Vol.2016-HPC-153 No.2
2016/3/1

非構造格子に対応した PCG 法の thread 並列化手法

内山学[†1]　ファム　バン　フック[†1]

分散並列計算を行う場合，通信負荷軽減の為には MPI 並列と Thread 並列を併用した Hybrid 並列が望ましい．構造解析や流体解析で計算時間の大部分を占めるのが連立方程式解法であり，係数行列の不完全分解を前処理行列とした共役勾配法が多く用いられている．本報告では前処理部分の thread 並列化の為に新しい ordering 法を提案する．解析領域を部分領域に分割し，隣接する部分領域と接続する格子/節点を切取り,切取った格子/節点を更に部分領域に分割して同様の操作を行う．本方法は，全体を Cuthill-Mckee 法で reordering した場合と比較して，収斂性の悪化が少ない．前処理行列に対する前進・後退代入演算と行列ベクトル積の計算方法についても提案と検討を行う．

Thread parallelization of Preconditioned CG Method for Unstructured Grids

MANABU UCHIYAMA[†1]　PHAM VAN PHUC[†1]

Hybrid parallelization, which is combination of MPI parallelization and thread parallelization, is necessary to decrease the communication between physical compute nodes. Preconditioned conjugate gradient (CG) method is commonly used to solve linear equations in FEM and CFD and Incomplete Cholesky factorization is frequently used for preconditioners. This paper proposes a new ordering method for thread parallelization of this preconditioner. Computational grid domain is divided to subdomains and the grids connected to other subdomains are cut off. Those cutoff grids are divided to subdomains and the grids connected to other subdomains are cut off. This process is done recursively until the number of cutoff grids becomes sufficiently small. A matrix has hierarchical structure and is stored in a special form. Number of iterations to convergence by using this method is almost the same as that by using Cuthill-Mckee ordering. Computational technique is also described.

1. はじめに

大規模な分散並列計算を行う場合に全体通信の負荷を軽減することが性能向上の点で重要である．その為には，計算時間の大部分を占める連立方程式解法に対して，全体通信回数の少ないアルゴリズム[1]によって同期回数を減らして性能向上を図る方法，全体通信を隠蔽するアルゴリズム[2]（MPI-3 が必要）による方法，計算ノード内を thread 並列化して hybrid 並列としてプロセス数を減らす方法が考えられる．本報告は，hybrid 並列への適用を目的として，fill-in 無しの不完全コレスキー分解共役勾配法（ICCG 法）の thread 並列化について検討を行う．なお，著者らは文献[3]で直交格子に限定して hybrid 並列の検討を行ったが，本報告では非構造格子を対象としている．

ICCG 法を thread 並列化する為には multicolor 法による reordering が行われることが多い. 文献[4,5]による方法は非構造格子にも適用が容易な優れた方法である．しかし，格子/節点を単位として multicolor 化すると ICCG 法の収斂性が悪化することが多い．一つのプロセスに与えられた領域をグループ分けし，グループを単位として multicolor 化すると収斂性の悪化は少ないと考えられるが，従来の multicolor 化する方法では各色に属するグループ数を制御するのは困難である．本報告では，文献[6]で FEM の剛性行列のアセンブルを thread 並列する為に開発した方法を ICCG 法の thread 並列化に応用する．更に，各グループ内の計算効率を高める方法を提案する．検討は流体解析システム OpenFOAM-2.2.x の解析ソルバーpisoFOAM を使用して行う．Thread 並列化ツールは OpenMP を使用する．

2. 格子のオーダリングと行列格納方法

2.1 文献[6]のグループ分け

文献[6]では, FEM の剛性行列のアセンブルを thread 並列化する為に，要素を図 1 のアルゴリズムでグループ分けしている．同じ level に属するグループ同士は独立である．図 2 は球殻をシェル要素でモデル化した例である．グラフ分割には METIS[7]の METIS_PartGraphKway 関数を用いている．図 1 の「3. グラフ圧縮」は，連結グラフに孤立した節点があると要素の取付かない節点だけのグループができることがあり，それを避けるためである．

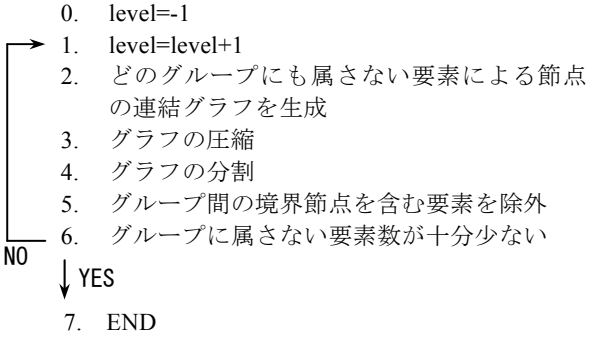


Figure 1 要素のグループ分けのアルゴリズム[6]

†1 清水建設株式会社
SHIMIZU Corporation

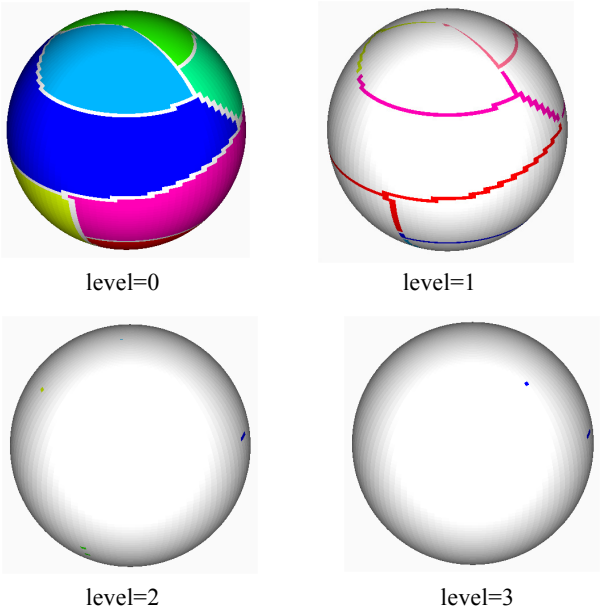


Figure 2 要素のグループ分け[6]

2.2 格子のグループ分け

前節の方法を格子のグループ分けに応用する．アルゴリズムを図 3 に示す．

0. level=-1
1. level=level+1
2. どのグループにも属さない格子について，既にグループ分けされている格子との接続を除外して格子の連結グラフを作成
3. グラフの分割
4. if(level=0) グループを単位として reordering
5. 自グループよりも小さい番号のグループに属する格子に接続する格子を切り取る
6. 切り取られた格子数が十分少ない
   NO → 1.
   YES ↓
7. END

Figure 3 格子のグループ分けのアルゴリズム

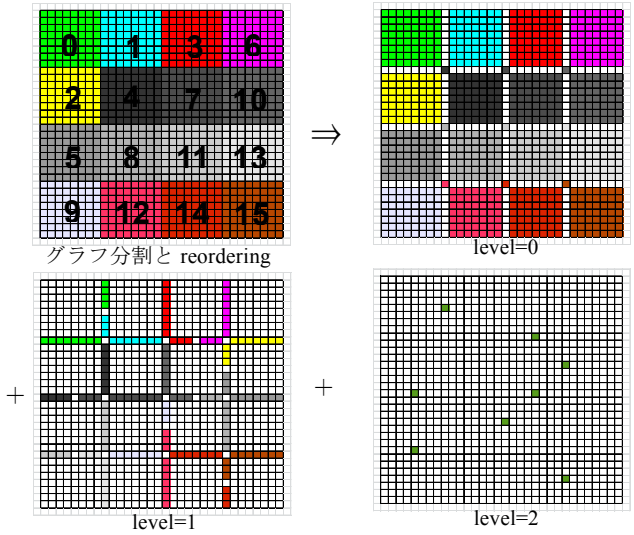


Figure 4 32×32 格子のグループ分け

図 4 は二次元直交格子の例(level=0, 1 のグループ数は, 夫々 16）である．辺を接する格子間にだけ連成項が生じる．この例では reordering に Cuthill-Mckee を使用しているが，数値実験では METIS_NodeND 関数を使用している．

図 5 は，後述する数値実験で使用する motorBike の格子を各 level のグループ数を 32, 4, 2 とした場合の係数行列のイメージ図である．非零項が存在するブロックが色付けされている．格子の 99%以上が level=0 と level=1 に含まれる．

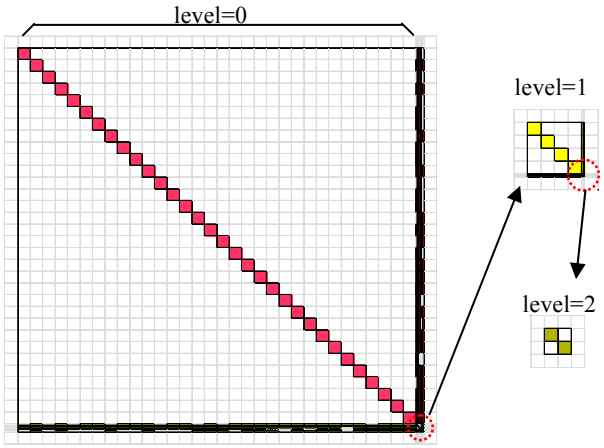
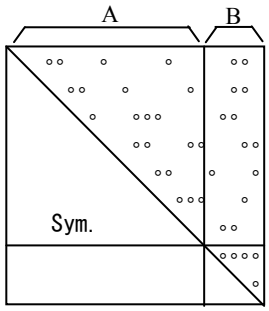


Figure 5 係数行列の非零項の分布イメージ

2.3 対角ブロックのオーダリング

係数行列の対角ブロック内のオーダリングは，Cuthill–Mckee (CM)，Reverse Cuthill-Mckee (RCM)，Sloan[8]等で行う方法が考えられるが，計算効率を高める為に図 6 に示すオーダリングを行う．上三角部分を考え，A の領域は U[*][m1]の二次元配列で記憶し，B の領域は非零項のみを記憶する．U[*][m1]の形にするに当たり，非零項数が m1 個に満たない行はダミーデータとして零を挿入する．対角ブロック内の計算量の増加には上限値を設定しておく．



0. 計算量の増加率の上限値αを設定
1. 対角ブロック内の連結グラフを作成
2. CM, RCM, Sloan 等で reordering を行う
3. 上三角部分に関して，行ごとに A の領域にある非零項の数をカウントし，最大値を m1 とする
4. if(演算量の増加<=α) GOTO 7.END
5. 3 で最大値を与えた行を B の領域に移動
6. GOTO 3
7. END

Figure 6 対角ブロック内のオーダリング

3. 計算方法

3.1 CG 法のアルゴリズム

CG 法のアルゴリズムは文献[2]の Chronopoulos and Gear の方法を使用する．図 7 にアルゴリズムを示す．(b)は全体通信の回数を減らすことを目的として提案されたものであるが，以下の利点がある．このアルゴリズムでは，前処理行列に関する計算の直後に行列ベクトル積の計算が行われるため，fill-in 無しの ICCG 法を使用する場合は前処理行列の後退代入計算と行列ベクトル積の計算をラップさせることができる．これにより，計算量と行列成分をメモリーから読み込む回数を減らすことができる．

1: $r_0 = b - Ax_0$;
2: for i=0, ···. do
3: $u_i = M^{-1} r_i$
4: $\gamma_i = (r_i, u_i)$
5: $\beta = \gamma_i / \gamma_{i-1}$
6: $p_i = u_i + \beta p_{i-1}$
7: $s = Ap_i$
8: $\delta = (s, p_i)$
9: $\alpha = \gamma_i / \delta$
10: $x_{i+1} = x_i + \alpha p_i$
11: $r_{i+1} = r_i - \alpha s$
12: Residual = $| r_{i+1} |$
13: end for

(a)通常版

1: $r_0 = b - Ax_0$;
2: for i=0, ···. do
3: $u_i = M^{-1} r_i$
4: $w_i = Au_i$
5: $\gamma_i = (r_i, u_i)$ (*)
6: $\delta = (w_i, u_i)$
7: $\beta = \gamma_i / \gamma_{i-1}$
8: $\alpha_i = \gamma_i / (\delta - \beta \gamma_i / \alpha_{i-1})$
9: $p_{i+1} = u_i + \beta p_i$
10: $s_{i+1} = w_i + \beta s_i$
11: $x_{i+1} = x_i + \alpha_i p_i$
12: $r_{i+1} = r_i - \alpha_i s_i$
13: Residual = $| r_{i+1} |$
14: end for

(b) Chronopoulos and Gear

Figure 7　Preconditioned CG（□：全体通信）

3.2 行列の対角ブロック

図 8，9 は図 6 の A 部分に関するプログラムである．二次元配列の整合寸法 m1 が 0～8 の場合を用意している．なお，γ は式(1)により前進代入計算と同時に計算を行う．

$$\gamma = (r, u) = (Uu)^{\mathrm{T}} (Uu) \tag{1}$$

```
for (register int j=0; j<n1; j++) {
  int i0=iL[j][0], i1=iL[j][1], i2=iL[j][2];
  double s1 = x[j];
  x[j]    *= rD[j];
  x[i0]   -= L[j][0]*x[j];
  x[i1]   -= L[j][1]*x[j];
  x[i2]   -= L[j][2]*x[j];
  gamma   += s1*x[j];
}
```

Figure 8 前進代入計算（m1=3）

```
for (register int i=n1-1; i>=0; i--) {
  int j0=ju[i][0], j1=ju[i][1], j2=ju[i][2];
  double s1 = U[i][0]*x[j0]+U[i][1]*x[j1]+U[i][2]*x[j2];
  x[i]    -= rD[i]*s1;          ← 後退代入
  z[i]    += D[i]   *x[i] +s1;
  z[j0]   += U[i][0]*x[i];
  z[j1]   += U[i][1]*x[i];      } 行列ベクトル積
  z[j2]   += U[i][2]*x[i];
}
```

Figure 9 後退代入計算と行列ベクトル積（m1=3）

数値実験での比較の為に，対角ブロック内の上三角部分を単純な CRS (Compressed Row Storage)方式で記憶する場合の計算方法を図 10, 11 に示しておく．なお，対称行列であるから，下三角部分は CCS (Compress Column Storage)となる．

```
for(register int k=0; k<nz0; k++) {
  int i=iu[k], j=ju[k];
  x[j] -= rD[j]*U[k]*x[i];
}
for(register int i=0; i<n0; i++) {
  gamma += x[i]*x[i]*DD[i];
}
```

Figure 10　前進代入計算

```
for(register int fi=nz0-1; fi>=0; fi--) {
  int i = ip1[fi], j = jp1[fi];
  double s0= U[fi]*x[j];
  x[i]    -= rD[i]*s0;
  z[i]    += s0;
}
for(register int fi=0; fi<nz0; fi++) {
  int i = ip1[fi], j = jp1[fi];
  z[j] += U[fi]*x[i];
}
for(register int i=0; i<n0; i++) {
  z[i] += DD[i]*x[i];
}
```

Figure 11　後退代入計算と行列ベクトル積

4. 数値実験

4.1 計算ケース

CG 法の収斂性と計算時間の比較を行うために，表 1 のケースを比較する．直交格子の場合は係数行列の上三角，下三角の部分を，夫々U[*][3], L[*][3]の形に格納できる為，

Table 1　解析ケース

name	各 level のグループ数	説明
CM0	—	係数行列全体を CM ordering し，図 10, 11 の方法で計算．但し，下三角部分を別途 RCS で記憶し，gamma の計算は同時に行わない．
EGMC	—	格子を単位として multicolor 化し，並列計算時は各色内を 32 分割して thread に割当てる．計算方法は CM0 と同じ．
b128_CM0	128, 32, 2	対角ブロック内を CM ordering. 係数行列全体を CM0 と同様に計算. 並列計算時はグループ単位で thread に割当てる．
b128_CM1	128, 32, 2	対角ブロック内を CM ordering し，図 10, 11 の方法で計算．
b128_S0	128, 32, 2	対角ブロック内を Sloan ordering し，図 6 の方法で並び換える．b128_CM0 と同じ方法で計算．
b128_S2	128, 32, 2	b128_S0 に対して対角ブロック内を図 8, 9 の方法で計算．
b256_S2	256, 32, 0	b128_S2 と同じ計算方法．

EGMC は計算を行単位に展開して行方向に loop を回すことで SIMD 計算可能であるが，非構造格子ではそのようなことはできない．また，各色内で行単位に loop を回してその外側の loop で並列化する方法も考えられるが，流体解析では最内側の loop 回転数が小さい為に非効率である．格子単位で multicolor 化するメリットは少ないと考える．EGMC は ICCG 法の収斂性比較の為の計算ケースである．

4.2 計算環境

Intel Xeon E5-2697 v2 (2.70GHz, 12 cores, 30 MB Cache)を 2 基搭載した計算機を使用する．メモリーは 128GB である．OS は CentOS-6.4，コンパイラは Intel C/C++ Compiler v13.1.3.192 である．

4.3 高層ビル

解析モデルを図 12 に示す．高層ビルとその周辺街区，及び影響が大きい超高層ビルをモデル化している．表 2 に諸元を示す．時間増分幅は 0.0001(s)とし，最初の 2 step の圧力方程式を解く ICCG 法の収斂計算回数と計算時間を比較する．圧力方程式は相対残差 0.05 を収束判定値として計算した後，係数行列の修正を行って相対残差 10^{-6} の収束判定値で計算している．計算時間は 5 回計算を実行して平均値を算出している．

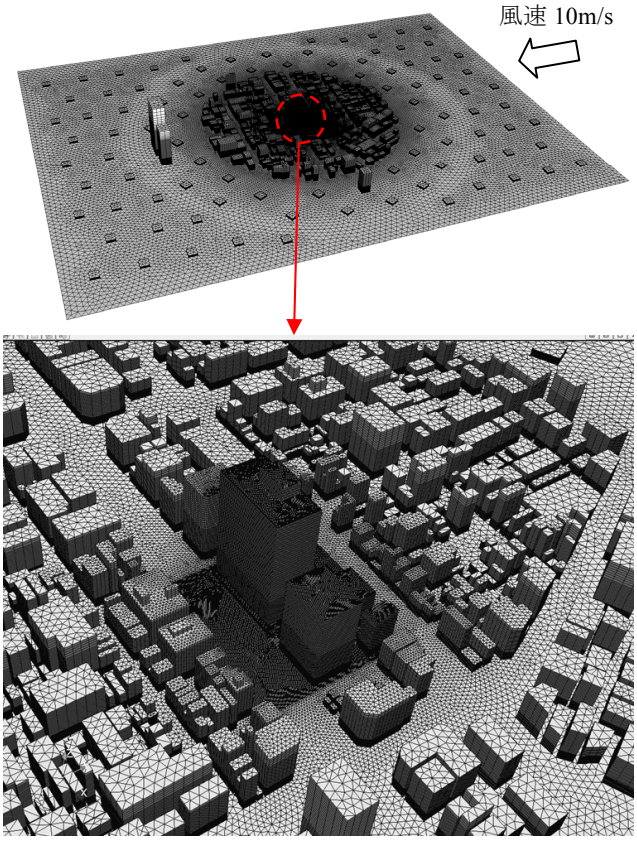


Figure 12 高層ビルと周辺街区

Table 2 計算領域と格子数

計算領域 (m)	1,500×2,000×500	
格子数	9,973,802	
格子種類と数	hexahedra	237,952
	prisms	9,735,850

EGMC は文献[4]の方法を使用して 6 色に multicolor 化できた．b128_*と b256_S2 の各 level の格子数の比は，夫々（0.96, 0.04, 10^{-7}），（0.95, 0.05, 0）である．

表 3 に総収斂計算回数を示す．格子を単位として multi-color 化した EGMC は CM0 の 1.56 倍の収斂計算回数となり，著しく収斂性が悪化している．一方，b128_*と b256_S2 は僅かに増加しているだけである．ICCG 法の計算は 2 step で 4 回行われているが，各回とも同じ傾向である．

計算時間を図 13 に示す．1 thread での計算に関しては，b128_CM0 と b128_S0 は CM0 よりもバンド幅が広がったことで計算時間が増加している．これはキャッシュミスが起こりやすくなった為と考えられる．b128_CM1 は対角ブロ

Table 3 総収斂計算回数

計算ケース	総収斂計算回数	CM0 に対する比
CM0	2519	—
EGMC	3619	1.56
b128_CM0/1	2538	1.01
b128_S0/2	2552	1.01
b256_S2	2584	1.03

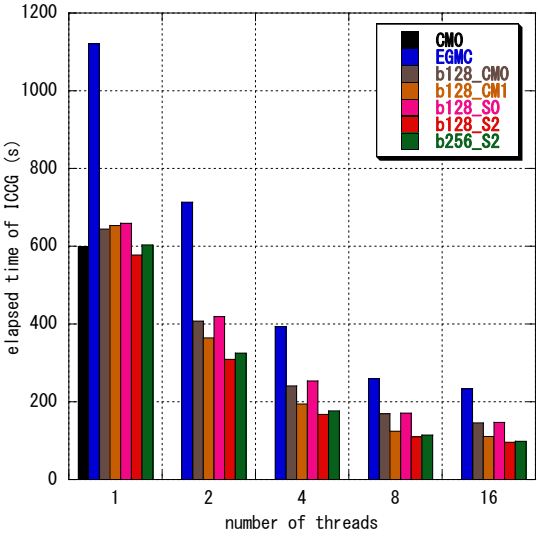


Figure 13 Elapsed time of ICCG

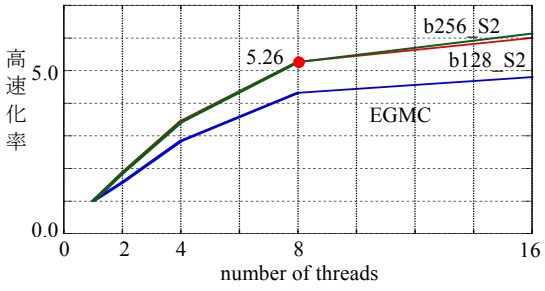


Figure 14 高速化率

ック内を分けて計算している為，b128_CM0 と b128_S0 に比べてキャッシュミスに関しては改善されているはずであるが，計算の制御が複雑となって計算時間がより増加したと考える．ただし，thread 並列化した場合は b128_CM1 の計算時間が短い．b128_S2 と b256_S2 は対角ブロック内を図 8, 9 の方法で計算することで計算効率の改善を行い，1 thread の場合には CM0 と同等かやや少ない計算時間である．Thread 並列化した場合も他の方法よりも速い．図 14 は thread 並列計算時の高速化率である．b128_S2 と b256_S2 は 8 threads 時に 5.26 である．

4.4 motorBike

OpenFOAM の tutorial に用意されているモデルである．解析モデルを図 15 に，表 4 に諸元を示す．Polyhedra（多面体）が多用された複雑な格子となっている．時間増分幅は 0.0001(s)とし，最初の 10 step の圧力方程式を解く ICCG 法の収束計算回数と計算時間を比較する．収束判定条件等は前節のモデルと同じである．b128_*の各 level の格子数の比は（0.96, 0.04, 10^{-4}）である．

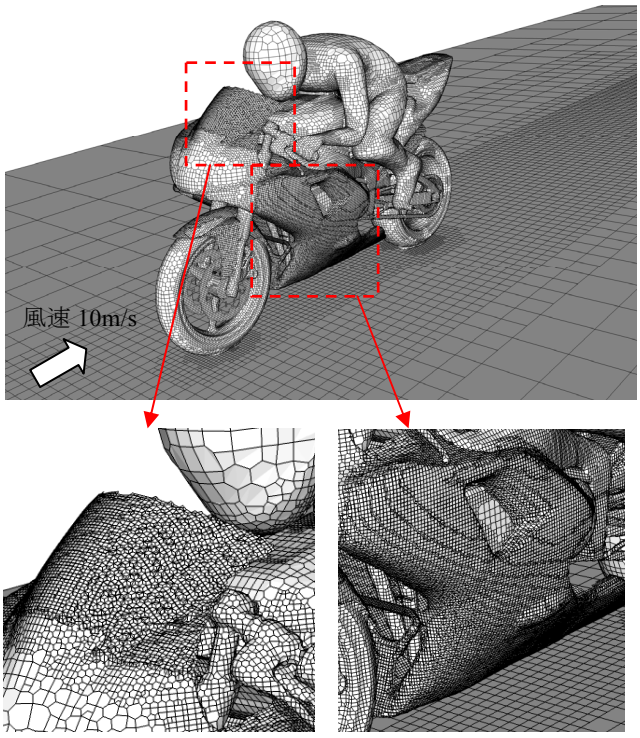


Figure 15 motorBike

Table 4 計算領域と格子数

計算領域 (m)	20×8×8	
格子数	4,601,581	
格子種類と数	hexahedra	3,897,137
	prisms	107,368
	wedges	19,160
	pyramids	535
	tet wedges	21,499
	tetrahedral	444
	polyhedra	555,438

表 5 に総収束計算回数を示す．b128_S0/2 は CM0 よりも僅かだが収束計算回数が減少し収束性が良くなっている．ICCG 法の計算は 10 step で 20 回行われているが，各回とも同じ傾向である．

計算時間と thread 並列化した場合の高速化率を図 16，17 に示す．計算時間は b128_S2 が最も少なく，高速化率は 8 threads 時で 5.33 である．

Table 5 総収束計算回数

計算ケース	総収束計算回数	CM0 に対する比
CM0	3665	―
b128_S0/2	3536	0.96

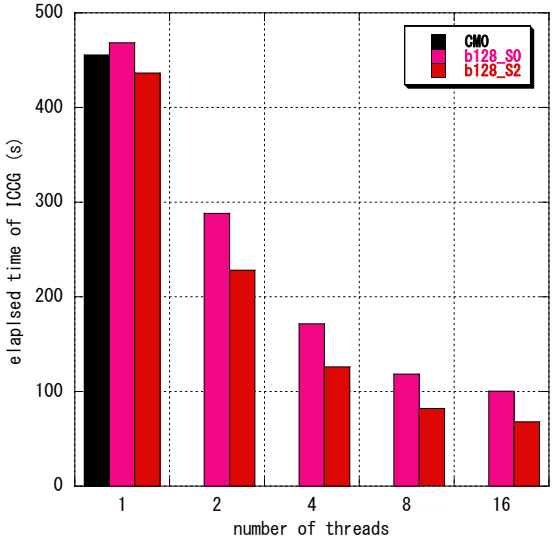


Figure 16 Elapsed time of ICCG

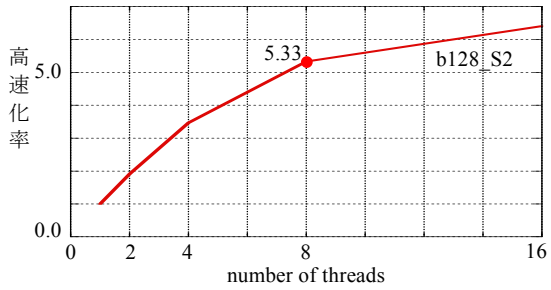


Figure 17 高速化率

4.5 数値実験のまとめ

EGMC では ICCG 法の収束性は著しく悪化し，高速化率も余り良くない結果である．

b128_S2 は，CM0 と比べて ICCG 法の収束性を殆ど悪化させず，thread 並列化しない状態で計算時間を約 5%短縮できている．対角ブロック内の非零項の記憶方法と計算方法を工夫することで計算効率が高まった為と考える． Thread 並列時の性能は，8 threads 時で高速化率が 5.3 と良い結果と言える．しかし，16 threads では高速化率の伸びが良くない．高層ビルのモデルでは b256_S2 も同様の傾向であるこ

とから，キャッシュメモリーの不足ではなく，メモリー帯域が不足してきた為と推測する．

使用する計算機のキャッシュメモリーの大きさやメモリー帯域によって thread 数と対角ブロックの大きさを調整する必要があるだろう．本報告で提案した方法では，対角ブロックの大きさは各 level のグループ数を指定することで制御可能である．

5. 結語

非構造格子を対象として，ICCG 法の thread 並列化を行った．提案の方法は，並列に計算できるグループ数を制御でき，収斂性を悪化させず，行列内の対角ブロックの計算方法を工夫することで性能を向上させることが示された．

今後，OpenFOAM の他の部分の thread 並列化も行い，MPI 並列と Hybrid 並列の比較を行う予定である．FEM 解析システムへの本手法の展開も行いたい．

参考文献

1) 本谷徹, 須田礼仁, k 段飛ばし共役勾配法 : 通信を回避することで大規模並列計算で有効な対象正定値疎行列連立 1 次方程式の反復解法, 情報処理学会報告, vol.2012-HPC-133, No.30, 2012.
2) P. Ghysels, W. Vanroose, Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm, Parallel Computing vol.40, pp224-228, 2014.
3) 内山学, ファム　バン　フック, 千葉修一, 井上義昭, 浅見暁, OpenFOAM による流体コードの Hybrid 並列化の評価, 第 151 回ハイパフォーマンスコンピューティング報告発表会, 2015.
4) T. Iwashita, M. Shimasaki, Algebraic multicolor ordering for parallelized ICCG solver in finite-element analyses, IEEE transactions on Magnetics, vol.38, No.2, pp429-432, 2002.
5) 岩下武史, 高橋康人, 中島浩, 代数ブロック化多色順序付け法による並列化 ICCG ソルバの性能評価, 情報処理学会研究報告, vol.2009-HPC-121, No.11, 2009.
6) 内山学, 構造解析システムの SMP 計算機上での並列化について – その 2, 日本建築学会学術講演梗概集（東北）, 2009.
7) http://glaros.dtc.umn.edu/gkhome/views/metis
8) S. W. Sloan, A FORTRAN program for profile and wavefront reduction, Comput. Struct., vol.33, no.2, 2651-2679, 1989.