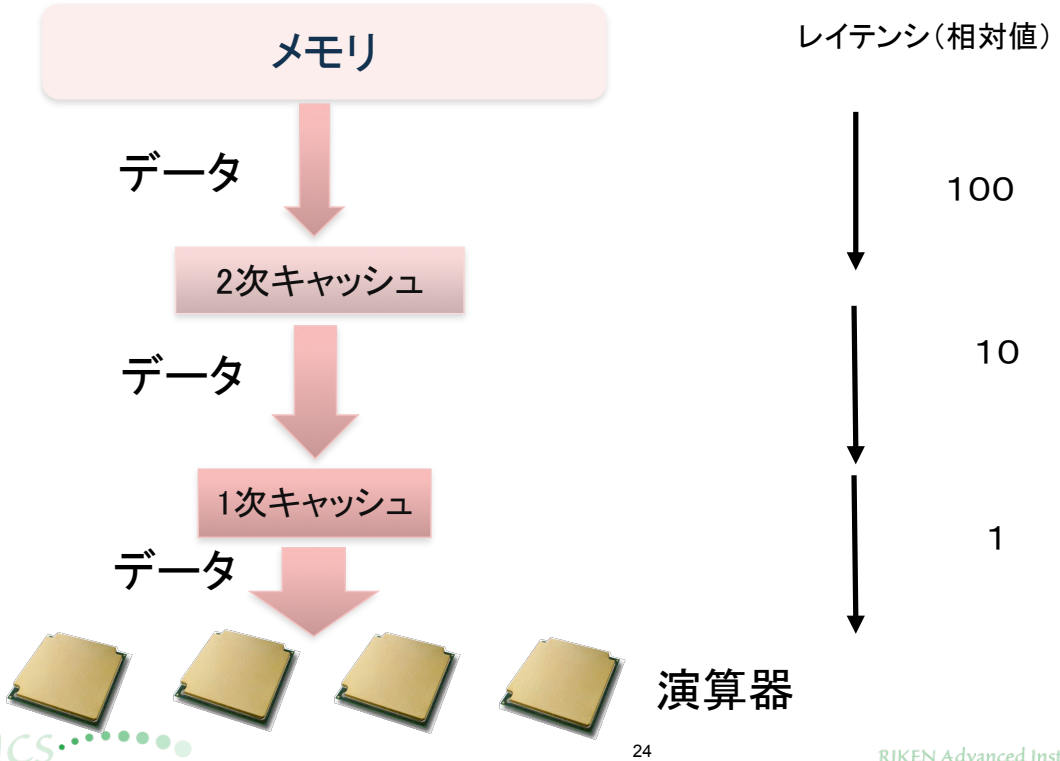
# CPU単体性能をあげるためには？

CPU内の複数コアでまずスレッド並列する事
は前提として

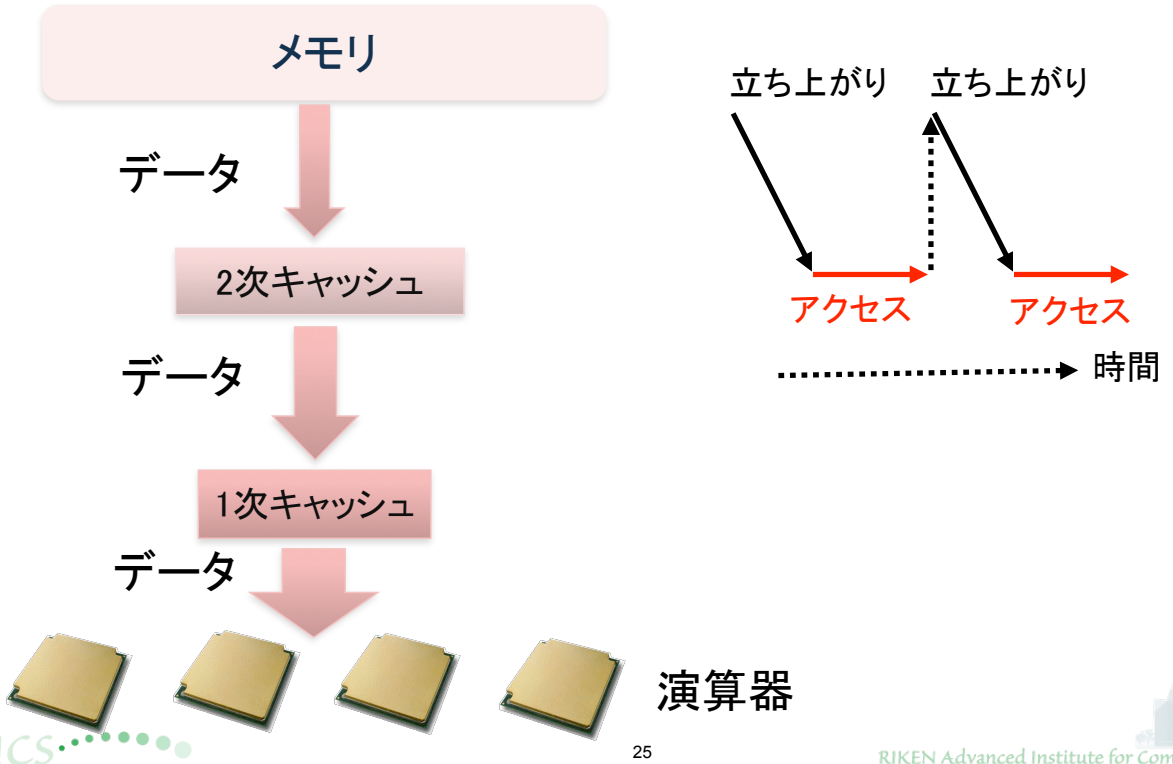
(1)プリフェッチの有効利用
(2)ラインアクセスの有効利用
(3)キャッシュの有効利用
(4)効率の良い命令スケジューリング
(5)演算器の有効利用



# レイテンシ(アクセスの立ち上がり)

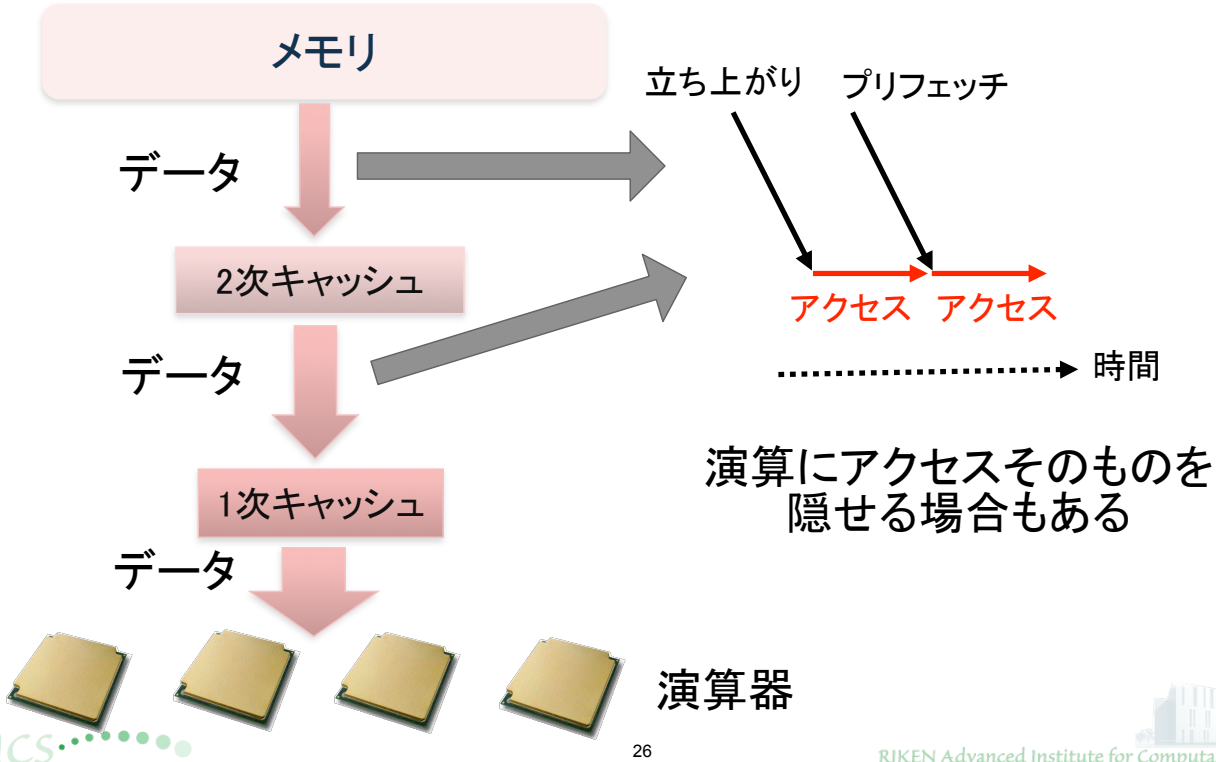


# レイテンシ(アクセスの立ち上がり)



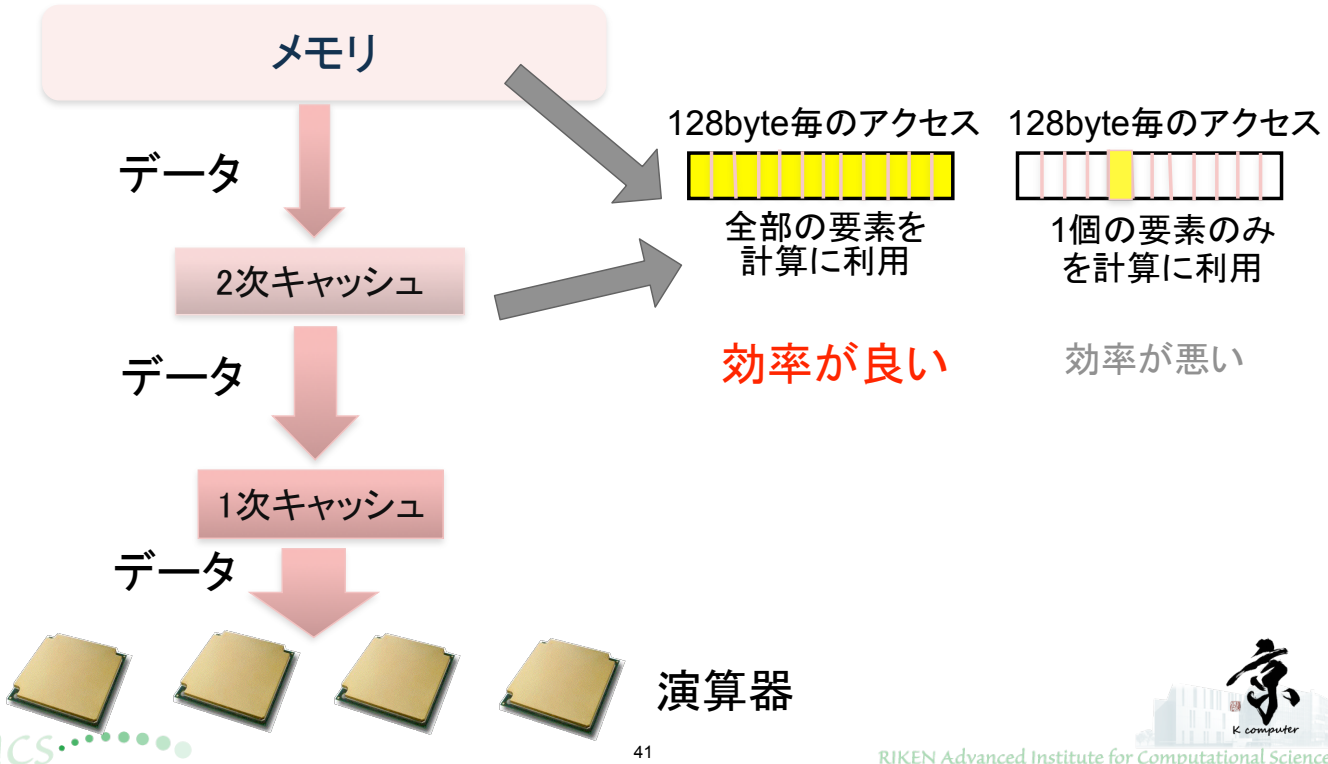



# (1)プリフェッチの有効利用




演算にアクセスそのものを
隠せる場合もある

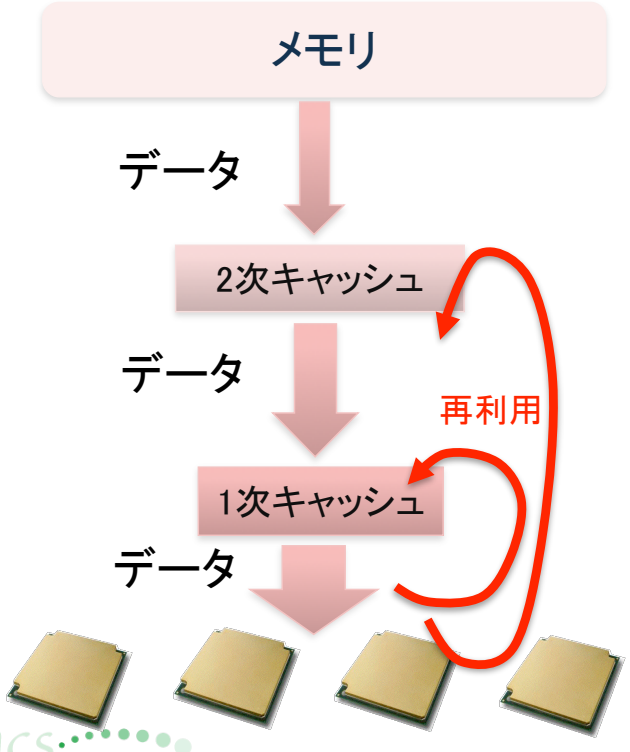
# (2)ラインアクセスの有効利用





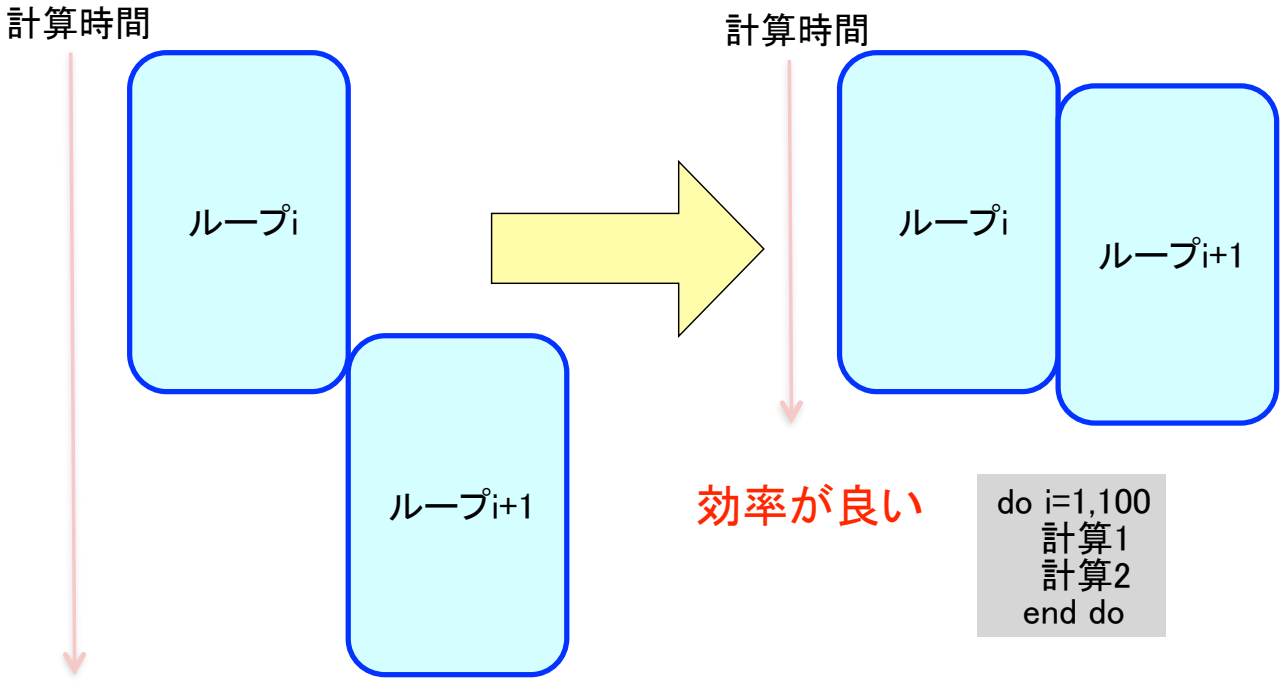



# (3)キャッシュの有効利用




・データ供給能力の高いキャッシュを設ける
・キャッシュに置いたデータを何回も再利用し演算を行なう
・こうする事で演算器の能力を十分使い切る

# (4)効率の良い命令スケジューリング






# (5)演算器の有効利用

乗算と加算を4個同時に計算可能

( 1 + 1 ) × 4 = 8

この条件に
近い程高効率

**1コアのピーク性能：8演算×2GHz =16G演算/秒**

# 要求B/F値と性能の関係



## 高い性能を得るための要素と要求B/F値の関係

要求するB/Fが小さいアプリケーションについて
・原理的にキャッシュの有効利用が可能
・まずデータをオンキャッシュにするコーディング:(3)が重要

・つぎに2次キャッシュのライン上のデータを有効に利用するコーディング:(2)が重要
・それが実現できた上で(4)(5)が重要

キャッシュブロック:
n個のデータをキャッシュに置いてデータの使い回しによりnの2乗回の計算をする

# 高い性能を得るための要素と要求B/F値の関係

要求するB/Fが大きいアプリケーションについて
・メモリバンド幅を使い切る事が大事
・一番重要なのは(1)(2)
　(1)(2)がうまく行ってないと不当な性能劣化が起こっている。まともな状態にすることが重要。

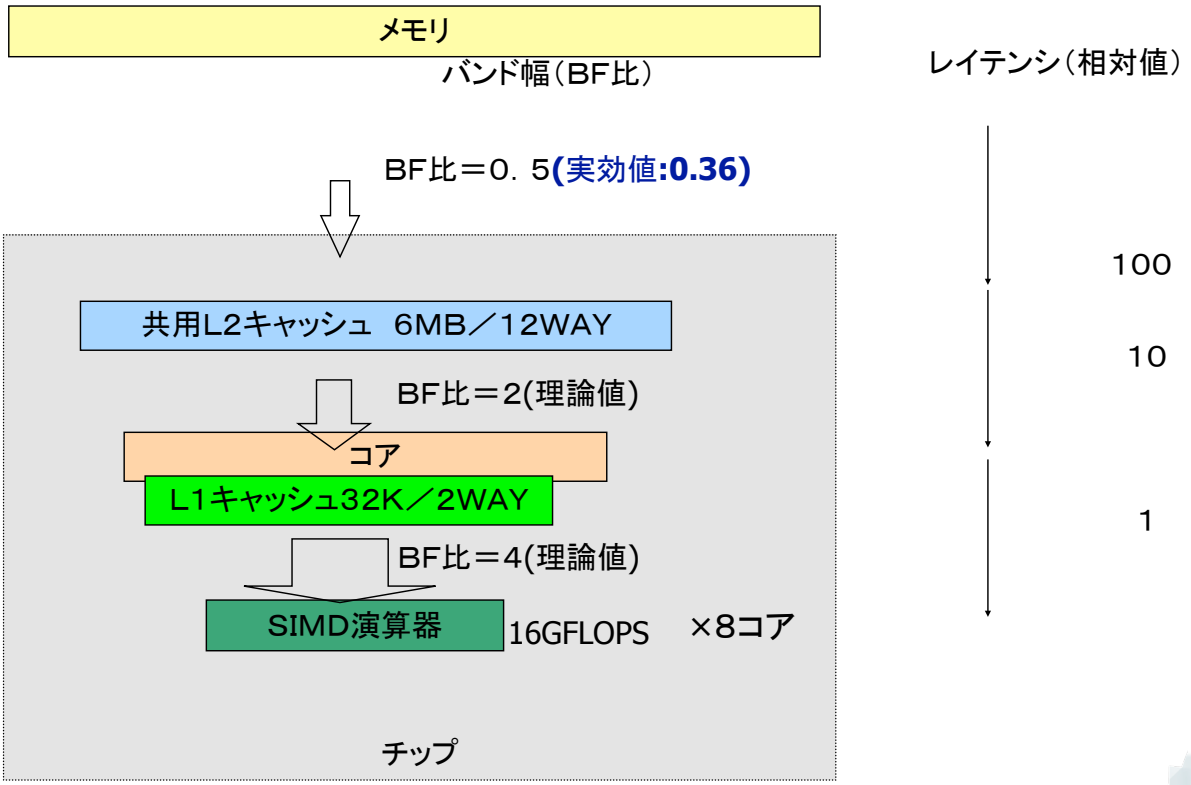
・次にできるだけオンキャッシュする(3)が重要
・これら(1)(2)(3)が満たされ計算に必要なデータが演算器に供給された状態で、それらのデータを十分使える程度に(4)のスケジューリングができて、さらに(5)の演算器が有効に活用できる状態である事が必要



# 性能予測・チューニング手法

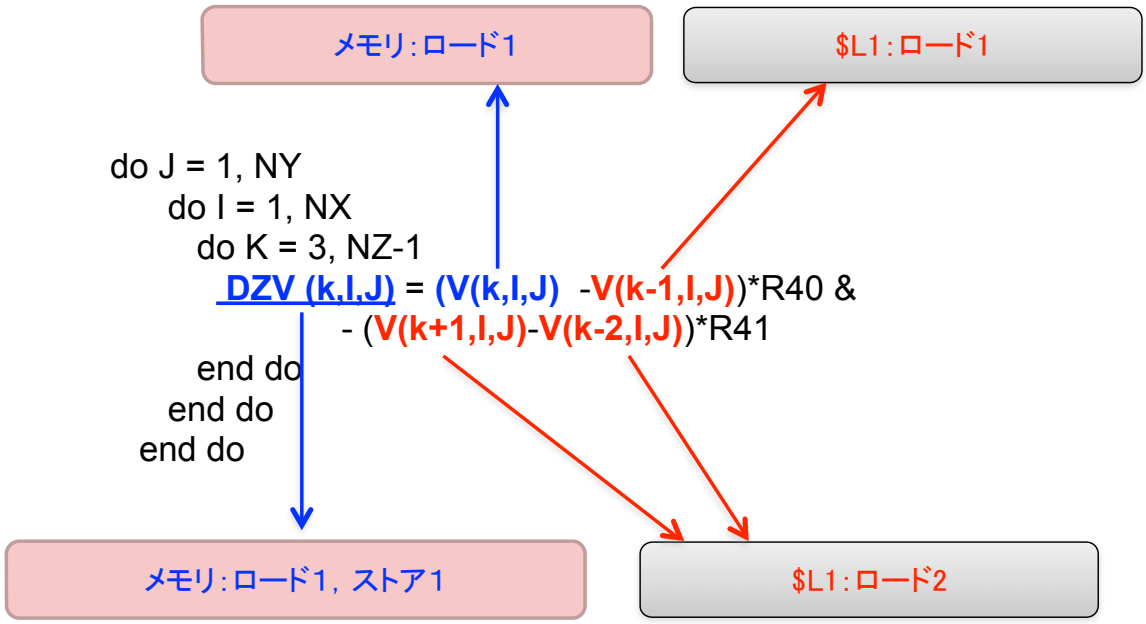
## (要求B/F値が高い計算について)

# ベースとなる性能値



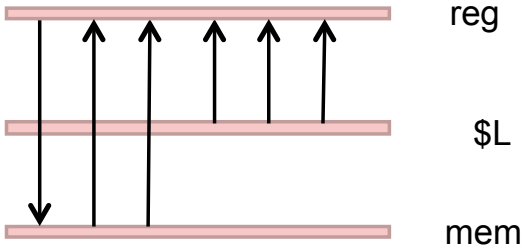



# メモリとキャッシュアクセス(1)



# メモリとキャッシュアクセス(2)

```
do J = 1, NY
    do I = 1, NX
      do K = 3, NZ-1
        DZV (k,I,J) = (V(k,I,J)  -V(k-1,I,J))*R40 &
                    - (V(k+1,I,J)-V(k-2,I,J))*R41
      end do
    end do
  end do
```




| | Store | Load | バンド幅比 ($L1) | データ移動時間の比(L1) | バンド幅比 ($L2) | データ移動時間の比(L2) |
|---|---|---|---|---|---|---|
| $L | 1 | 5 | 11.1 (8*64G/s) | 0.5= 6/11.1 | 5.6 (256G/s) | 1.1=6/5.6 |
| M | 1 | 2 | 1(46G/s) | 3=3/1 | 1 (46G/s) | 3=3/1 |

データ移動時間の比を見るとメモリで律速される
→ メモリアクセス変数のみで考慮すれば良い。



# 性能見積り

```
do J = 1, NY
    do I = 1, NX
      do K = 3, NZ-1
        DZV (k,I,J) = (V(k,I,J)  -V(k-1,I,J))*R40 &
                    - (V(k+1,I,J)-V(k-2,I,J))*R41
      end do
    end do
  end do
```

- 最内軸(K軸)が差分
- 1ストリームでその他の3配列は$L1に載っており再利用できる。

| 要求B/F | 12/5 = 2.4 |
|---|---|
| 性能予測 | 0.36/2.4 = 0.15 |
| 実測値 | 0.153 |

**要求Byteの算出**:

1store,2loadと考える

4x3 = 12byte

**要求flop:**

add：3 mult：2 = 5

# 空間微分X方向の計算a)b)(2次元目の差分)

```
do J = 1, NY
   do I = 1, NX
     do K = 1, NZ
       DXV (k,I,J) = (V(k,I,J)  -V(k,I-1,J))*R40&
                   - (V(k,I+1,J)-V(k,I-2,J))*R41
     end do
   end do
 end do
```

- 第2軸(I軸)が差分
- 1ストリームでその他の3配列は$L1or$L2に載っており再利用できる
- 従って1次元目が差分のパターンと同じ性能になる

**要求Byteの算出**:

P12より、メモリコストだけを考慮する。

1sore,2loadと考える

4x3 = 12byte

**要求flop:**

add：3 mult：2 = 5

| 要求B/F | 12/5 = 2.4 |
|---|---|
| 性能予測 | 0.36/2.4 = 0.15 |
| 実測値 | 0.135 |

- 実測値が13.5%と少し低い
- 実測したメモリバンド幅は42.9GB/sec
- この値で性能予測をすると14.0%となる
- 14.0%に比較すれば13.5%は良い値
- L2キャッシュ負荷の増大によりメモリバンド幅が下がった可能性(京特有の現象)



# 空間微分Y方向の計算a)b)(3次元目の差分)

```
do J = 1, NY
   do I = 1, NX
     do K = 1, NZ
       DYV (k,I,J) = (V(k,I,J)  -V(k,I,J-1))*R40 &
                   - (V(k,I,J+1)-V(k,I,J-2))*R41
     end do
   end do
 end do
```

- 第3軸が差分 →　再利用性なし

**要求flop:**

add：3 mult：2 = 5

**要求Byteの算出:**

1store/5loadより

(5+1) * 4byte = 24

| 要求B/F | 24/5 = 4.8 |
|---|---|
| 性能予測 | 0.36/4.8= 0.075 |
| 実測値 | 0.076 |

# 空間微分Y方向の計算a)b)
# (3次元目をcyclicでスレッド並列化)

```
!$OMP DO SCHEDULE(static,1),PRIVATE(I,J,K)
 do J = 1, NY
    do I = 1, NX
      do K = 1, NZ
        DYV (k,I,J) = (V(k,I,J)  -V(k,I,J-1))*R40 &
                    - (V(k,I,J+1)-V(k,I,J-2))*R41
      end do
    end do
  end do
```

キャッシュに載せる

- 第3軸をcyclic分割 → 1ストリームで3配列がL2に乗る(説明次項)
- 性能が２倍になる

| 要求B/F | 12/5 = 2.4 |
|---|---|
| 性能予測 | 0.36/2.4 = 0.15 |
| 実測値 | 0.136 |

**要求Byteの算出**:

1sore,2loadと考える

4x3 = 12byte

**要求flop:**

add：3 mult：2 = 5



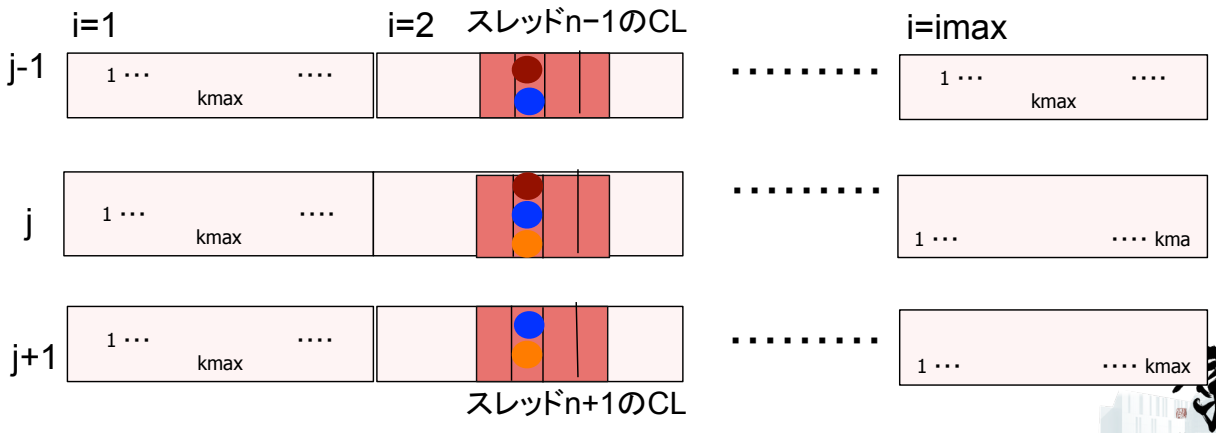
# (cyclic分割スレッド並列の説明)

プログラム例

```
Do j=1,jmax
  do i=1,imax
    do k=1,kmax
      a(k,i,j)=・・・c0*v(k,i,j-1)+c1* v(k,i,j)+c2* v(k,i,j+1)・・・
    end do
  end do
end do
```

● :スレッドn−1で参照するデータ
● :スレッドnで参照するデータ
● :スレッドn+1で参照するデータ

キャッシュライン(CL)



# 空間微分Y方向の計算a)b)
# (ZXYループ融合cyclicスレッド並列)

```
!$OMP DO SCHEDULE(static,1)
   do J = 1, NY
     do I = 1, NX
       do K = 3, NZ-1
         DZV (k,I,J) = (V(k,I,J)  -V(k-1,I,J))*R42 &
                     - (V(k+1,I,J)-V(k-2,I,J))*R43
         DXV (k,I,J) = (V(k,I,J)  -V(k,I-1,J))*R40&
                     - (V(k,I+1,J)-V(k,I-2,J))*R41
         DYV (k,I,J) = (V(k,I,J)  -V(k,I,J-1))*R40 &
                     - (V(k,I,J+1)-V(k,I,J-2))*R41
       end do
     end do
   end do
```

**要求B/F値を下げる**
**キャッシュに載せる**

- K,I,J軸差分のループを融合することにより、V(K,I,J)のロードを共通化でき、プログラムの要求B/F比を下げる。

**要求Byteの算出**:

Store 3 +4 load と考えると、

(3+4)*4 = 28byte

**要求flop:**

add：9 mult：6 = 15

| 要求B/F | 28/15 = 1.86 |
|---|---|
| 性能予測 | 0.36/1.86 = 0.19 |
| 実測値 | 0.177 |



# 速度時間積分の計算 e)

オリジナルコードの結果

| 要求B/F | 72/52=1.38 |
|---|---|
| 性能予測 | 0.36/1.38 = 0.26 |
| 実測値 | 0.240 |

# 基本プロファイラを用いた性能ボトルネック解析と改善

## (要求B/F値が高いMGを例として)




## 目的

- アプリケーションの計算機的特性を把握し、性能の妥当性を判定する。
- アプリケーションの性能ボトルネックが通信部なのか演算部なのかを素早く、正しく切り分ける。
- 性能ボトルネックが演算ループであれば、そのループのスレッド並列化、simd化、ソフトウェアパイプライニングまでを確認する。

# Compile/Execution/Analyze to MG

- Compile (Class = C, 4process)
  $cd $Hands/FJTools
  $Make suite
  (Option : -Kfast,parallel,ocl,openmp,optmsg=2 -Qt)

- Execution
  $cd bin
  $pjsub --interact ./run_MGc4.sh

- Analyze
  $fipppx -A -d ./data_c4 -Icall,hwm,balance,cpu,src:../MG
  -o mg.C.4.fipp.txt



# 第1着眼点

アプリケーション全体の
✓MFLOPS/PEAK(%)
- 計算手法の目標値に比べてどうか？
- スケールするか？

✓Mem throughput
- 計算手法の目標値に比べてどうか?
- スケールするか？

✓プロセス間のロードインバランス
- 各プロセスで実行時間や性能にバラつきはないか？
- スケールさせたときにどうか？

これらの情報から
性能チューニングが必要かを判断する

# 第1着眼点でMGの結果を見る（1）

$cat mg.C.4.fipp.txt

Application - performance monitorsで
MFLOPS/PEAK(%), Mem throughputを確認

10%には程遠く、
なんだか低い…

プロセスのロード
インバランスは
発生していない

1chipあたり46Gの
最大値に比べて
なんだか低い…

```
Performance monitor event

*******************************************************************************
    Application - performance monitors
*******************************************************************************

    Elapsed(s)        MFLOPS MFLOPS/PEAK(%)          MIPS    MIPS/PEAK(%)
-------------------------------------------------------------------------------
    16.6556      10361.0515          2.0236      18605.9895          7.2680    Application
-------------------------------------------------------------------------------
    16.6556       2590.2629          2.0236       4651.6009          7.2681    Process    0
    16.6052       2598.1206          2.0298       4668.2023          7.2941    Process    2
    16.5407       2608.2634          2.0377       4683.4974          7.3180    Process    1
    16.5213       2611.3249          2.0401       4686.9329          7.3233    Process    3

    Elapsed(s)  Mem throughput _chip(GB/S)  Mem throughput /PEAK(%)      SIMD(%)
-------------------------------------------------------------------------------
    16.6556                    32.3398                  12.6327          48.7947    Application
-------------------------------------------------------------------------------
    16.6556                     8.0885                  12.6382          48.7936    Process    0
    16.6052                     8.1070                  12.6672          48.7676    Process    2
    16.5407                     8.1357                  12.7120          48.7981    Process    1
    16.5213                     8.1551                  12.7423          48.8196    Process    3
```



# 第1着眼点でMGの結果を見る（2）

$cat mg.D.32.fipp.txt(C.4のWeak Scale)

Application - performance monitorsで
MFLOPS/PEAK(%), Mem throughputを確認

4並列：2.0%
32並列：2.0%

プロセスのロード
インバランスも
スケールしても問題なし

4並列：8.0G/s
32並列：8.0G/s

```
Performance monitor event

*******************************************************************************
    Application - performance monitors
*******************************************************************************

    Elapsed(s)        MFLOPS MFLOPS/PEAK(%)          MIPS    MIPS/PEAK(%)
-------------------------------------------------------------------------------
    39.3654      83044.5238          2.0275     149003.2766          7.2756    Application
-------------------------------------------------------------------------------
    39.3654       2595.1412          2.0275       4654.6126          7.2728    Process    0
    39.3575       2595.6627          2.0279       4650.9112          7.2670    Process    7
    39.3546       2595.8520          2.0280       4656.6897          7.2761    Process    8
                        :

    Elapsed(s)  Mem throughput _chip(GB/S)  Mem throughput /PEAK(%)      SIMD(%)
-------------------------------------------------------------------------------
    39.3654                   258.7562                  12.6346          49.3192    Application
-------------------------------------------------------------------------------
    39.3654                     8.0773                  12.6208          49.3303    Process    0
    39.3575                     8.0573                  12.5895          49.3647    Process    7
    39.3546                     8.0544                  12.5850          49.3071    Process    8
                                  :
```

# 第1着眼点による判定

アプリケーション全体の

✓MFLOPS/PEAK(%)
- 計算手法の目標値に比べてどうか？
- スケールするか？

✓Mem throughput
- 計算手法の目標値に比べてどうか?
- スケールするか？

✓プロセス間のロードインバランス
- 各プロセスで実行時間や性能にバラつきはないか？
- スケールさせたときにどうか？

CPU単体チューニングが必要



# 第2着眼点

アプリケーション全体の

✓高コスト部の把握
- 演算ルーチンか、通信ルーチンか
- スケールさせたときにどうか？

✓高コスト部の状態の把握
- 演算ルーチン
  最適化(スレッド並列化、simd化、ソフトウェアパイプライニング)はどうか？
- 通信ルーチン
  隣接か集団か？スケールさせたときにどうか？
  （通信の解析には詳細プロファイラが必要）

# 第2着眼点でMGの結果を見る（1）

$cat mg.C.4.fipp.txt

Procedures profileで
高コスト部、演算と通信のバランスを確認

全体のコストに対し、
スレッドバリアが12.5%、
通信が0.27%
演算の割合が大きい！

```
Procedures profile

*************************************************************************
    Application - procedures
*************************************************************************

          Cost        %       Barrier        %        MPI        %   Start      End
-------------------------------------------------------------------------------------
          3942  100.0000           493  12.5063         11   0.2790      --       --   Application
-------------------------------------------------------------------------------------
          1881   47.7169           199  10.5795          0   0.0000     737      754   resid._PRL_1_
           885   22.4505           104  11.7514          0   0.0000     666      683   psinv._PRL_1_
           382    9.6905            25   6.5445          0   0.0000     913      938   interp._PRL_1_
           278    7.0523            24   8.6331          0   0.0000     831      852   rprj3._PRL_1_
           194    4.9214             0   0.0000          0   0.0000    1236     1238   ready._PRL_1_
            85    2.1563            80  94.1176          0   0.0000     707      774   resid._PRL_1_
            47    1.1923             0   0.0000          0   0.0000    9558            PRL_1_
            36    0.9132            3   8.8889          0   0.0000
            26    0.6596            0   0.0000
            21    0.5327            0   0.0000
```

上位4ルーチンは全て
演算であり、
全体の86.7%を占める。



# 第2着眼点でMGの結果を見る（2）

$cat mg.D.32.fipp.txt(C.4のWeak Scale)

Procedures profileで
高コスト部、演算と通信のバランスを確認

全体のコストに対し、
スレッドバリアが14.6%、
通信が0.35%
4並列と比べて傾向が変わらない。

```
Procedures profile

*************************************************************************
    Application - procedures
*************************************************************************

          Cost        %       Barrier        %        MPI        %   Start      End
-------------------------------------------------------------------------------------
         75024  100.0000         11017  14.6846        266   0.3546      --       --   Application
-------------------------------------------------------------------------------------
         35408   47.1956          4150  11.7205          0   0.0000     736      753   resid._PRL_1_
         17197   22.9220          2205  12.8220          0   0.0000     666      683   psinv._PRL_1_
          7262    9.6796           690   9.5015          0   0.0000     911      936   interp._PRL_1_
          5994    7.9894           611  10.1935          0   0.0000     829      850   rprj3._PRL_1_
          3778    5.0357             0   0.0000          0   0.0000    1234     1236   ready._PRL_1_
          1889    2.5179          1810  95.8179          0   0.0000     707      772   resid._PRL_1_
           947    1.2623           905  95.5649          0   0.0000     637            PRL_1_
           496    0.6611             0   0.0000        171            758
           356    0.4745           349  98.0337          0                00
           299    0.3985            15   5.0167         21   7.0234
```

上位4ルーチンの順位や
割合が
4並列時と変わらない。

47%を占める演算ルーチンの最適化状況を確認する

# （ここまでの）第2着眼点による判定

アプリケーション全体の
✓高コスト部の把握
- 演算ルーチンか、通信ルーチンか　　演算ルーチン
- スケールさせたときにどうか？　　同一傾向

✓高コスト部の状態の把握
- 演算ルーチン
  最適化(スレッド並列化、simd化、ソフトウェアパイプライニング)はどうか？
- 通信ルーチン
  隣接か集団か？スケールさせたときにどうか？
  （通信の解析には詳細プロファイラが必要）



# 第2着眼点でMGの結果を見る（3）

$cat ../MG/mg.lst

.lstファイルで
高コスト部の最適化状況を確認する

```
      735     1                  do i3=2,n3-1
      736     2       8             do i2=2,n2-1
                          <<< Loop-information Start >>>
                          <<<  [PARALLELIZATION]
                          <<<    Standard iteration count: 373
                          <<<  [OPTIMIZATION]
                          <<<    SIMD
                          <<<    SOFTWARE PIPELINING
                          <<< Loop-information  End >>>
      737     3  pp   4v                do i1=1,n1
      738     3   p   4v                   u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
      739     3                     >              + u(i1,i2,i3-1) + u(i1,i2,i3+1)
      740     3   p   4v                   u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
      741     3                     >              + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
      742     3   p   4v                enddo
                          <<< Loop-information Start >>>
                          <<<  [PARALLELIZATION]
                          <<<    Standard iteration count: 400
                          <<<  [OPTIMIZATION]
                          <<<    SIMD
                          <<<    SOFTWARE PIPELINING
                          <<< Loop-information  End >>>
      743     3  pp   6v                do i1=2,n1-1
      744     3   p   6v                   r(i1,i2,i3) = v(i1,i2,i3)
      745     3                     >                 - a(0) * u(i1,i2,i3)
      746     3                   c-----------------------------------------------------
      747     3                   c  Assume a(1) = 0      (Enable 2 lines below if a(1) not= 0)
      748     3                   c-----------------------------------------------------
      749     3                   c    >                  - a(1) * ( u(i1-1,i2,i3) + u(i1+1,
      750     3                   c    >                           + u1(i1) )
      751     3                   c-----------------------------------------------------
      752     3                     >                  - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
      753     3                     >                  - a(3) * ( u2(i1-1) + u2(i1+1) )
      754     3   p   6v                enddo
      755     2       8             enddo
      756     1                  enddo
```

Simd/SWPL：OK
最内で並列化
されている。

スレッドバリアの10.5%も
これが原因？

# 第2着眼点からの改善

```
      735                        !$OMP PARALLEL DO PRIVATE(u1,u2)
      736     1   p                    do i3=2,n3-1
      737     2   p                       do i2=2,n2-1
                                  <<< Loop-information Start >>>
                                  <<<  [OPTIMIZATION]
                                  <<<    SIMD
                                  <<<    SOFTWARE PIPELINING
                                  <<< Loop-information  End >>>
      738     3   p   4v                     do i1=1,n1
      739     3   p   4v                        u1(i1) = u(i1,i2-1,i3) + u(i1,i
      740     3                        >                + u(i1,i2,i3-1) + u(i1,
      741     3   p   4v                        u2(i1) = u(i1,i2-1,i3-1) + u(i
      742     3                        >                + u(i1,i2-1,i3+1) + u(i
      743     3   p   4v                     enddo
                                  <<< Loop-information Start >>>
                                  <<<  [OPTIMIZATION]
                                  <<<    SIMD
                                  <<<    SOFTWARE PIPELINING
                                  <<< Loop-information  End >>>
      744     3   p   6v                     do i1=2,n1-1
      745     3   p   6v                        r(i1,i2,i3) = v(i1,i2,i3)
      746     3                        >                    - a(0) * u(i1,i2,i3)
      747     3                      c-------------------------------------------------------------
      748     3                      c  Assume a(1) = 0      (Enable 2 lines below if a(1) not= 0)
      749     3                      c-------------------------------------------------------------
      750     3                      c    >                  - a(1) * ( u(i1-1,i2,i3) + u(i1+1,i2,i3)
      751     3                      c    >                           + u1(i1) )
      752     3                      c-------------------------------------------------------------
      753     3                           >                  - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
      754     3                           >                  - a(3) * ( u2(i1-1) + u2(i1+1) )
      755     3   p   6v                  enddo
      756     2   p                    enddo
      757     1   p
```

u1,u2に対し、コンパイラは値を保証しようとする。

スレッドでプライベートにしてやれば最外で並列化可能



# 第2着眼点からの改善結果（1）

$cat mg.C.4.fipp.txt

Application - performance monitorsで MFLOPS/PEAK(%), Mem throughputを確認

```
Performance monitor event

*******************************************************************************
    Application - performance monitors
*******************************************************************************

    Elapsed(s)        MFLOPS MFLOPS/PEAK(%)          MIPS   MIPS/PEAK(%)
-------------------------------------------------------------------------------
      11.0426    15551.6663          3.0374   23731.9899          9.2703  Application
-------------------------------------------------------------------------------
      11.0426     3887.9166          3.0374    5942.1974          9.2847  Process     1
      10.9639     3915.8214          3.0592    5968.5331          9.3258  Process     0
      10.9068     3936.3272          3.0753    6005.5731          9.3837  Process     2
      10.9058     3936.6545          3.0755    6006.4411          9.3851  Process     3

    Elapsed(s)  Mem throughput _chip(GB/S)  Mem throughput /PEAK(%)          SIMD(%)
-------------------------------------------------------------------------------
      11.0426              57.7425                22.5557          57.9622  Application
-------------------------------------------------------------------------------
      11.0426              14.4472                22.5737          57.8725  Process     1
      10.9639              14.5169                22.6827          58.0307  Process     0
      10.9068              14.6198                22.84            57.97    Process     2
      10.9058              14.6229                22.8482          57.97    Process     3
```

Elapsed
16.65 → 11.04

MFLOPS/PEAK
2.02 → 3.03

Mem Throughput
8.08 → 14.44

簡単な分析と対処で性能向上！

# 結果の確認

$cat run_MGc4.sh.o*

Before

```
Benchmark completed
VERIFICATION SUCCESSFUL
L2 Norm is  0.5706732285740E-06
Error is    0.7124494269654E-13
```

After

```
Benchmark completed
VERIFICATION SUCCESSFUL
L2 Norm is  0.5706732285740E-06
Error is    0.7124494269654E-13
```

ひとつの修正、実行毎の結果確認は
かならず実施！



# 第2着眼点でMGの結果を見る（4）

$cat mg.C.4.fipp.txt

Procedures profileで高コスト部を確認

他の３つの主要ルーチンの
最適化状況の確認へ

Procedures profile

Application - procedures

| Cost | % | Barrier | % | MPI | % | Start | End | |
|---|---|---|---|---|---|---|---|---|
| 2581 | 100.0000 | 273 | 10.5773 | 9 | 0.3487 | -- | -- | Application |
| 861 | 33.3592 | 121 | 14.0534 | 0 | 0.0000 | 666 | 683 | psinv._PRL_1_ |
| 680 | 26.3464 | 0 | 0.0000 | 0 | 0.0000 | 735 | 757 | resid._OMP_1_ |
| 357 | 13.8318 | 42 | 11.7647 | 0 | 0.0000 | 914 | 939 | interp._PRL_1_ |
| 316 | 12.2433 | 26 | 8.2278 | 0 | 0.0000 | 832 | 853 | rprj3._PRL_1_ |
| 162 | 6.2766 | 0 | 0.0000 | 0 | 0.0000 | 1237 | 1239 | ready._PRL_1_ |
| 44 | 1.7048 | 43 | 97.7273 | 0 | 0.0000 | 637 | 702 | psinv_ |
| 23 | 0.8911 | 23 | 100.0000 | 0 | 0.0000 | 877 | 1041 | interp_ |
| 20 | 0.7749 | 0 | 0.0000 | 4 | 20.0000 | 1259 | 1389 | give3_ |
| 16 | 0.6199 | 15 | 93.7500 | 0 | 0.0000 | 780 | 872 | rprj3_ |
| 16 | 0.6199 | 0 | 0.0000 | 0 | 0.0000 | -- | -- | __jwe_etbf |

結局他の3つの主要ルーチンも同じ状況

# 第2着眼点からの改善結果（3）

$cat mg.C.4.fipp.txt

Application - performance monitorsで
MFLOPS/PEAK(%), Mem throughputを確認

Elapsed
16.65 →6.21

MFLOPS/PEAK
2.02 →5.35

Mem Throughput
8.08 →28.25

```
Performance monitor event

*******************************************************************************
    Application - performance monitors
*******************************************************************************

    Elapsed(s)        MFLOPS MFLOPS/PEAK(%)          MIPS    MIPS/PEAK(%)
-------------------------------------------------------------------------------
        6.2157   27426.4817          5.3567   35479.9589         13.8594   Application
-------------------------------------------------------------------------------
        6.2157    6856.6205          5.3567    8865.0637         13.8517   Process      0
        6.1850    6890.6751          5.3833    8919.7295         13.9371   Process      2
        6.1657    6912.2510          5.4002    8942.9662         13.9734   Process      3
        6.1485    6931.5752          5.4153    8965.2016         14.0081   Process      1

    Elapsed(s)  Mem throughput _chip(GB/S)  Mem throughput /PEAK(%)      SIMD(%)
-------------------------------------------------------------------------------
        6.2157              113.2019                44.2195      69.0129   Application
-------------------------------------------------------------------------------
        6.2157               28.2514                44.1427      69.0513   Process      0
        6.1850               28.4542                44.4597      68.9689   Process      2
        6.1657               28.5359                44.5[illegible]      69.00[illegible]   Process
        6.1485               28.6405                44.7[illegible]      69.0[illegible]   Process
```



# 第2着眼点でMGの結果を見る（7）

$cat mg.C.4.fipp.txt

Procedures profileで高コスト部を確認

最外スレッド化により
スレッドバリアのコストも
激減！

```
Procedures profile

*******************************************************************************
    Application - procedures
*******************************************************************************

      Cost        %    Barrier        %        MPI        %    Start      End
-------------------------------------------------------------------------------
      1461  100.0000          2   0.1369          8   0.5476       --       --   Application
-------------------------------------------------------------------------------
       594   40.6571          0   0.0000          0   0.0000      736      758   resid._OMP_1_
       282   19.3018          0   0.0000          0   0.0000      664      686   psinv._OMP_1_
       182   12.4572          0   0.0000          0   0.0000     1240     1242   ready._PRL_1_
       143    9.7878          0   0.0000          0   0.0000      913      948   interp._OMP_1_
       121    8.2820          0   0.0000          0   0.0000      826      858   rprj3._OMP_1_
        41    2.8063          0   0.0000          0   0.0000     2562     2568   zero3._PRL_1_
        24    1.6427          0   0.0000          6  25.0000     1262     1392   give3_
        15    1.0267          0   0.0000          0   0.0000     1397     1492   take3_
        14    0.9582          0   0.0000          0   0.0000       --       --   __jwe_etbf
        14    0.9582          0   0.0000          0   0.0000       42       78   vranlc_
```

最適化、並列化を確認、実施した4ルーチンが全体の77.8%を占めるため、
基本プロファイラによるボトルネック解析と改善はひとまず終了

# 第2着眼点による判定

アプリケーション全体の
✓高コスト部の把握
- 演算ルーチンか、通信ルーチンか　　演算ルーチン
- スケールさせたときにどうか？　　同一傾向

✓高コスト部の状態の把握
- 演算ルーチン
  最適化(スレッド並列化、simd化、ソフトウェアパイプライニング)はどうか？ 最外のスレッド並列化が必要
- 通信ルーチン
  隣接か集団か？スケールさせたときにどうか？
  （通信の解析には詳細プロファイラが必要）MGでは不要



# まとめ/ポイント

- 基本プロファイラでは以下の確認/修正を中心に実施すべし。

  ✓計算手法の目標値に比べてどうか？

  ✓性能ボトルネック/スケーラビリティの把握
  - 演算ルーチン？　通信ルーチン？
  - スケーラビリティは？

  ✓高コスト演算ルーチンの並列化、最適化
  - 最外でのスレッド並列化
  - ソフトウェアパイプライニング
  - Simd演算
    - .lstファイルを確認しながら。

# 詳細プロファイラを用いた演算性能ボトルネック解析と改善

## (要求B/F値が高いMGを例として)




## 目的

- ボトルネックとなった演算ループのより詳細な情報を取得し、その情報から更にチューニングが必要か否かを判断する。

精密PA

- 詳細プロファイラの機能の一部。Excelシートによって（現状判る限りの）ハードウェアリソースの活用度合いを把握する。
- 詳細プロファイラのCUI/GUIは精密PAの一部の情報のみ採取可能（今回は割愛）

# 精密PAの分析対象

Procedures profile

```
*******************************************************************************
    Application - procedures
*******************************************************************************

           Cost        %        Barrier        %        MPI        %    Start      End
-------------------------------------------------------------------------------------------
           1367   100.0000              1   0.0732          8   0.5852       --       --   Application
-------------------------------------------------------------------------------------------
            620    45.3548              0   0.0000          0   0.0000      736      758   resid._OMP_1_
            277    20.2633              0   0.0000          0   0.0000      664      686   psinv._OMP_1_
            140    10.2414              0   0.0000          0   0.0000      826      858   rprj3._OMP_1_
            131     9.5830              0   0.0000          0   0.0000      913      948   interp._OMP_1_
             80     5.8522              0   0.0000          0   0.0000     1241     1243   ready._PRL_1_
             20     1.4631              0   0.0000          2  10.0000     1398     1493   take3_
             19     1.3899              0   0.0000          6  31.5789     1263     1393   give3_
             17     1.2436              0   0.0000          0   0.0000     2290     2296   zran3._PRL_1_
             15     1.0973              0   0.0000          0   0.0000       --       --   __jwe_etbf
             14     1.0241              0   0.0000          0   0.0000     2115     2310   zran3_
```

基本プロファイラ編でチューニングした
rprj3(10.2%)を分析/高速化する。



# Compile/Execution/Analyze to MG (rprj3)

- Source Edit(mg-omp-p5-pa.f)
  call start_collection( “XXX” )
  call stop_collection( “XXX” )
- Compile (Class = C, 4process)
  $cd $Hands/FJTools
  $Make suite
  (Option : -Kfast,parallel,ocl,openmp,optmsg=2 -Qt)
- Execution
  $cd bin
  $pjsub --interact ./run_MGc4_fappPA.sh
- Analyze
  $./fapppx.sh ( *.csv file is generated)

# ループのB/Fから推定性能を算出する

| | | 回転数 | Mst 配列 | Mld 配列 | L2st 配列 | L2ld 配列 | L1st 配列 | L1ld 配列 | 加減算 | 乗算 | 要求B/F | 推定性能 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | rprj3 | (最大値) | 1 | 7 | 0 | 12 | 2 | 7 | 20 | 4 | 2.666667 | 0.135 |
| 1 | do j3=2,m3j-1 | 256 | | | | | | | | | | |
| | i3 = 2*j3-d3 | | | | | | | | | | | |
| 2 | do j2=2,m2j-1 | 256 | | | | | | | | | | |
| 3 | i2 = 2*j2-d2 | | | | | | | | | | | |
| 4 | do j1=2,m1j | 514 | | | | | | | | | | |
| 5 | i1 = 2*j1-d1 | | | | | | | | | | | |
| 6 | x1(i1-1) = r(i1-1,i2-1,i3 ) + r(i1-1,i2+1,i3 ) | | | | | | | | | | | |
| | > + r(i1-1,i2, i3-1) + r(i1-1,i2, i3+1) | | | 3 | | 1 | 1 | 1 | 3 | | | |
| 7 | y1(i1-1) = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1) | | | | | | | | | | | |
| | > + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1) | | | | | 4 | 1 | 1 | 3 | | | |
| 8 | enddo | | | | | | | | | | | |
| 9 | do j1=2,m1j-1 | 512 | | | | | | | | | | |
| 10 | i1 = 2*j1-d1 | | | | | | | | | | | |
| 11 | y2 = r(i1, i2-1,i3-1) + r(i1, i2-1,i3+1) | | | | | | | | | | | |
| | > + r(i1, i2+1,i3-1) + r(i1, i2+1,i3+1) | | | 2 | | 2 | | | 3 | | | |
| 12 | x2 = r(i1, i2-1,i3 ) + r(i1, i2+1,i3 ) | | | | | | | | | | | |
| | > + r(i1, i2, i3-1) + r(i1, i2, i3+1) | | | 1 | | 3 | | | 3 | | | |
| 13 | s(j1,j2,j3) = | | | | | | | | | | | |
| | > 0.5D0 * r(i1,i2,i3) | | | | | | | | | | | |
| | > + 0.25D0 * ( r(i1-1,i2,i3) + r(i1+1,i2,i3) + x2) | | | | | | | | | | | |
| | > + 0.125D0 * ( x1(i1-1) + x1(i1+1) + y2) | | | | | | | | | | | |
| | > + 0.0625D0 * ( y1(i1-1) + y1(i1+1) ) | | 1 | 1 | | 2 | | 5 | 8 | 4 | | |
| 14 | enddo | | | | | | | | | | | |
| 15 | enddo | | | | | | | | | | | |
| 16 | enddo | | | | | | | | | | | |

**性能の目安　B/F = 64/24= 2.67**
メモリ律速と仮定すると、期待される性能値は　0.36/2.57 = 13.5%

# ループrprj3の分析（1）

- ✓ ピーク性能比　4.94%
- ✓ メモリスループット

最大値: 46G/s

メモリビジー率：
メモリスループットの最大値(46G/s)に対する割合

Memory・Cache

| メモリスループット (GB/sec) | L2エンジンスループット (GB/sec) | L2 スループット (GB/sec) | メモリビジー率 | L2エンジンビジー率 | L1エンジンビジー率 |
|---|---|---|---|---|---|
| 3.37 | 15.76 | 11.17 | | | 44% |
| 3.34 | 15.71 | 11.18 | | | 44% |
| 3.34 | 15.72 | 11.16 | | | 44% |
| 3.35 | 15.69 | 11.14 | | | 44% |
| 3.35 | 15.66 | 11.09 | 58% | 70% | 44% |
| 3.33 | 15.72 | 11.18 | | | 44% |
| 3.37 | 15.80 | 11.22 | | | 44% |
| 3.35 | 15.73 | 11.17 | | | 44% |
| 26.78 | 125.70 | 89.23 | | | 44% |

B/Fが高いループであるが、メモリスループットが低い。。

# ループrprj3の分析（2）

✓ キャッシュ状況

L1D、L2ミス率
倍精度であれば6.25%が基準値
それを上回り、L1Dミスdm率が20%を超えている場合は
L1キャッシュスラッシングが発生していると考える。

Cache

| | L1I ミス率（/有効総命令数） | L1D ミス率（/ロード・ストア数） | ロード・ストア数 | L1D ミス数 | L1D ミス dm 率（/L1D ミス数） | L1D ミス hwpf 率（/L1D ミス数） | L1D ミス swpf 率（/L1D ミス数） | L2 ミス率（/ロード・ストア数） | L2 ミス数 | L2 ミス dm 率（/L2 ミス数） | L2 ミス pf 率（/L2 ミス数） | μDTLB ミス率（/ロード・ストア数） | mDTLB ミス率（/ロード・ストア数） |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Thread 0 | 0.03% | 12.52% | 2.99E+08 | 3.75E+07 | 41.47% | 58.53% | 0.00% | 3.43% | 1.03E+07 | 9.62% | 90.38% | 0.00599% | 0.00017% |
| Thread 1 | 0.03% | 12.58% | 2.99E+08 | 3.76E+07 | 41.52% | 58.48% | 0.00% | 3.40% | 1.02E+07 | 8.57% | 91.43% | 0.00449% | 0.00010% |
| Thread 2 | 0.03% | 12.57% | 2.99E+08 | 3.76E+07 | 41.81% | 58.19% | 0.00% | 3.42% | 1.02E+07 | 8.04% | 91.96% | 0.00431% | 0.00005% |
| Thread 3 | 0.03% | 12.53% | 2.99E+08 | 3.75E+07 | 41.36% | 58.64% | 0.00% | 3.41% | 1.02E+07 | 9.05% | 90.95% | 0.00417% | 0.00004% |
| Thread 4 | 0.03% | 12.48% | 2.99E+08 | 3.73E+07 | 41.78% | 58.22% | 0.00% | 3.43% | 1.03E+07 | 9.20% | 90.80% | 0.00414% | 0.00003% |
| Thread 5 | 0.03% | 12.59% | 2.99E+08 | 3.77E+07 | 41.66% | 58.33% | 0.00% | 3.41% | 1.02E+07 | 8.31% | 91.69% | 0.00430% | 0.00006% |
| Thread 6 | 0.03% | 12.63% | 2.99E+08 | 3.78E+07 | 41.84% | 58.16% | 0.00% | 3.44% | 1.03E+07 | 7.97% | 92.03% | 0.00415% | 0.00004% |
| Thread 7 | 0.03% | 12.57% | 2.99E+08 | 3.76E+07 | 41.56% | 58.44% | 0.00% | 3.43% | 1.03E+07 | 8.96% | 91.04% | 0.00494% | 0.00004% |
| Process | 0.03% | 12.56% | 2.39E+09 | 3.01E+08 | 41.62% | 58.37% | 0.00% | 3.42% | 8.19E+07 | 8.71% | 91.29% | 0.00456% | 0.00007% |

L1キャッシュスラッシングが発生していると判定する。



# ループrprj3の分析結果

- L1キャッシュスラッシングが発生している。
- 演算系はSIMD化率が低い

⇒　本ループは、演算の比率が低い（B/Fが高い）ため、
まずはスラッシングを回避する処置が必要と判断する。

## スラッシングの回避策:

- 使用する配列数を減らす
  - ✓ ループ分割、配列融合、スカラ化
- 使用する配列のアドレスをずらす
  - ✓ パディング
  - ✓ Common化

# ループrprj3のチューニング（1）

テンポラリ配列x,yの利用をやめることにより、配列数を少なくするのと同時に
ループ融合を実施し、無駄なロードを発生させないチューニングを採用する。

```
 do  j3=2,m3j-1
       i3 = 2*j3-d3
C       i3 = 2*j3-1
       do  j2=2,m2j-1
         i2 = 2*j2-d2
C          i2 = 2*j2-1

         do j1=2,m1j
           i1 = 2*j1-d1
C           i1 = 2*j1-1
           x1(i1-1) = r(i1-1,i2-1,i3  ) + r(i1-1,i2+1,i3  )
     >              + r(i1-1,i2,  i3-1) + r(i1-1,i2,  i3+1)
           y1(i1-1) = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1)
     >              + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1)
         enddo

         do  j1=2,m1j-1
           i1 = 2*j1-d1
C           i1 = 2*j1-1
           y2 = r(i1,  i2-1,i3-1) + r(i1,  i2-1,i3+1)
     >        + r(i1,  i2+1,i3-1) + r(i1,  i2+1,i3+1)
           x2 = r(i1,  i2-1,i3  ) + r(i1,  i2+1,i3  )
     >        + r(i1,  i2,  i3-1) + r(i1,  i2,  i3+1)
           s(j1,j2,j3) =
     >           0.5D0 * r(i1,i2,i3)
     >         + 0.25D0 * ( r(i1-1,i2,i3) + r(i1+1,i2,i3) + x2)
     >         + 0.125D0 * ( x1(i1-1) + x1(i1+1) + y2)
     >         + 0.0625D0 * ( y1(i1-1) + y1(i1+1) )
         enddo

       enddo
     enddo
```

```
!$OMP PARALLEL DO PRIVATE(xy1,y2,x2,i3,i2,i1)
       do  j3=2,m3j-1
         i3 = 2*j3-d3
         do  j2=2,m2j-1
           i2 = 2*j2-d2
           do  j1=2,m1j-1
             i1 = 2*j1-d1
C             i1 = 2*j1-1
             xy11m = r(i1-1,i2-1,i3  ) + r(i1-1,i2+1,i3  )
     >               + r(i1-1,i2,  i3-1) + r(i1-1,i2,  i3+1)
             xy11p = r(i1+1,i2-1,i3  ) + r(i1+1,i2+1,i3  )
     >               + r(i1+1,i2,  i3-1) + r(i1+1,i2,  i3+1)
             xy12m = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1)
     >               + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1)
             xy12p = r(i1+1,i2-1,i3-1) + r(i1+1,i2-1,i3+1)
     >               + r(i1+1,i2+1,i3-1) + r(i1+1,i2+1,i3+1)
             y2 = r(i1,  i2-1,i3-1) + r(i1,  i2-1,i3+1)
     >          + r(i1,  i2+1,i3-1) + r(i1,  i2+1,i3+1)
             x2 = r(i1,  i2-1,i3  ) + r(i1,  i2+1,i3  )
     >          + r(i1,  i2,  i3-1) + r(i1,  i2,  i3+1)
             s(j1,j2,j3) =
     >             0.5D0 * r(i1,i2,i3)
     >           + 0.25D0 * ( r(i1-1,i2,i3) + r(i1+1,i2,i3) + x2)
     >           + 0.125D0 * ( xy11m + xy11p + y2)
     >           + 0.0625D0 * ( xy12m + xy12p )
           enddo

         enddo
       enddo
```



# ループのB/Fから推定性能を算出する

| | | 回転数 | Mst 配列 | Mld 配列 | L2st 配列 | L2ld 配列 | L1st 配列 | L1ld 配列 | 加減算 | 乗算 | 要求B/F | 推定性能 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | rprj3 | (最大値) | 1 | 4 | 0 | 10 | 0 | 14 | 26 | 4 | 1.333333 | 0.27 |
| 1 | do j3=2,m3j-1 | 256 | | | | | | | | | | |
| | i3 = 2*j3-d3 | | | | | | | | | | | |
| 2 | do j2=2,m2j-1 | 256 | | | | | | | | | | |
| 3 | i2 = 2*j2-d2 | | | | | | | | | | | |
| 4 | do j1=2,m1j | 514 | | | | | | | | | | |
| 5 | i1 = 2*j1-d1 | | | | | | | | | | | |
| 6 | xy11m = r(i1-1,i2-1,i3 ) + r(i1-1,i2+1,i3 ) | | | | | | | | | | | |
| | > + r(i1-1,i2, i3-1) + r(i1-1,i2, i3+1) | | | 3 | | 1 | | | 3 | | | |
| 7 | xy11p = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1) | | | | | | | | | | | |
| | > + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1) | | | | | 4 | | | 3 | | | |
| 8 | | | | | | | | | | | | |
| 9 | xy12m = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1) | | | | | | | | | | | |
| 10 | > + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1) | | | | | 4 | | | 3 | | | |
| 11 | xy12p = r(i1+1,i2-1,i3-1) + r(i1+1,i2-1,i3+1) | | | | | | | | | | | |
| | > + r(i1+1,i2+1,i3-1) + r(i1+1,i2+1,i3+1) | | | | | | | 4 | 3 | | | |
| | y2 = r(i1, i2-1,i3-1) + r(i1, i2-1,i3+1) | | | | | | | | | | | |
| 12 | > + r(i1, i2+1,i3-1) + r(i1, i2+1,i3+1) | | | | | | | 4 | 3 | | | |
| | x2 = r(i1, i2-1,i3 ) + r(i1, i2+1,i3 ) | | | | | | | | | | | |
| | > + r(i1, i2, i3-1) + r(i1, i2, i3+1) | | | | | | | 4 | 3 | | | |
| 13 | s(j1,j2,j3) = | | | | | | | | | | | |
| | > 0.5D0 * r(i1,i2,i3) | | | | | | | | | | | |
| | > + 0.25D0 * ( r(i1-1,i2,i3) + r(i1+1,i2,i3) + x2) | | | | | | | | | | | |
| | > + 0.125D0 * ( xy11m + xy11p + y2) | | | | | | | | | | | |
| | > + 0.0625D0 * ( xy12m + xy12p ) | | 1 | 1 | | 1 | | 2 | 8 | 4 | | |
| 14 | enddo | | | | | | | | | | | |
| 15 | enddo | | | | | | | | | | | |
| 16 | enddo | | | | | | | | | | | |

**性能の目安**　B/F =40/30= 1.33
メモリ律速と仮定すると、期待される性能値は　0.36/1.33=27.0%
ここまでくるとメモリ律速でなくキャッシュ律速になると予測される

# ループrprj3のチューニング（2）

| | Before | After |
|---|---|---|
| L1Dミス率 | 12.56% | 8.46% |
| L1Dミスdm率 | 41.62% | 44.77% |
| メモリスループット | 26.78G/s | 32.75G/s |
| Peak性能 | 4.94% | 8.72% |

改善はしたが、まだスラッシングが
発生している状態である。



# ループrprj3のチューニング（3）

配列rをpaddingすることにより、アドレスを明にずらす。
（rは1次元配列を3次元配列の引数で受け取るため、様々なルーチンに手が入る）

```
 do  j3=2,m3j-1
       i3 = 2*j3-d3
C       i3 = 2*j3-1
       do  j2=2,m2j-1
         i2 = 2*j2-d2
C         i2 = 2*j2-1

          do j1=2,m1j
            i1 = 2*j1-d1
C           i1 = 2*j1-1
            x1(i1-1) = r(i1-1,i2-1,i3  ) + r(i1-1,i2+1,i3  )
     >               + r(i1-1,i2,  i3-1) + r(i1-1,i2,  i3+1)
            y1(i1-1) = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1)
     >               + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1)
          enddo

          do  j1=2,m1j-1
            i1 = 2*j1-d1
C           i1 = 2*j1-1
            y2 = r(i1,  i2-1,i3-1) + r(i1,  i2-1,i3+1)
     >         + r(i1,  i2+1,i3-1) + r(i1,  i2+1,i3+1)
            x2 = r(i1,  i2-1,i3  ) + r(i1,  i2+1,i3  )
     >         + r(i1,  i2,  i3-1) + r(i1,  i2,  i3+1)
            s(j1,j2,j3) =
     >           0.5D0 * r(i1,i2,i3)
     >         + 0.25D0 * ( r(i1-1,i2,i3) + r(i1+1,i2,i3) + x2)
     >         + 0.125D0 * ( x1(i1-1) + x1(i1+1) + y2)
     >         + 0.0625D0 * ( y1(i1-1) + y1(i1+1) )
          enddo

        enddo
      enddo
```

```
 double precision r(m1k+F_PAD,m2k,m3k), s(m1j+F_PAD,m2j,m3j)
                                  :
!$OMP PARALLEL DO PRIVATE(xy11m,xy11p,xy12m,xy12p,y2,x2,i3,i2,i1)
       do  j3=2,m3j-1
         i3 = 2*j3-d3
         do  j2=2,m2j-1
           i2 = 2*j2-d2
           do  j1=2,m1j-1
             i1 = 2*j1-d1
C            i1 = 2*j1-1
             xy11m = r(i1-1,i2-1,i3  ) + r(i1-1,i2+1,i3  )
     >               + r(i1-1,i2,  i3-1) + r(i1-1,i2,  i3+1)
             xy11p = r(i1+1,i2-1,i3  ) + r(i1+1,i2+1,i3  )
     >               + r(i1+1,i2,  i3-1) + r(i1+1,i2,  i3+1)
             xy12m = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1)
     >               + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1)
             xy12p = r(i1+1,i2-1,i3-1) + r(i1+1,i2-1,i3+1)
     >               + r(i1+1,i2+1,i3-1) + r(i1+1,i2+1,i3+1)
             y2 = r(i1,  i2-1,i3-1) + r(i1,  i2-1,i3+1)
     >          + r(i1,  i2+1,i3-1) + r(i1,  i2+1,i3+1)
             x2 = r(i1,  i2-1,i3  ) + r(i1,  i2+1,i3  )
     >          + r(i1,  i2,  i3-1) + r(i1,  i2,  i3+1)
             s(j1,j2,j3) =
     >             0.5D0 * r(i1,i2,i3)
     >           + 0.25D0 * ( r(i1-1,i2,i3) + r(i1+1,i2,i3) + x2)
     >           + 0.125D0 * ( xy11m + xy11p + y2)
     >           + 0.0625D0 * ( xy12m + xy12p )
           enddo

         enddo
       enddo
```

# ループrprj3のチューニング（4）

| | Before | After |
|---|---|---|
| L1Dミス率 | 8.46% | 6.45% |
| L1Dミスdm率 | 44.77% | 15.44% |
| メモリスループット | 32.75G/s | 41.33G/s |
| Peak性能 | 8.72% | 10.96% |

倍精度の連続ミス率基準値(6.25%)に近くなり、
性能、メモリスループットが改善
ピーク性能についてはまだ改善の余地はあるがキャッシュ律速の
場合の予測値の求め方は現在研究中



# 基本プロファイラによる最終確認
(その他residのチューニングも実施後)

$cat mg.C.4.fipp.txt

Application - performance monitorsで
MFLOPS/PEAK(%), Mem throughputを確認

Elapsed
5.85→5.17

MFLOPS/PEAK
5.69→6.61

Mem Throughput
27.31→27.93

```
Performance monitor event

*************************************************************************************
    Application - performance monitors
*************************************************************************************

    Elapsed(s)        MFLOPS MFLOPS/PEAK(%)          MIPS    MIPS/PEAK(%)
-------------------------------------------------------------------------------------
       5.1776      33847.9384         6.6109   40467.6649         15.8077   Application
-------------------------------------------------------------------------------------
       5.1776       8461.9844         6.6109   10108.0957         15.7939   Process     0
       5.1570       8495.7458         6.6373   10158.6487         15.8729   Process     1
       5.1564       8496.6799         6.6380   10161.4671         15.8773   Process     3
       5.1563       8496.8480         6.6382   10163.0159         15.8797   Process     2

    Elapsed(s)  Mem throughput _chip(GB/S)  Mem throughput /PEAK(%)        SIMD(%)
-------------------------------------------------------------------------------------
       5.1776              111.8551                43.6934         73.1978   Application
-------------------------------------------------------------------------------------
       5.1776               27.9324                43.6443         73.2617   Process     0
       5.1570               28.1528                43.9887         73.1880   Process     1
       5.1564               28.0569                43.8389         73.1757   Process     3
       5.1563               28.0547                43.8354         73.1660   Process     2
```

他のループも同様に実施することにより、
更なる性能向上が期待できる。

# 詳細PAの着眼点

- ✓ Performance
  - ➢ ピーク性能比は期待する性能と比べてどうか？
- ✓ Memory・Cache
  - ➢ メモリ、キャッシュスループットは十分に出ているか？
- ✓ Simd
  - ➢ Simdの命令率は高いか？
  - ➢ 但し、B/Fが高い場合は優先順位は後
- ✓ Cache
  - ➢ ミス率が妥当か？
  - ➢ L1スラッシングが発生していないか？
- ✓ 実行時間内訳：
  - ➢ 演算待ちの割合はどれくらいか？