2016年6月10日（金）
FrontISTR研究会

FrontISTRのノード内並列計算

matvec_33　：　疎行列ベクトル積

precond_33　：　マルチカラー依存性の排除

奥田洋司

okuda@k.u-tokyo.ac.jp

東京大学大学院・新領域創成科学研究科・人間環境学専攻

目次

- 導入
- なぜ並列化か？
  - 並列アーキテクチャ、並列プログラミング
- FrontISTRにおける並列計算
- 実効性能について

- ノード間並列
  - 領域分割とMPI
- ノード内並列（単体性能）
  - ループ分割とOpenMP

今回はこの話が中心です

高コストなルーチン matvec_33 と precond_33 のOpenMP化

ハイブリッド並列化のためには、高コストなルーチン matvec_33 と precond_33 に手でOpenMPの指示文を挿入して、スレッド並列化する。

matvec_33

```
      do i= 1, hecMAT%N
        X1= X(3*i-2)
        X2= X(3*i-1)
        X3= X(3*i  )
        YV1= hecMAT%D(9*i-8)*X1 + hecMAT%D(9*i-7)*X2  &
     &                          + hecMAT%D(9*i-6)*X3
   (中略)
        jS= hecMAT%indexL(i-1) + 1
        jE= hecMAT%indexL(i  )
        do j= jS, jE
          in  = hecMAT%itemL(j)
          X1= X(3*in-2)
          X2= X(3*in-1)
          X3= X(3*in  )
          YV1= YV1 + hecMAT%AL(9*j-8)*X1 + hecMAT%AL(9*j-7)*X2  &
     &                                   + hecMAT%AL(9*j-6)*X3
   (中略)
        enddo
        jS= hecMAT%indexU(i-1) + 1
        jE= hecMAT%indexU(i  )
        do j= jS, jE
   (中略)
        enddo
        Y(3*i-2)= YV1
        Y(3*i-1)= YV2
        Y(3*i  )= YV3
      enddo
```

外側のループ

内側のループ

配列Xは参照のみ。

配列Yは代入のみ。

matvec_33には参照・代入の依存関係は無いことが確認できた。

precond_33

```
!C-- FORWARD
        do i= 1, hecMAT%N
          SW1= ZP(3*i-2)
          SW2= ZP(3*i-1)
          SW3= ZP(3*i  )
          isL= hecMAT%indexL(i-1)+1
          ieL= hecMAT%indexL(i)
          do j= isL, ieL
            k= hecMAT%itemL(j)
            X1= ZP(3*k-2)
            X2= ZP(3*k-1)
            X3= ZP(3*k  )
            SW1= SW1 - hecMAT%AL(9*j-8)*X1 - hecMAT%AL(9*j-7)*X2  &
     &                                     - hecMAT%AL(9*j-6)*X3
   (中略)
          enddo
   (中略)
          X1= hecMAT%ALU(9*i-8)*( X1 - hecMAT%ALU(9*i-6)*X3  &
     &                               - hecMAT%ALU(9*i-7)*X2)
          ZP(3*i-2)= X1
          ZP(3*i-1)= X2
          ZP(3*i  )= X3
        enddo
```

外側ループに関する以前の行番号を参照

前の行のZPの値が参照されている！

precond_33には参照・代入の依存関係がある。

参照・代入に依存関係があるので、スレッド並列化できない。
OpenMP化には、節点ループに依存関係がないように、節点番号のリオーダリング処理（カラーリング）が必要。

matvec_33およびprecond_33のOpenMP化を実施

プログラムの場所　FrontISTR_V44/hecmw1/src/solver/solver_33

matvec_33 :

hecmw_solver_CG_33.f90

　サブルーチン名　　hecmw_solve_CG_33 （3x3CGソルバ）

hecmw_solver_las_33.f90

　サブルーチン名　　hecmw_matvec_33 （疎行列ベクトル積）

precond_33 :

hecmw_precond_SSOR_33.f90

　サブルーチン名　　hecmw_precond_SSOR_33_setup

　サブルーチン名　　hecmw_precond_SSOR_33_apply

hecmw_matrix_ordering_MC.f90

　サブルーチン名　　hecmw_matrix_ordering_MC

precond_33のOpenMP化

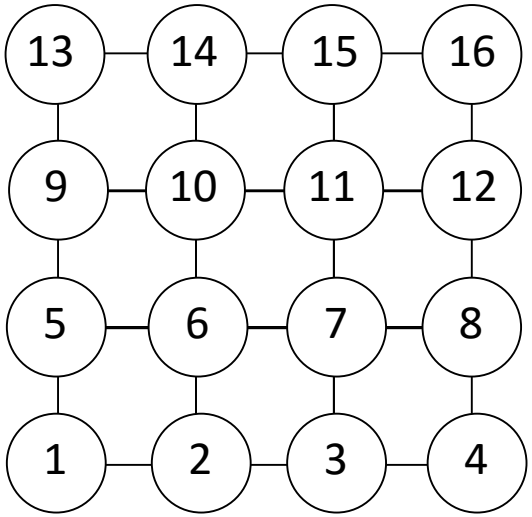
外側の変数 i のループは各 i について配列の参照・代入の依存関係がある。
節点の集合をマルチカラー法(MC法)によりカラーリングすることによって、
i のループ内で配列の参照・代入の依存関係が無いようにすることができる。

MC法の基本的なアルゴリズム

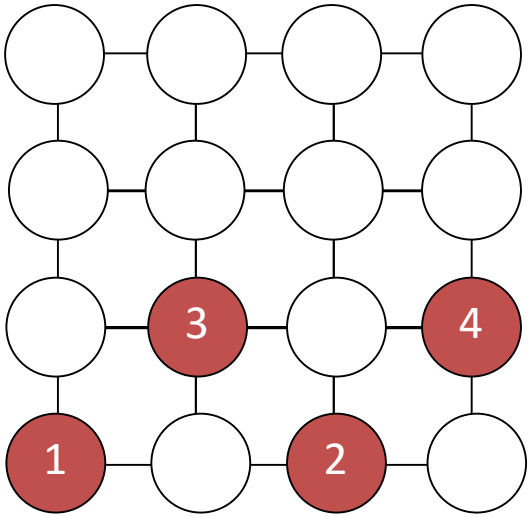
1. 「節点数÷色数」を「m」とする。
2. 初期節点番号が若い順に互いに独立な節点を「m」個選び出して彩色し、次の色へ進む。
3. 指定した色数に達して、全ての節点が彩色されるまで 2を繰り返す。
4. 色番号の若い順番に節点を再番号づけする（各色内では初期節点番号が若い順）。

MC法：16節点。4色に設定。
16/4=4なので、1色当り最大4個。
4色に彩色される。

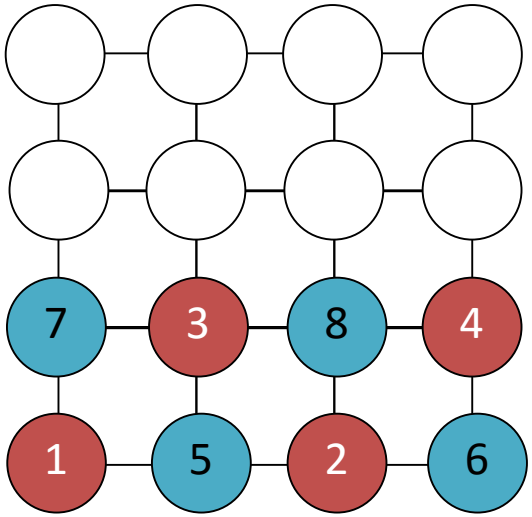
丸は節点を表わす。丸と丸を結ぶ線分は、節点と節点が隣接関係であることを表わす。

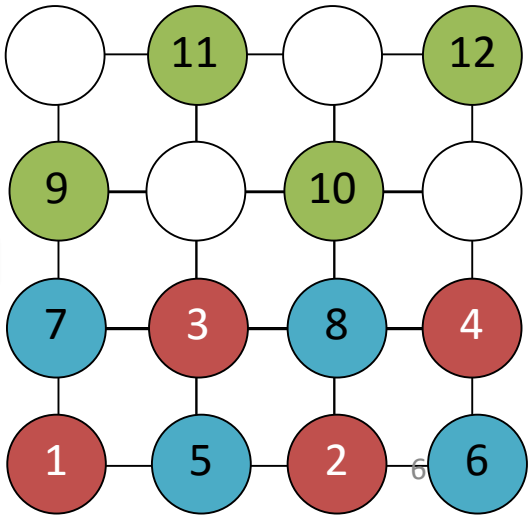


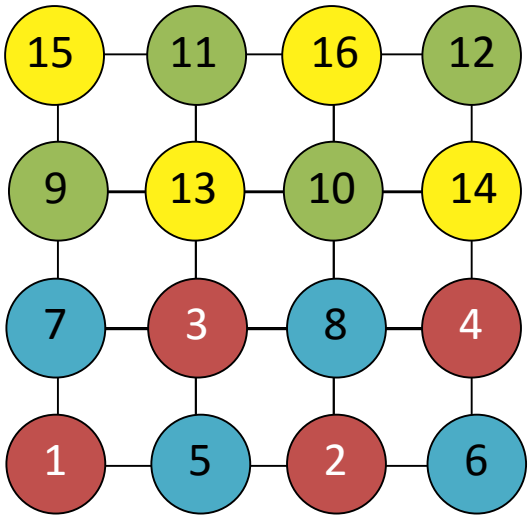
古い節点番号



新しい節点番号



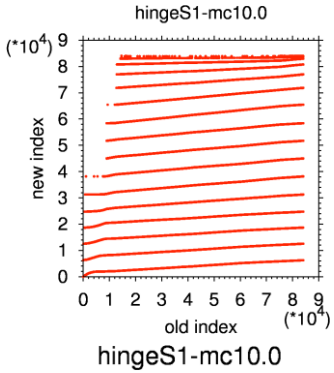




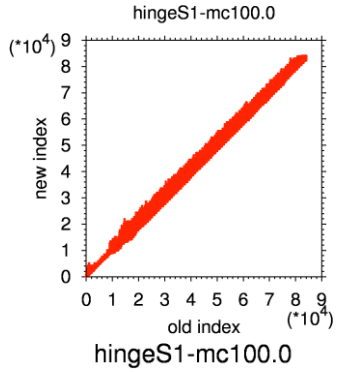
1. 「節点数÷色数」を「m」とする。
2. 初期節点番号が若い順に互いに独立な節点を「m」個選び出して彩色し、次の色へ進む。
3. 指定した色数に達して、全ての節点が彩色されるまで2を繰り返す。
4. 色番号の若い順番に節点を再番号づけする（各色内では初期節点番号が若い順）。

カラーリング処理適用後の行列の構造

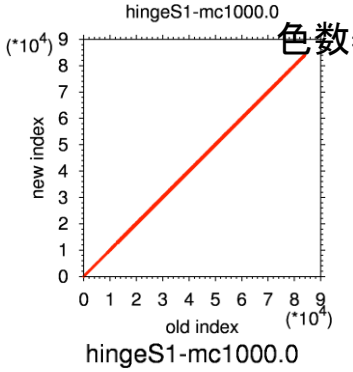
色数=10



色数=100



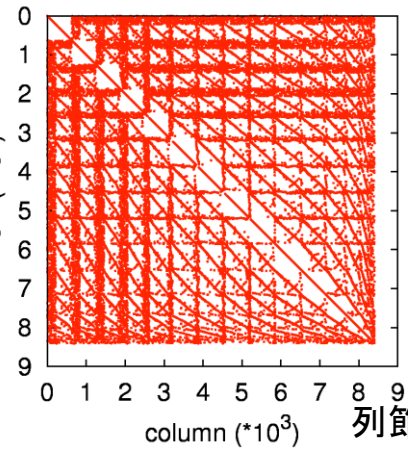
色数=1000



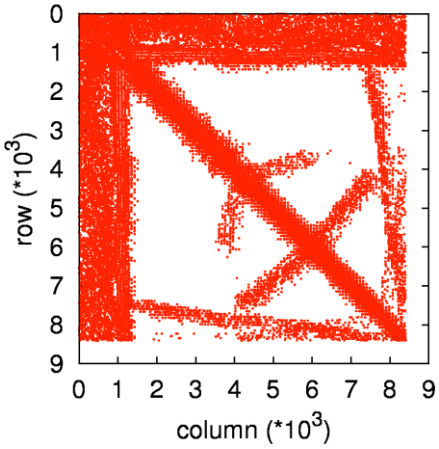
新しい節点番号

古い節点番号

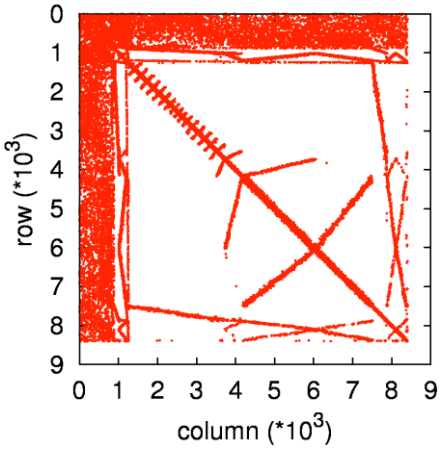
hingeS1-mc10.0



hingeS1-mc100.0



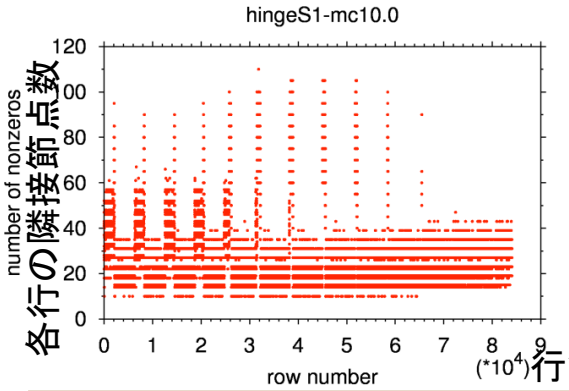
hingeS1-mc1000.0



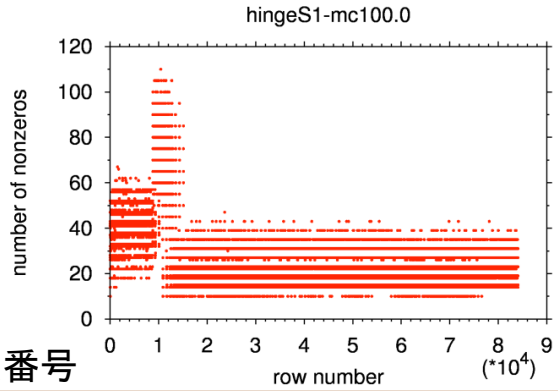
行節点番号

列節点番号

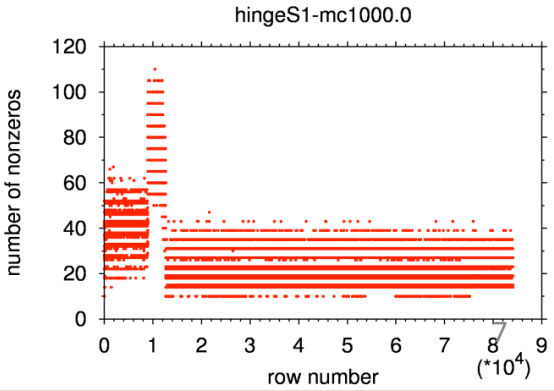
hingeS1-mc10.0



hingeS1-mc100.0



hingeS1-mc1000.0

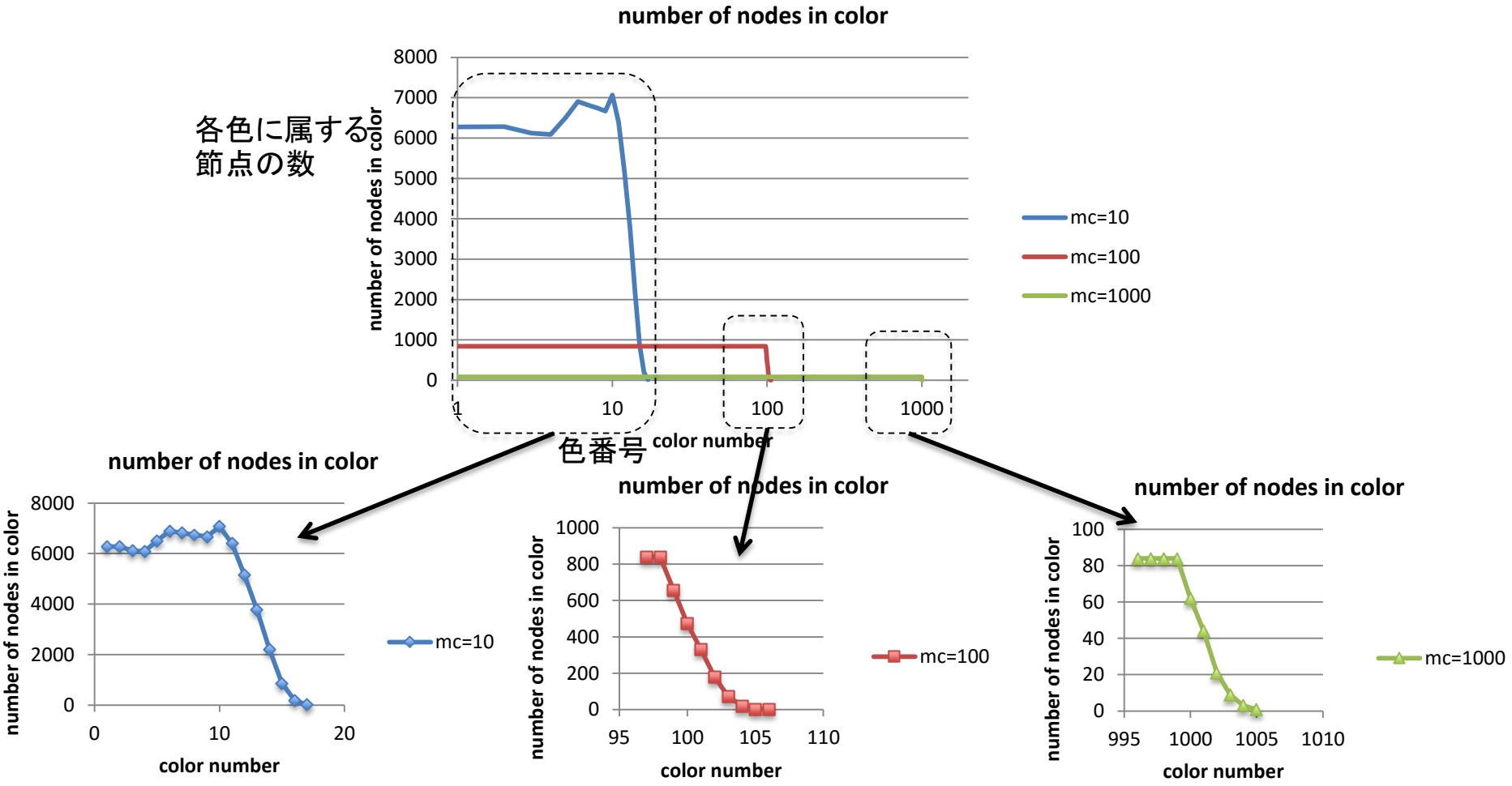


各行の隣接節点数

行節点番号

- 同じ行列でも、カラーリング後の新しい節点番号で並びかえると、色数により行列の非ゼロ要素のパターンが変わることに注意。

各色に属する節点の数の分布



- 色数が増加すると、一つの色に属する節点数は少なくなる。

precond_33とmatvec_33のOpenMP化（ブロック分割）

precond_33のOpenMP化

変数 i のループは各 i について配列の参照・代入の依存関係がある。
節点の集合を依存関係が無いようにカラーリングすることによって、
i のループをOpenMPでスレッド並列化することが可能である。
コンパイルオプション -Kfast,openmp,optmsg=2 -V -Qt でコンパイルする。

例: 前進代入（後退代入も同様）

行（節点）iのループがスレッド並列化された

一つの色内で行（節点）ループをブロック分割

色のループ

行（節点）のループ

列（隣接節点）のループ

カラーリング処理により、節点iの属する色icと、節点iの隣接節点kの属する色kcは、異なることが保証されている。

各色ごとの同期によって、色icの節点iの処理の最中に、参照しようとする節点kでの変数ZP(3*k-2)の値は変更されることがない。

一つの色の計算が終わってから次の色の計算に行く必要がある。つまり、各色ごとで同期が必要。

```
356   2          !C-- FORWARD
357   3                do ic=1,hecMESH%NCOLORTot
358   3          !$omp parallel default(none) shared(hecMESH,hecMAT,ic,ZP) private(SW1,SW2,SW3,X1,X2,X3,i,isL,ieL,j,k)
359   3          !$omp do
360   4  p             do i= hecMESH%COLORindex(ic-1)+1, hecMESH%COLORindex(ic)
361   4  p               SW1= ZP(3*i-2)
                     (中略)
                  <<< Loop-information Start >>>
                  <<<  [OPTIMIZATION]
                  <<<    SIMD
                  <<<    SOFTWARE PIPELINING
                  <<< Loop-information  End >>>
366   5  p    v          do j= isL, ieL
367   5  p    v            k= hecMAT%itemL(j)
368   5  p    v            X1= ZP(3*k-2)
                     (中略)
371   5  p    v            SW1= SW1 - hecMAT%AL(9*j-8)*X1 - hecMAT%AL(9*j-7)*X2 - hecMAT%AL(9*j-6)*X3
                     (中略)
374   5  p    v          enddo ! j
376   5  p               if (hecMAT%cmat%n_val.ne.0) then
                     (中略)
388   5  p               endif
390   4  p               X1= SW1
                     (中略)
397   4  p               X1= hecMAT%ALU(9*i-8)*( X1 - hecMAT%ALU(9*i-6)*X3 - hecMAT%ALU(9*i-7)*X2)
398   4  p               ZP(3*i-2)= X1
                     (中略)
401   4  p             enddo ! i
402   3          !$omp end do
403   3          !$omp end parallel
404   3                enddo ! ic
```

matvec_33のOpenMP化

外側の行（節点）i のループは各 i について配列の参照・代入の依存関係が無い。
i のループをOpenMPでスレッド並列化することが可能。

コンパイルオプション -Kfast,openmp,optmsg=2 -V -Qt でコンパイルする。

行（節点）iのループがスレッド並列化された

```
30                !$OMP PARALLEL DEFAULT(NONE) PRIVATE(i,X1,X2,X3,YV1,YV2,YV3,jS,jE,j,in) &
                  !$OMP    SHARED(hecMAT,X,Y)
32                !$OMP DO
33      1   p             do i= 1, hecMAT%N
34      1   p               X1= X(3*i-2)
                (中略)
37      1   p               YV1= hecMAT%D(9*i-8)*X1 + hecMAT%D(9*i-7)*X2
38      1                 &                       + hecMAT%D(9*i-6)*X3
                (中略)
46      2   p   v           do j= jS, jE
47      2   p   v             in  = hecMAT%itemL(j)
48      2   p   v             X1= X(3*in-2)
                (中略)
51      2   p   v             YV1= YV1 + hecMAT%AL(9*j-8)*X1 + hecMAT%AL(9*j-7)*X2          &
52      2                 &                                  + hecMAT%AL(9*j-6)*X3
                (中略)
57      2   p   v           enddo
                (中略)
72      1   p               Y(3*i-2)= YV1
                (中略)
75      1   p             enddo
76                !$OMP END DO
77                !$OMP END PARALLEL
```

色のループは不要

行（節点）のループ

配列Xは参照のみ。

節点番号は全てカラーリング後の番号

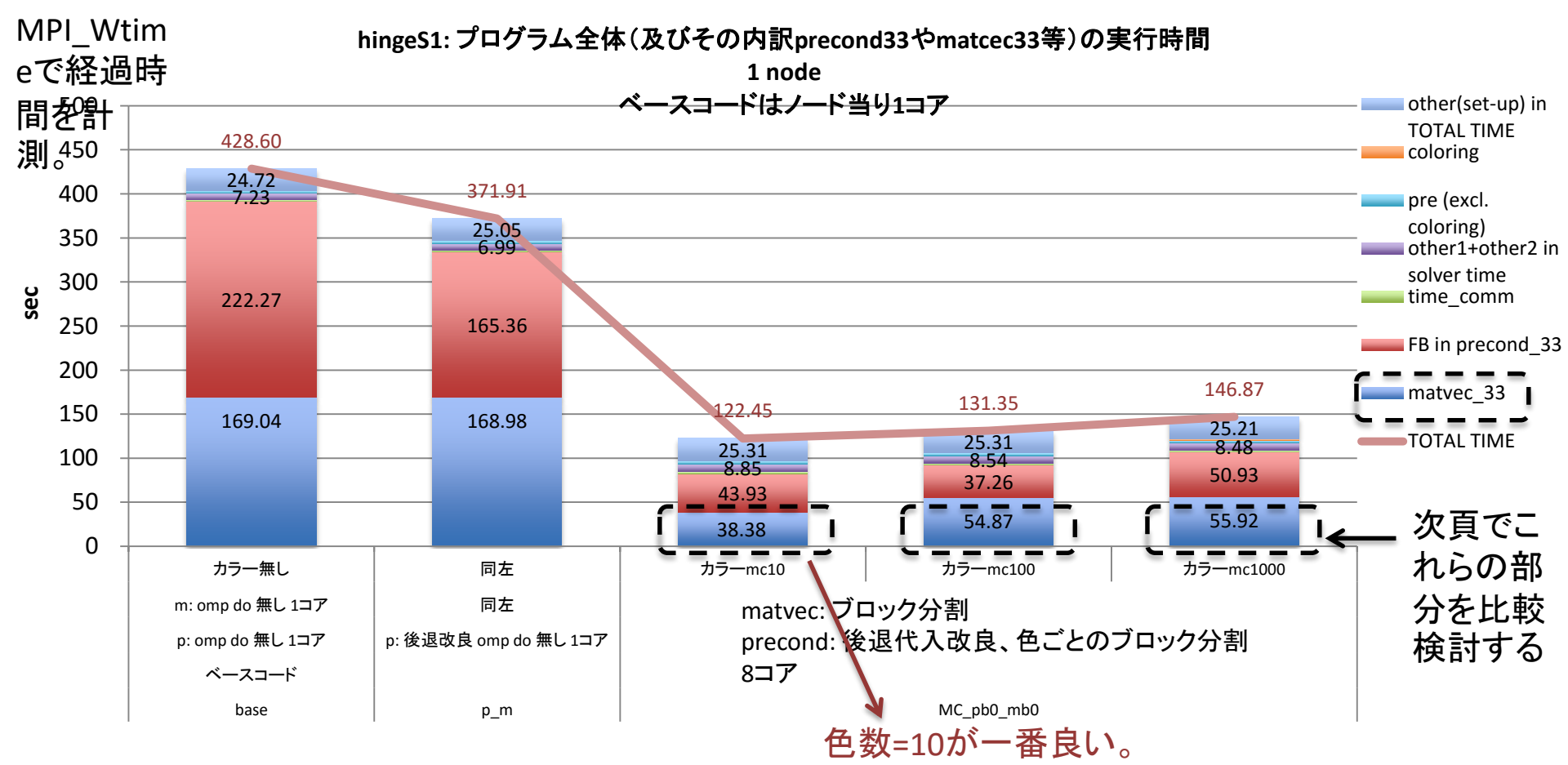
列（隣接節点）のループ

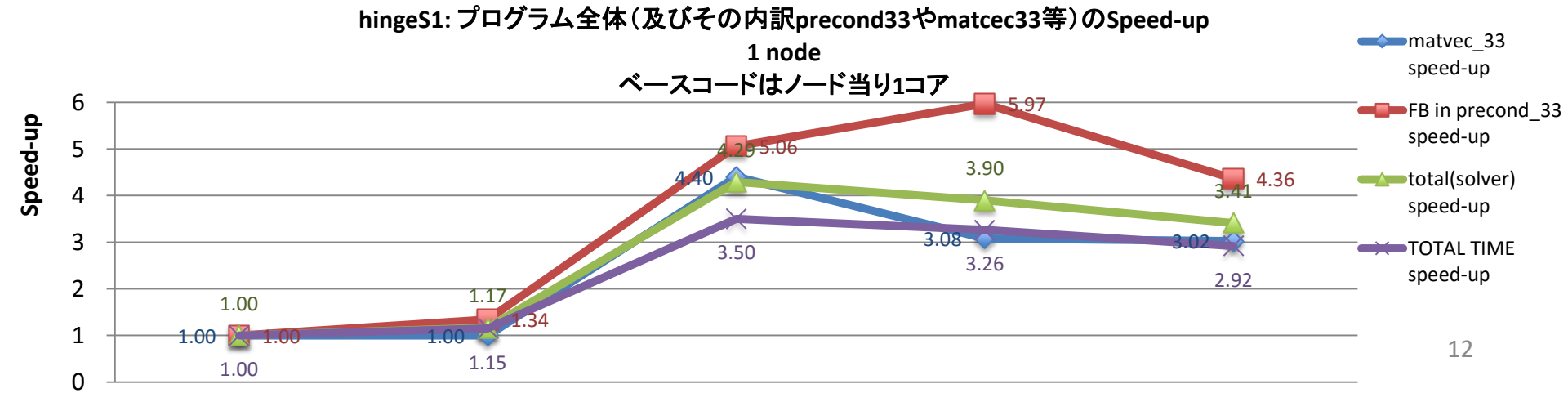
変数 in は 節点 i の隣接節点番号

配列Xは参照のみ。

配列Yは代入のみ。

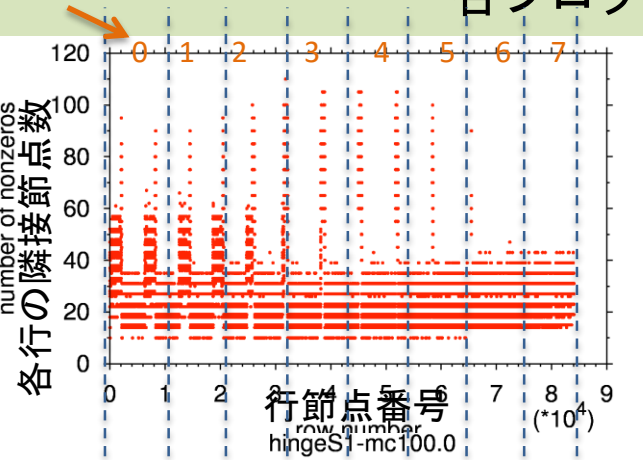
MPI_Wtimeで経過時間を計測。



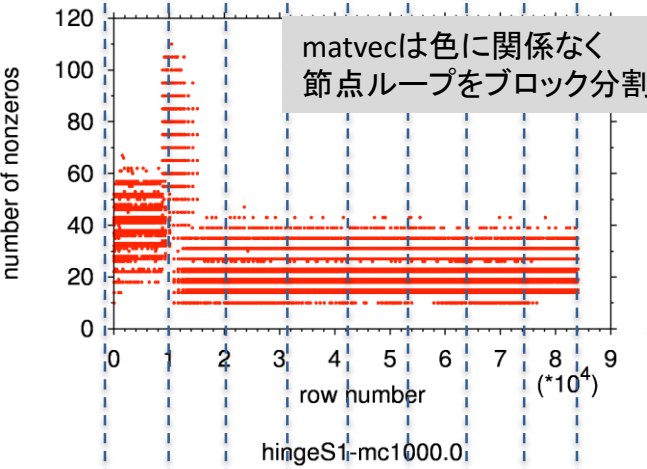


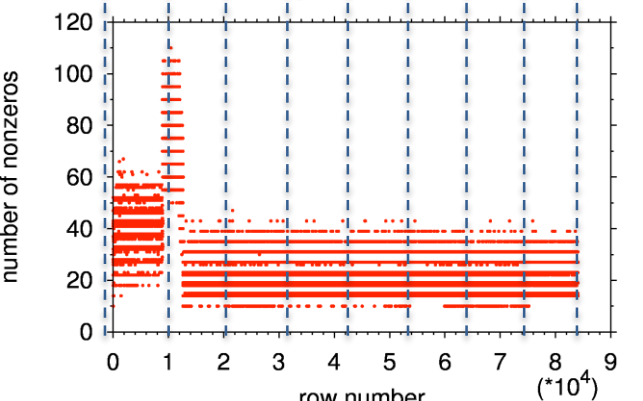
Matvec33: 行（節点）ループをブロック分割した場合の各ブロック内の非ゼロ要素の数の分布の評価

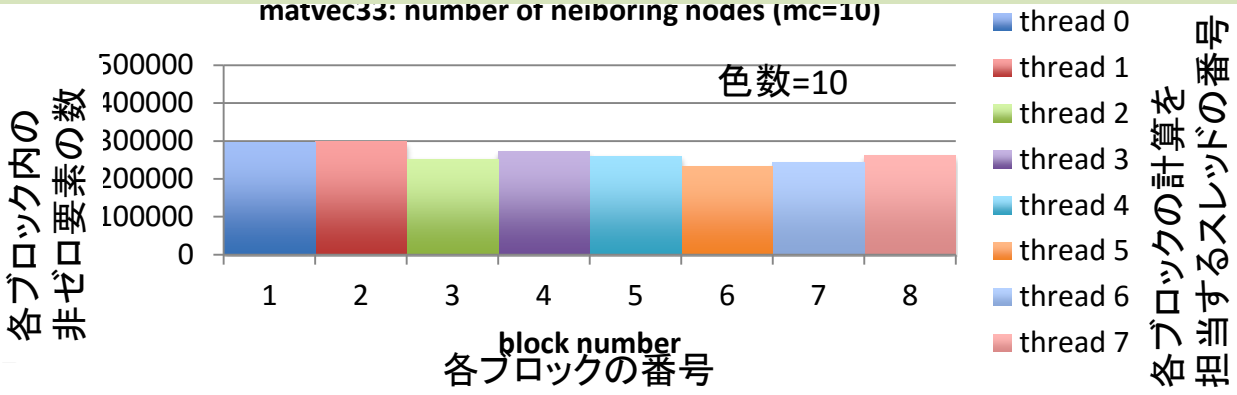
各ブロックの計算を担当するスレッドの番号



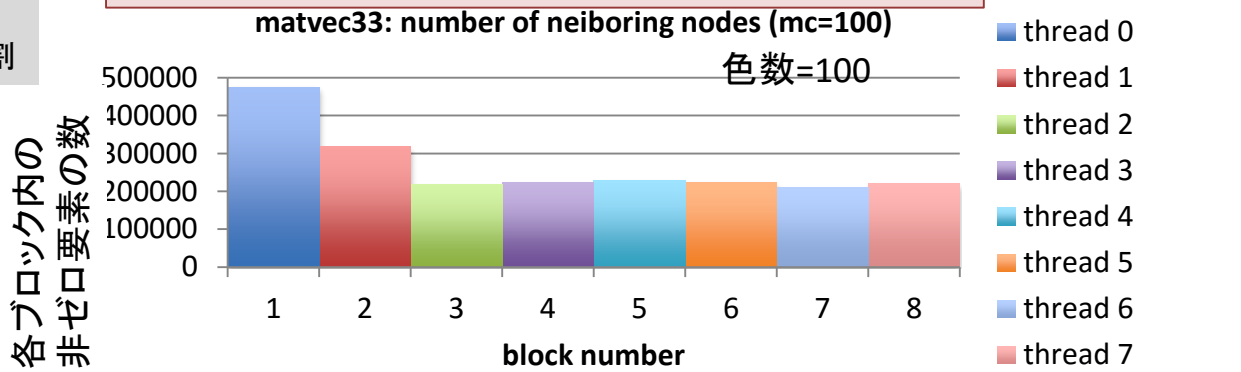
matvecは色に関係なく節点ループをブロック分割

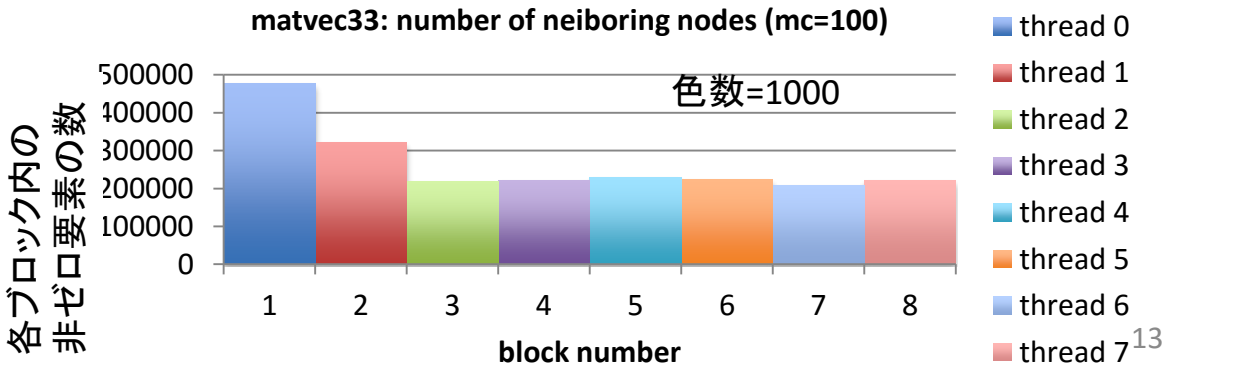




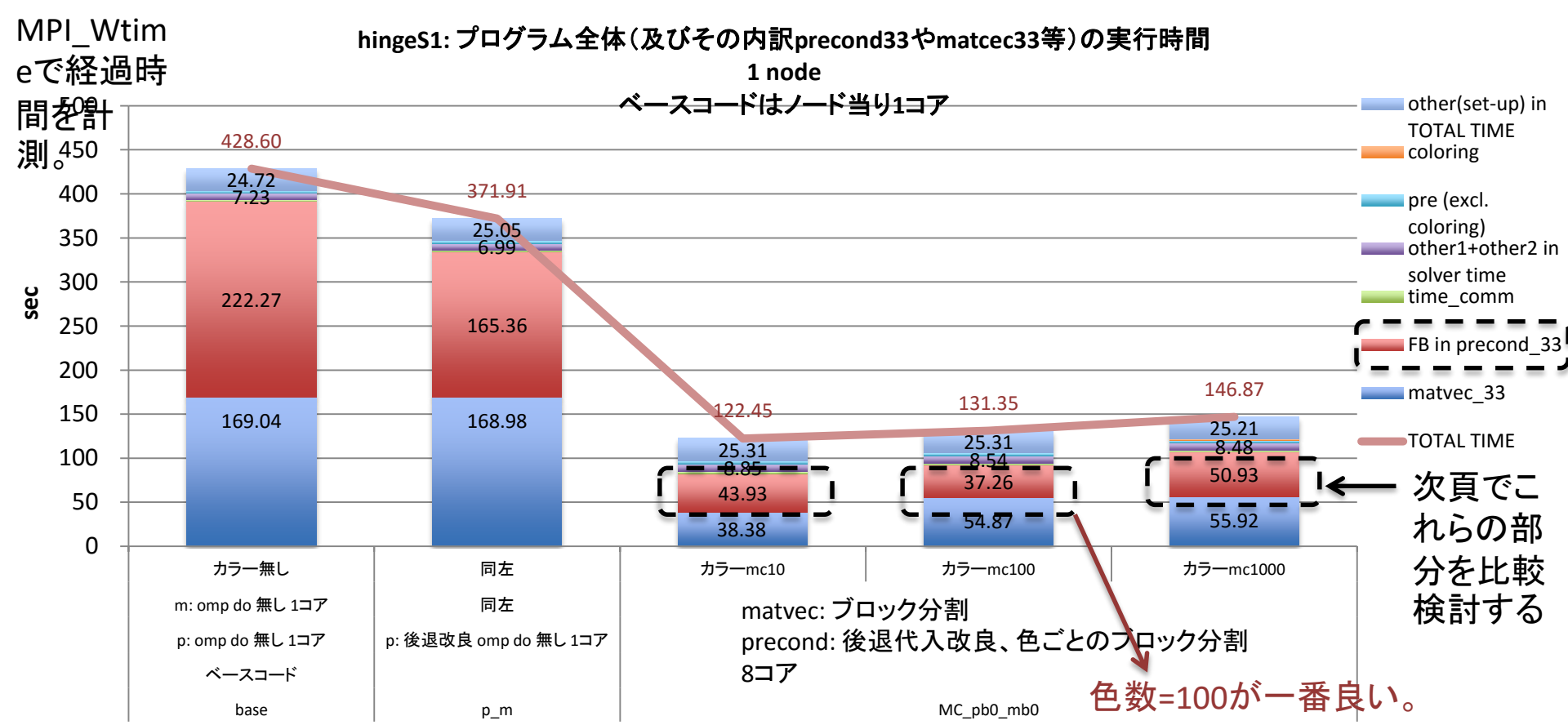


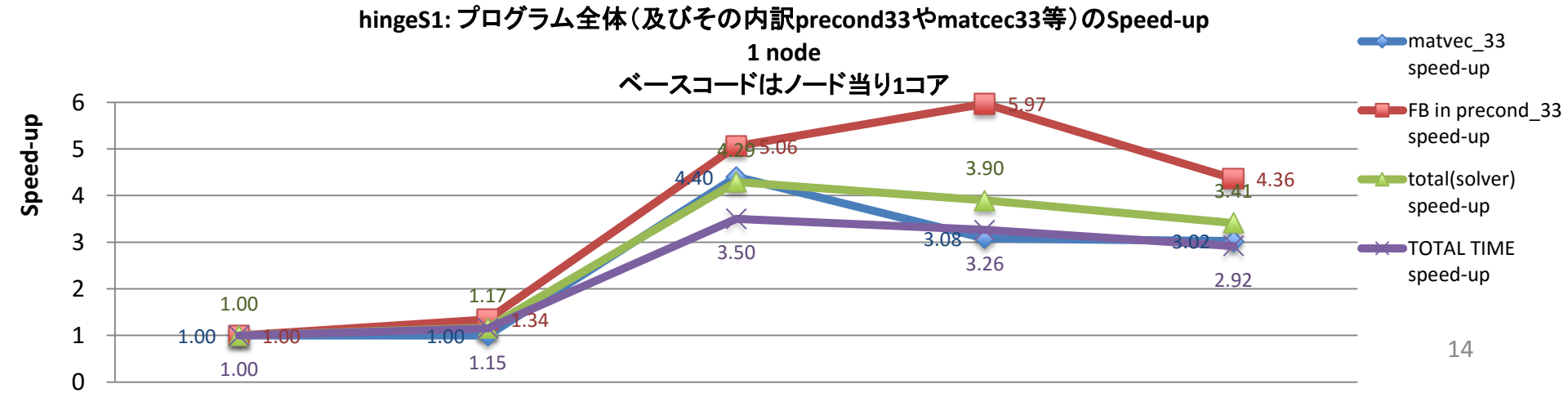
色数が10を超えるとロードインバランスが大きくなる





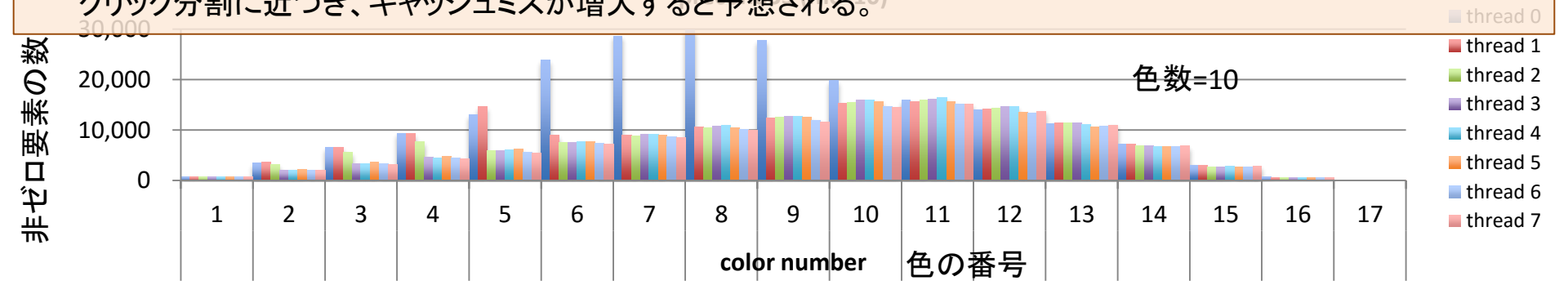
MPI_Wtimeで経過時間を計測。

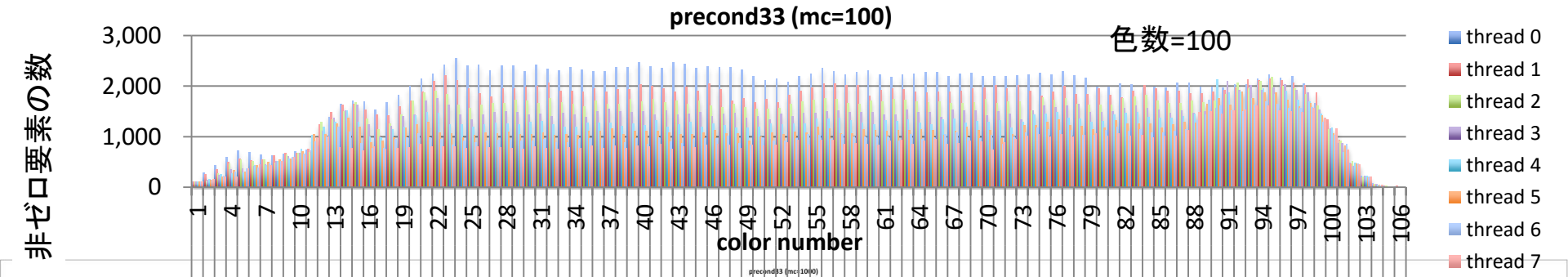


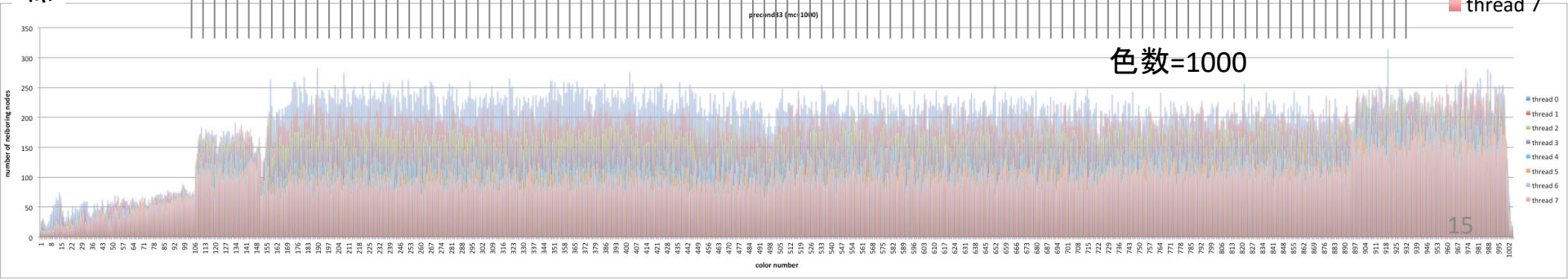


Precond33: 色ごとに行（節点）ループをブロック分割した場合の各ブロック内の非ゼロ要素の数の分布の評価

- 色数が増加すると、一つの色に属する節点数は少なくなる。
- 一つの色の計算が終わってから次の色の計算に行く必要がある。つまり、各色ごとで同期が必要。
- 色数の増加で同期のオーバーヘッドが増加すると予想される。
- Precond33の色ごとのブロック分割（!$omp do）の場合、色数の増加で1スレッドで担当するブロック幅は小さくなる。
- Precond33の色ごとのブロック分割は、行全体としてはブロックサイクリック分割に相当。従って、色数の増加でサイクリック分割に近づき、キャッシュミスが増大すると予想される。







ロードバランス及びアクセスパターンを均等化したブロックサイクリック分割の試行

matvec33: 行（節点）iのループの分割方法の検討：ロードバランス及びアクセスパターンを均等化したブロックサイクリック分割の試行

```
!$omp do          行に関する単純なブロック分割
    do i=1,N
```

```
!$omp do schedule(static,1)    行に関する単純なサイクリック分割
    do i=1,N
```

```
!$omp do schedule(static,100)
    do i=1,N
```
行に関する単純なブロックサイクリック分割

```
    threadNum = omp_get_thread_num()
    do i= startPos(threadNum), endPos(threadNum)
```

スレッド0から7の繰り返しのサイクル数は100固定とする。全部で800ブロックとなる。

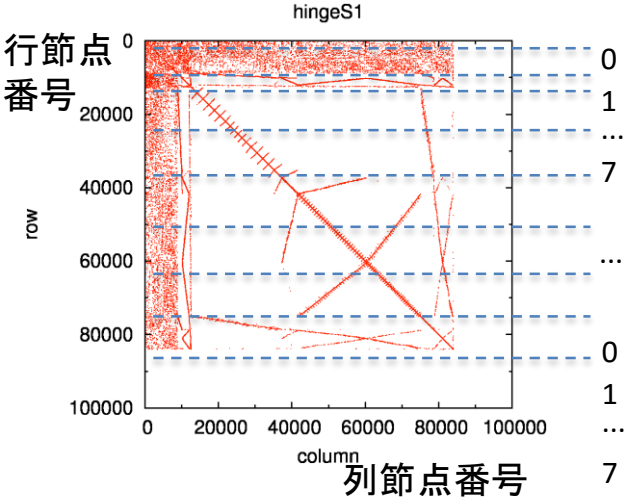
この方法を試す。

”非ゼロ要素数の均等分割”：
ロードバランスを均等化したブロック分割
各スレッドがおよそ同じ非ゼロ要素数を処理する

```
numOfBlockPerThread = 100
numOfThread = omp_get_max_threads()
threadNum = omp_get_thread_num()
do blockNum = 0 , numOfBlockPerThread - 1
  blockIndex = blockNum * numOfThread  + threadNum
  do i = startPos(blockIndex), endPos(blockIndex)
```

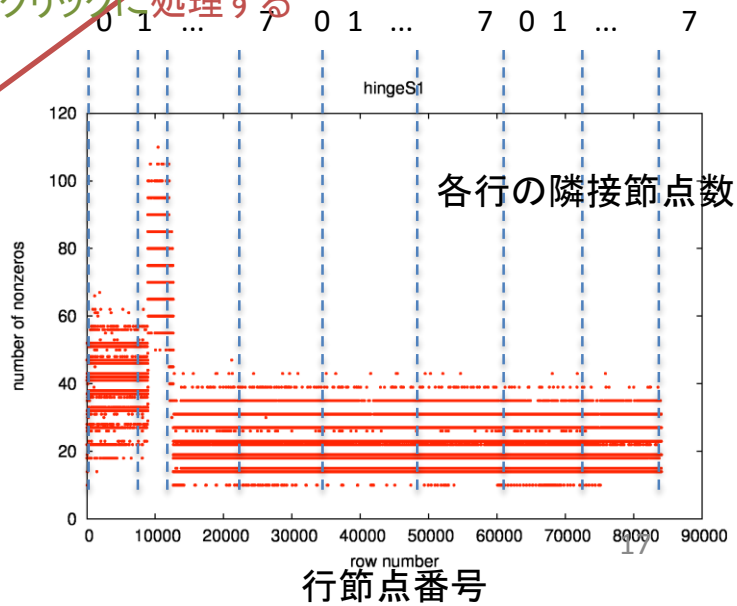
”非ゼロ要素数のブロックサイクリック均等分割”：
ロードバランス及びアクセスパターンを均等化したブロックサイクリック分割

各スレッドがおよそ同じ非ゼロ要素数をブロックサイクリックに処理する



スレッド間での処理量が均等化できる。

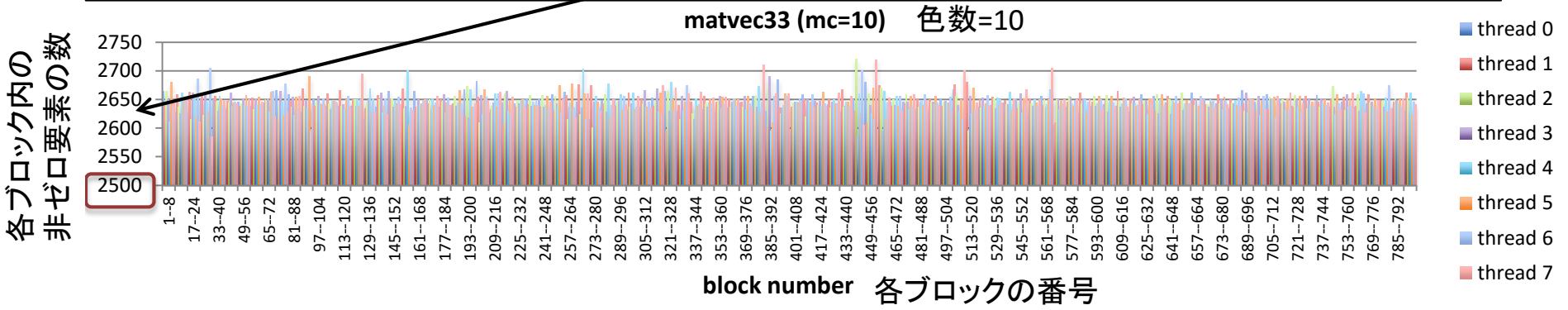
ベクトルのアクセスパターンの均等化の向上が期待できる。

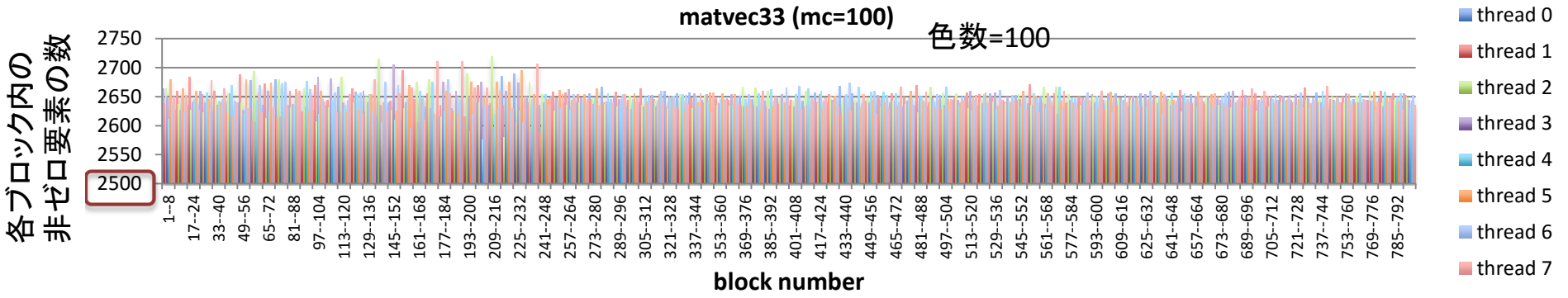


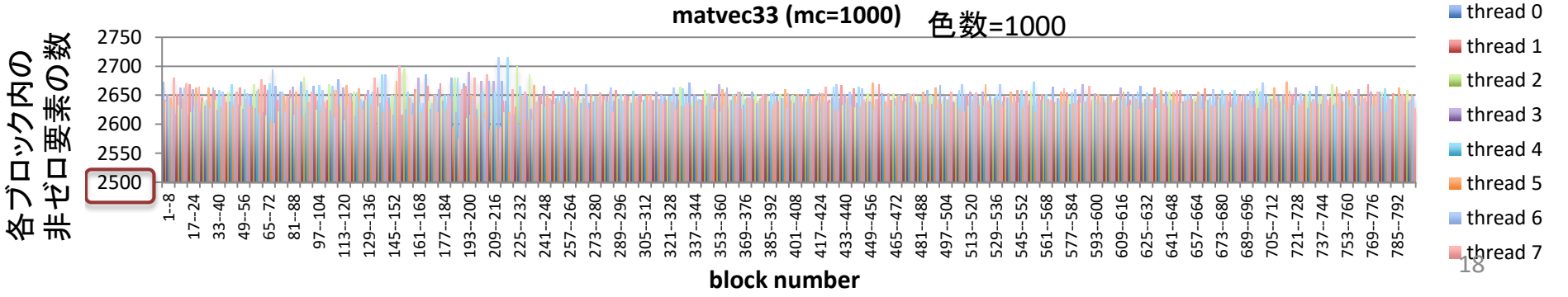
Matvec33

非ゼロ要素数のブロックサイクリック均等分割の場合の各ブロック内の非ゼロ要素の数の分布の評価

3x3ブロックを1要素と数えるとして、非ゼロ要素の総数/ブロックの数 = 2115968/800 = 2644.96 が、
1ブロック当りが処理する非ゼロ要素の平均的な値（この値を基準値として各ブロックのブロック幅を決める）







precond33: 行（節点）iのループの分割方法の検討：ロードバランス及びアクセスパターンを均等化したブロックサイクリック分割の試行

行に関する単純なブロック分割

```
   do ic=1,NCOLORTot
!$omp parallel
!$omp do
     do i=COLORindex(ic-1)+1,COLORindex(ic)
```

この方法を試す。

”非ゼロ要素数のブロックサイクリック均等分割”：ロードバランス及びアクセスパターンを均等化したブロックサイクリック分割

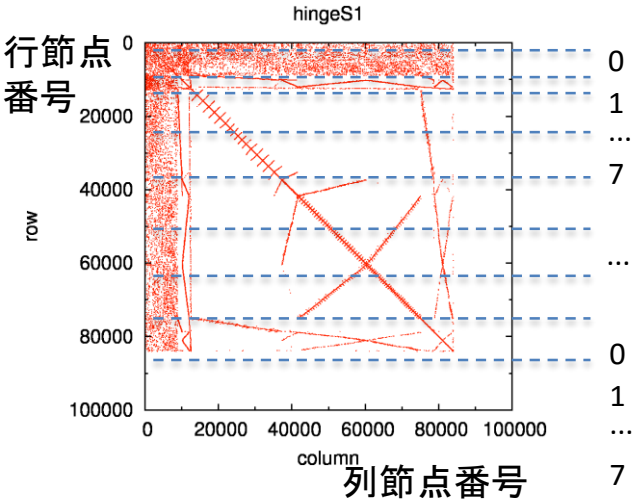
```
!$omp parallel
  do ic=1,NCOLORTot
!$omp do schedule (static,1)
    do blockIndex = icToBlockIndex(ic-1)+1, icToBlockIndex(ic)
      do i = blockIndexToColorIndex(blockIndex-1)+1, blockIndexToColorIndex(blockIndex)
```

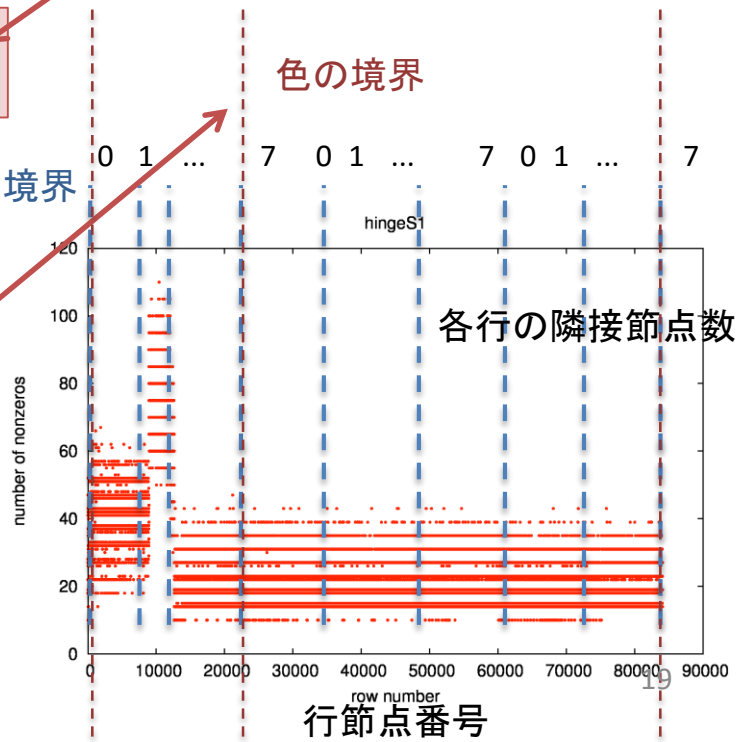
各スレッドがおよそ同じ非ゼロ要素数をブロックサイクリックに処理する

スレッド間での処理量が均等化できる。

ベクトルのアクセスパターンの均等化の向上が期待できる。

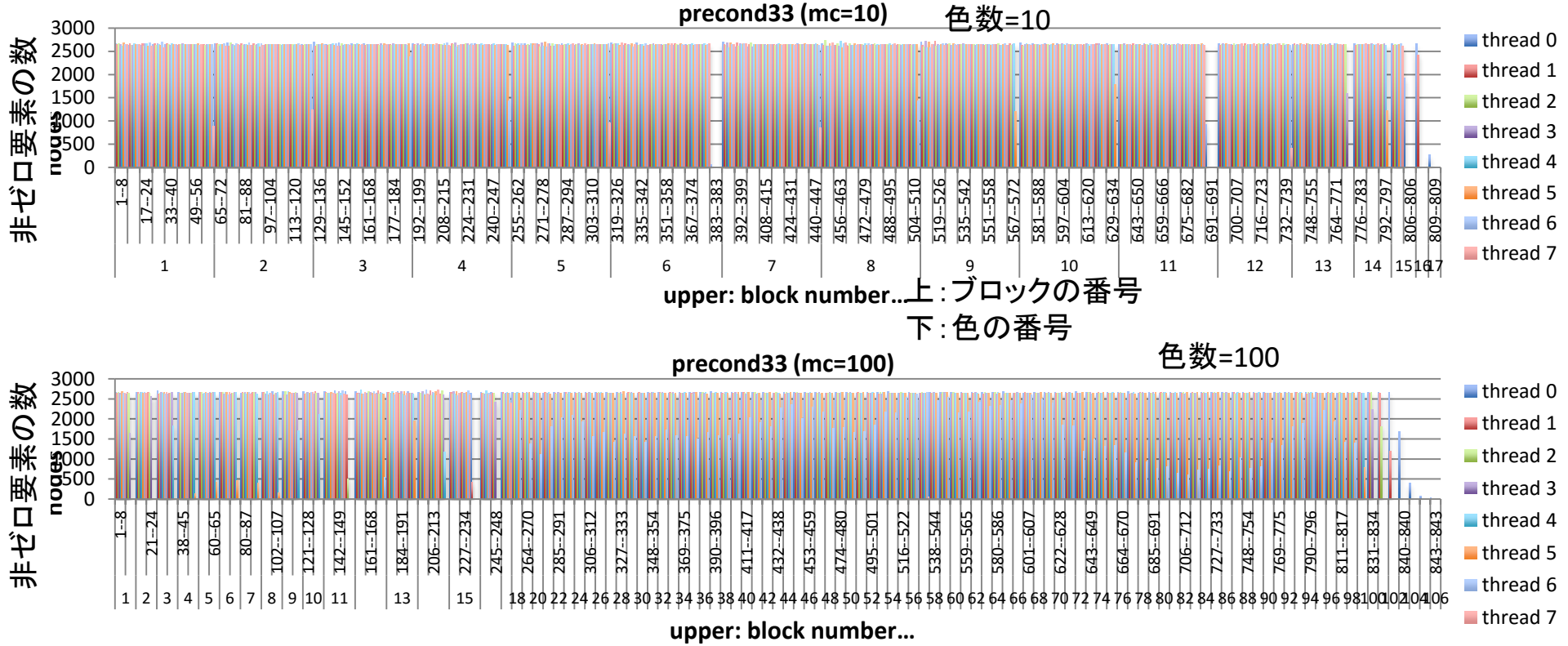
matvec33の時と違って、precond33ではブロックの境界が色の境界に来るようにブロック幅を決める必要がある。

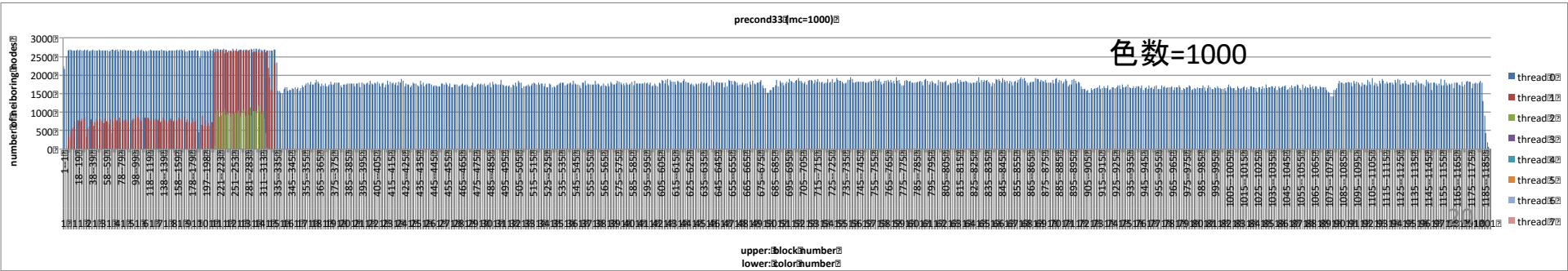




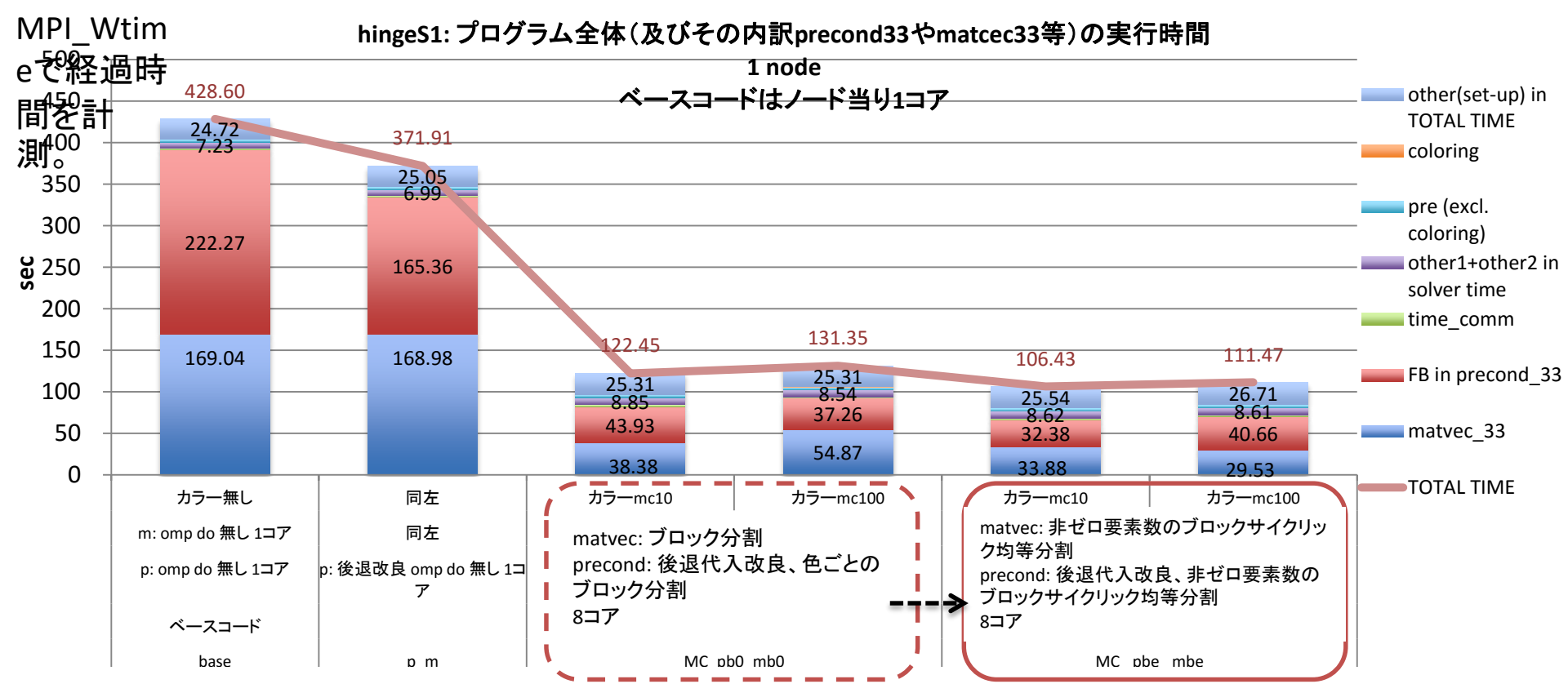
precond33: 非ゼロ要素数のブロックサイクリック均等分割の場合の各ブロック内の非ゼロ要素の数の分布の評価

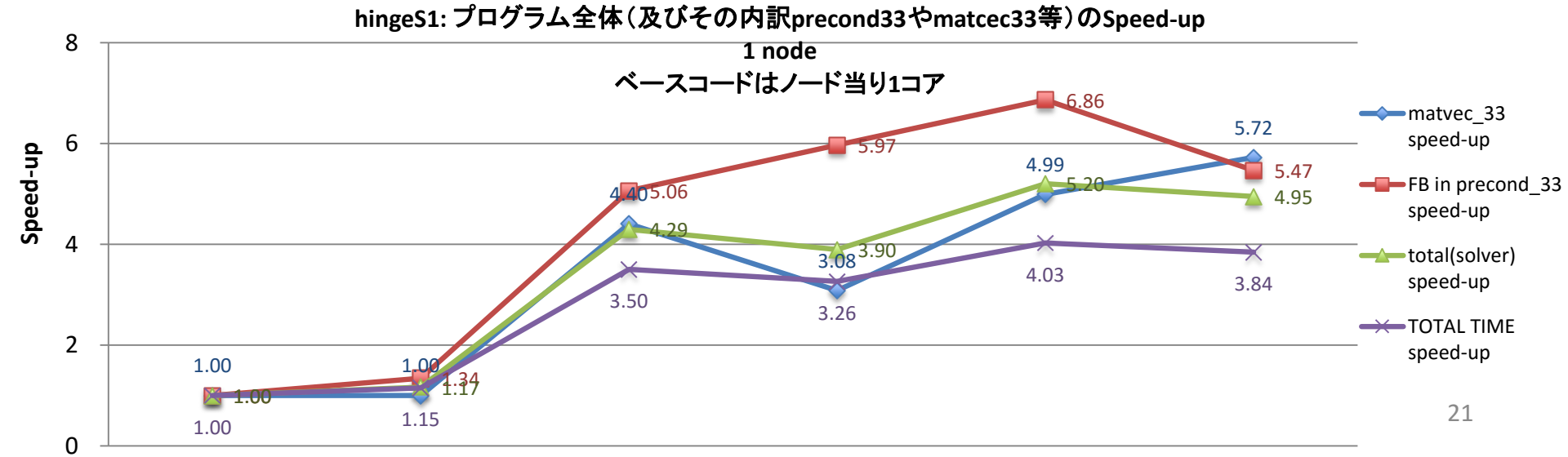
- hingeS1に対しては、色数が増加すると遊んでいるスレッドが多くなる。非ゼロ要素数のブロックサイクリック均等分割が有効なのは、hingeS1に対して色数10程度までと予想される。



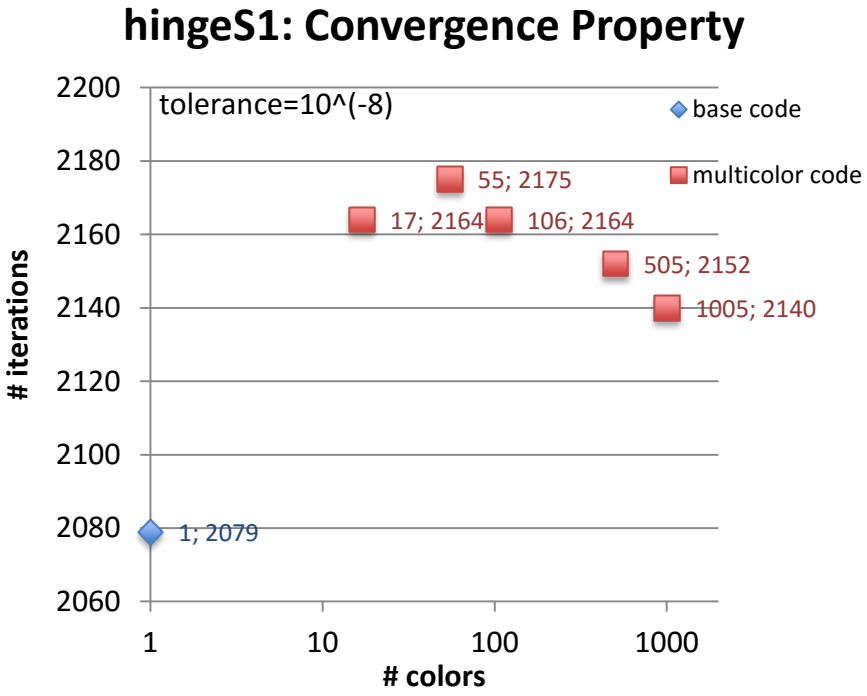


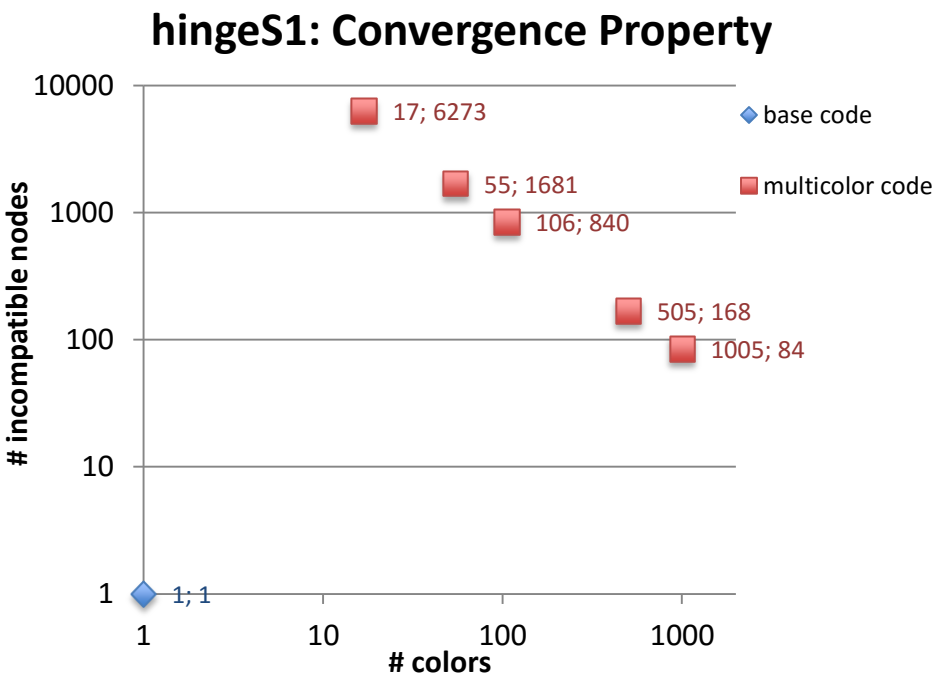
MPI_Wtimeで経過時間を計測。





付録

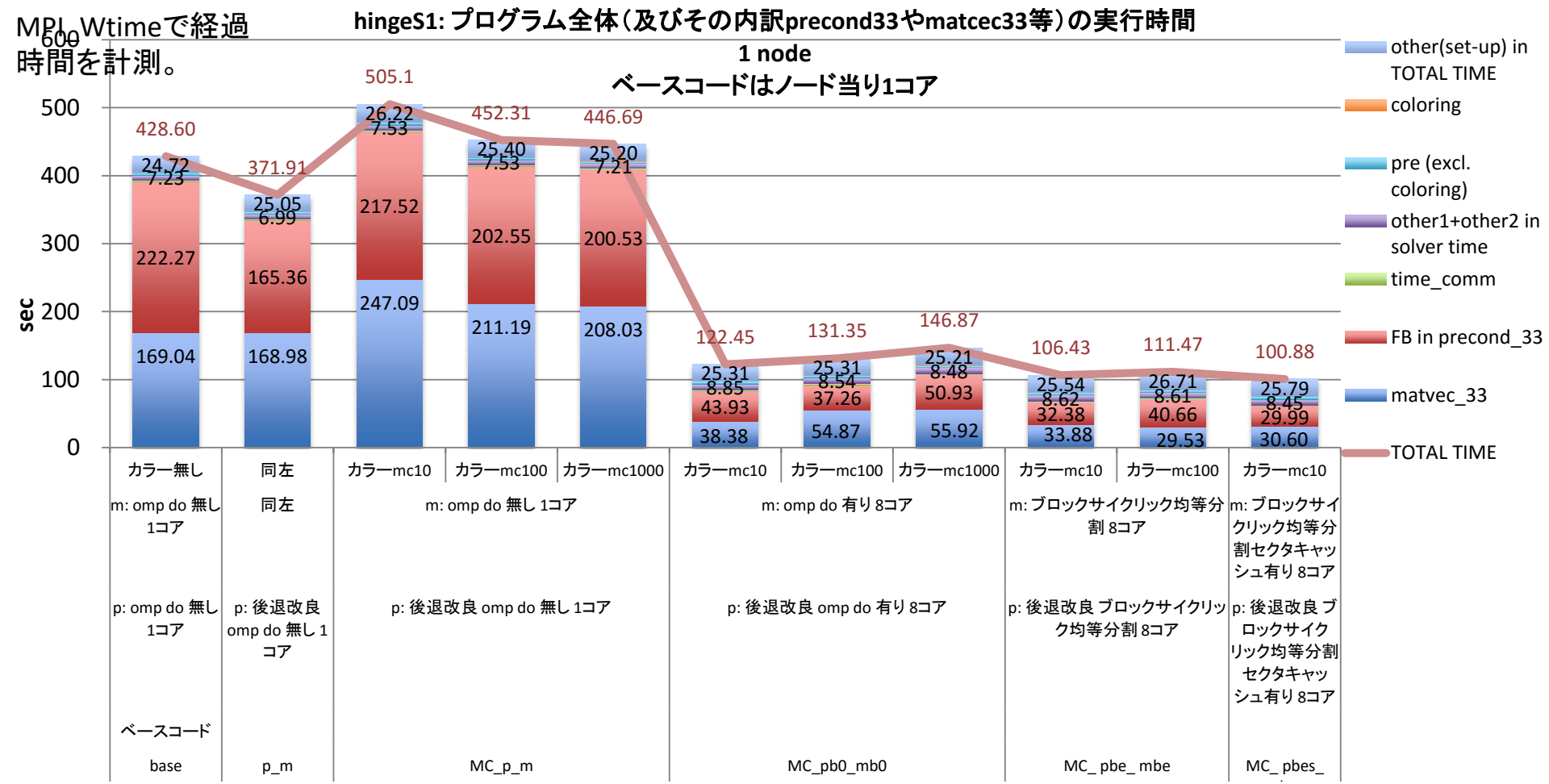


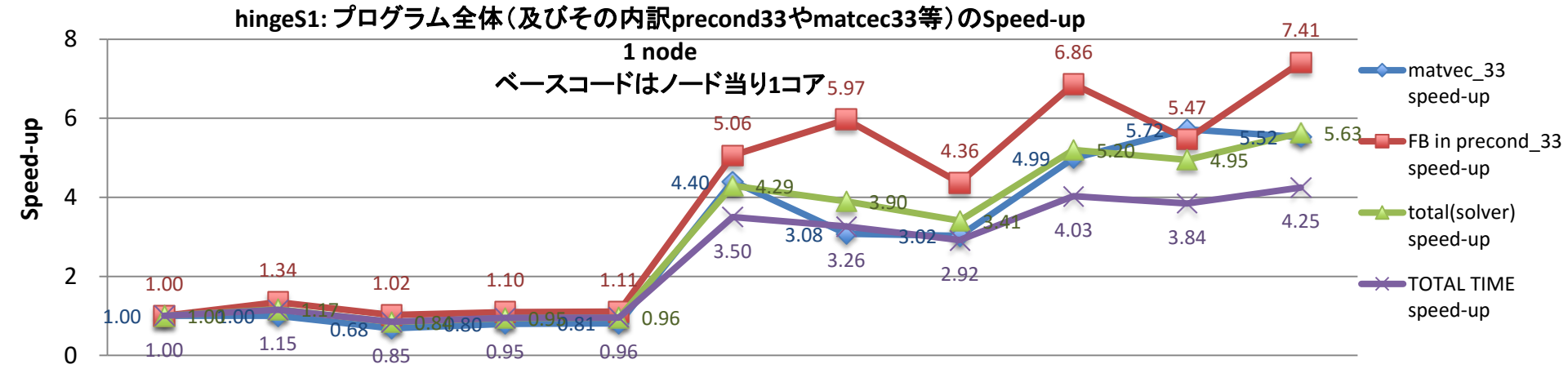


MC法の特徴

- 色数が少ないと反復回数が多くなる傾向がある。
- 色数を増やすと反復回数は減るが、同期オーバーヘッドの影響が大きくなる。

MPI_Wtimeで経過時間を計測。





matvec_33のコードとhingeS1メッシュからの特性分析

	matvec33一回当りのFLOP/節点	matvec33一回当りの要求バイト数/節点	matvec33一回当りの要求B/F	推定限界ピーク性能比
hingeS1	450.12	2553.33	5.67	6.35%

precond_33のコードとhingeS1メッシュからの特性分析

	precond33一回当りのFLOP/節点	precond33一回当りの要求バイト数/節点	precond33一回当りの要求B/F	推定限界ピーク性能比
hingeS1	936.24	5346.66	5.71	6.30%

0.36/5.67=6.35%

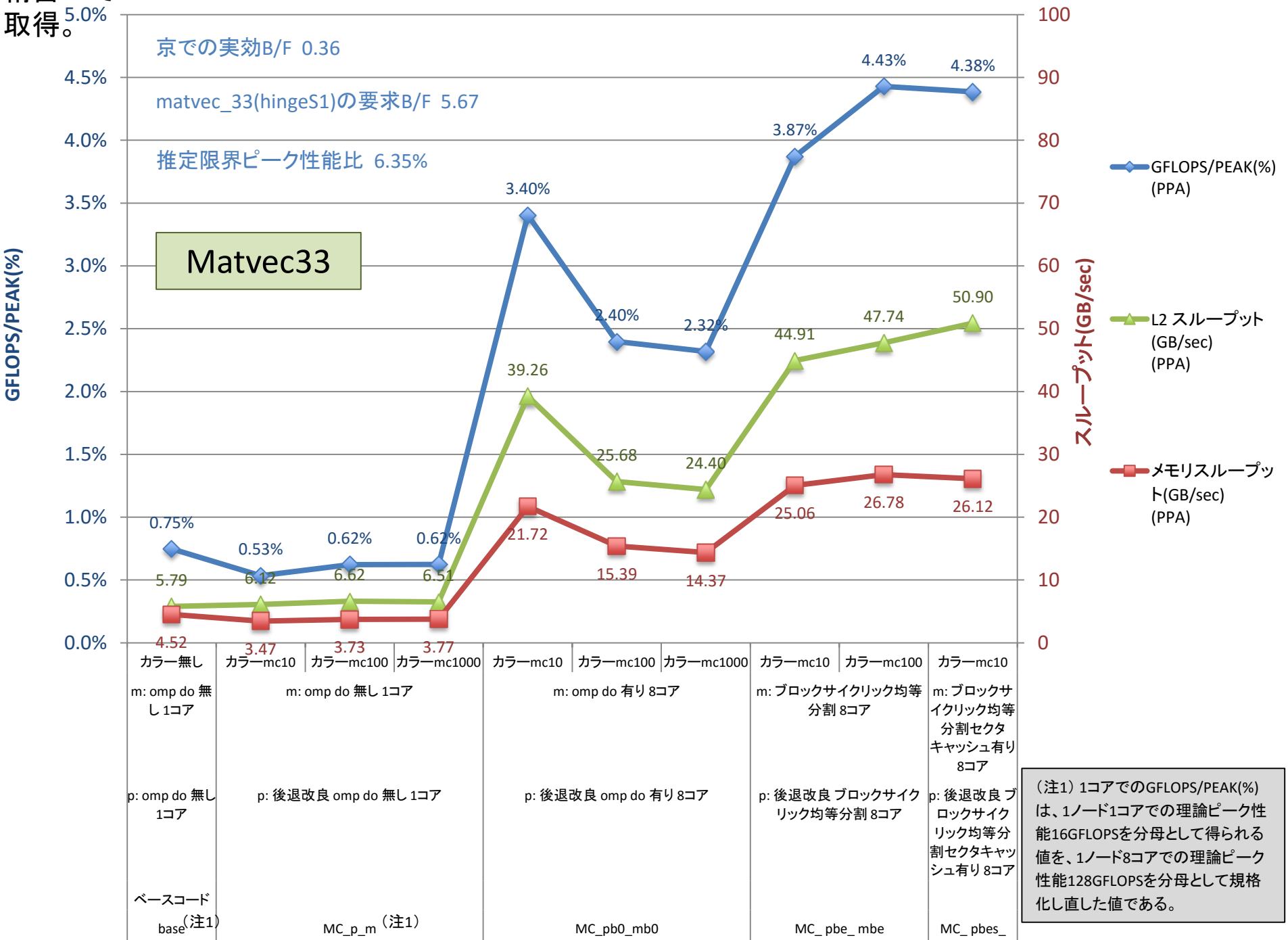
0.36/5.71=6.30%

京のハードウェアの性能

京での1ノード8コアでの理論ピーク性能 GFLOPS	京での1ノード8コアでの理論バンド幅 GB/s	京の理論B/F	京での実効B/F
128	64	0.5	0.36

精密PAで取得。

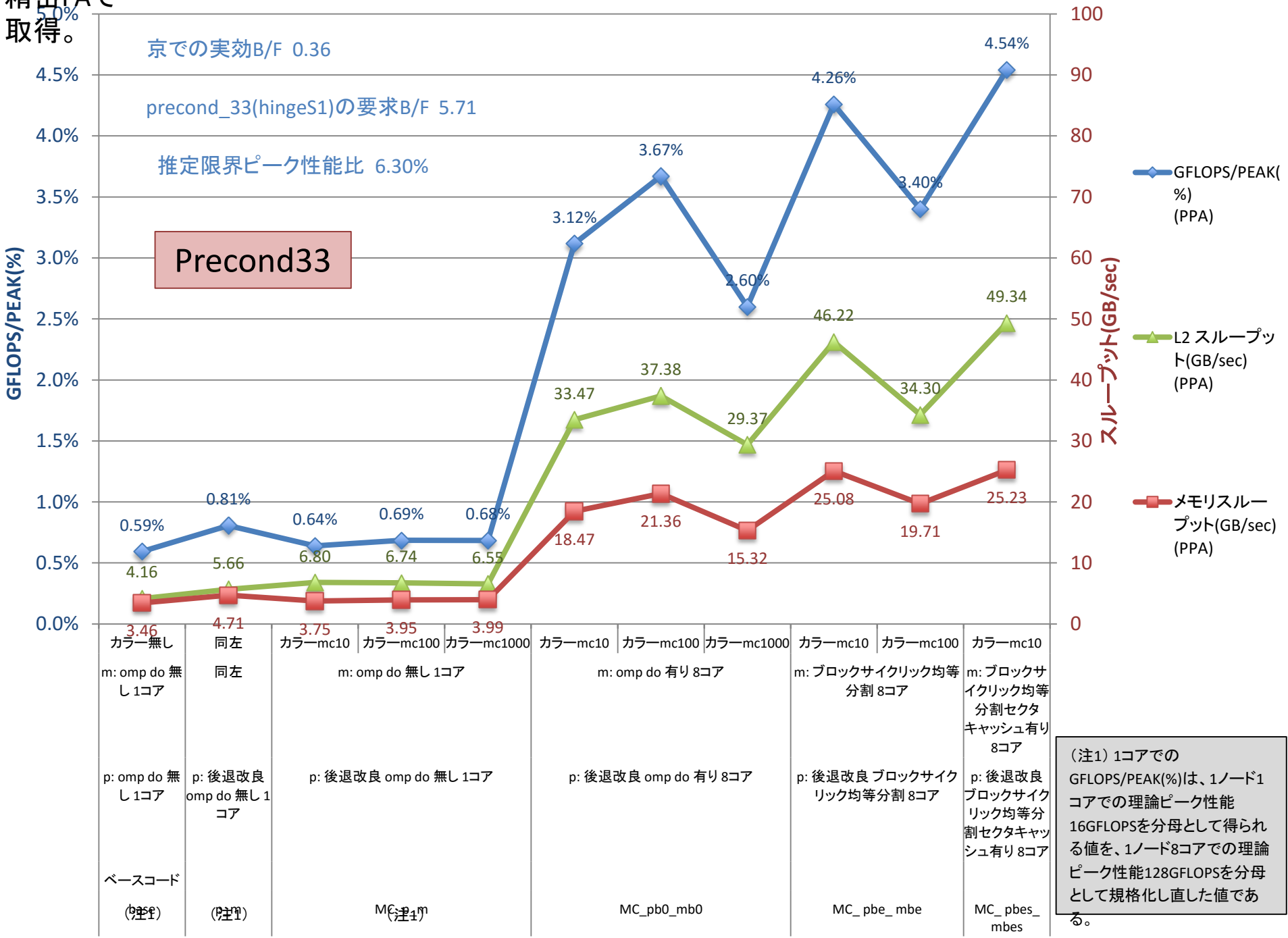
hingeS1: matvec_33: 1ノードでの実行のピーク性能比及びスループット



（注1）1コアでのGFLOPS/PEAK(%)は、1ノード1コアでの理論ピーク性能16GFLOPSを分母として得られる値を、1ノード8コアでの理論ピーク性能128GFLOPSを分母として規格化し直した値である。

精密PAで取得。

hingeS1: precond_33: 1ノードでの実行のピーク性能比及びスループット



（注1）1コアでのGFLOPS/PEAK(%)は、1ノード1コアでの理論ピーク性能16GFLOPSを分母として得られる値を、1ノード8コアでの理論ピーク性能128GFLOPSを分母として規格化し直した値である。