シミュレーションソフトのスレッド並列化と高速化

内山　学
（技術研究所）

Thread Parallelization and Improvement of Performance of Simulation Software

Manabu Uchiyama

大規模な流体解析やFEM解析では計算時間が膨大となる為、その短縮が望まれている。計算時間の大部分は連立方程式解法の部分である。流体解析システムのOpenFOAMに対しては、不完全LU分解を用いたsmootherをスレッド並列化する為の疎行列の並び替え方法を提案する。また、CG法のアルゴリズムの改良、他の部分の高速化についても示す。FEM解析システムのFrontISTRに対しては、そこで使用されているAMG法前処理CG法の改良について述べ、その著しい性能向上を示す。

Decreasing computational time is desired because that of large-scale simulation of CFD or solid mechanics becomes enormous. Almost computational time is spent on solving simultaneous equations. For OpenFOAM, which is a CFD software, the modified algorithm of conjugate gradient method and improvement of other parts are presented. For FrontISTR, which is a FEM software of solid mechanics, improvement of AMG preconditioned CG method is presented, and its performance is shown.

1. はじめに

大規模な流体解析やFEM解析では計算時間が膨大となる為、その短縮が望まれている。計算時間の大部分は反復解法による連立方程式解法の部分であり、前処理に係数行列の不完全LU分解やAMG法を用いたCG法が使われることが多い。分散並列計算とすることで計算時間の短縮が可能であるが、領域数が増加すると収斂性の悪化や通信による遅延が目立ってくる為、スレッド並列計算と併用することで、通信負荷を抑えながら高速化することが考えられる。しかし、前進後退代入計算を行う前処理のスレッド並列化は難しく、例えば、スレッド並列可能なように疎行列を並び替える方法としてAlgebraic Multicolor[1]が提案されているが、収斂性の悪化が見られる[2]。

本報告では、オープンソースの流体解析システムOpenFOAM[3]に対してスレッド並列化の為の疎行列の並び替え方法、CG法アルゴリズム、連立方程式解法以外の部分のスレッド並列化の方法を提案する。また、過度に部品化された機能を展開して纏めることでloopの計算密度をあげるとともにメモリーへのアクセスを減らして高速化している。作業領域の使用も減る為に、初期化の時間も節約できる。

更に，国産のオープンソースのFEM解析システムFrontISTR[4]に対しては、使用されているAMG法前処理CG法の高速化を行う。AMG法はTrilinosのML[5]を使用しており、そのコア部分は90年代初めに作成さたと思われ、演算性能が良いとは言えない。ソリッドモデル限定で新規にAMG法を適用して計算時間の短縮を行う。

2. OpenFOAMのスレッド並列化と高速化

2.1 手法

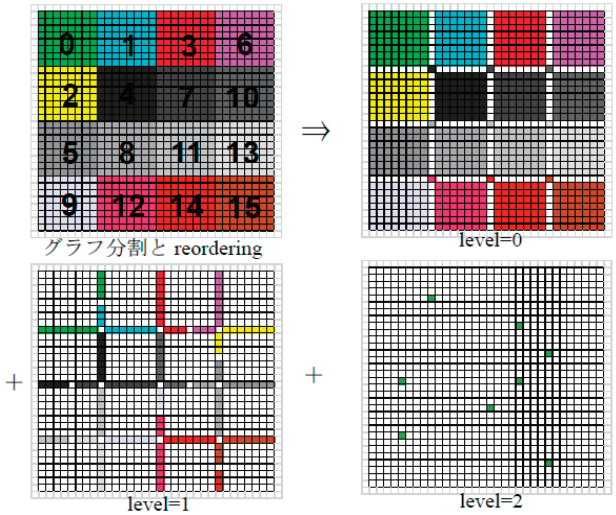
2.1.1 格子のオーダリングと行列格納方法

文献[6]ではFEMの剛性行列のアセンブルをスレッド並列化する為にMETIS[7]を使用して要素をグループ分けしている。こ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－1**に示す。**図－2**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。グループを単位としたreorderingはMETIS_NodeNDを用いるのが良い。**図－3**は各levelのグループ数を32, 4, 2とした

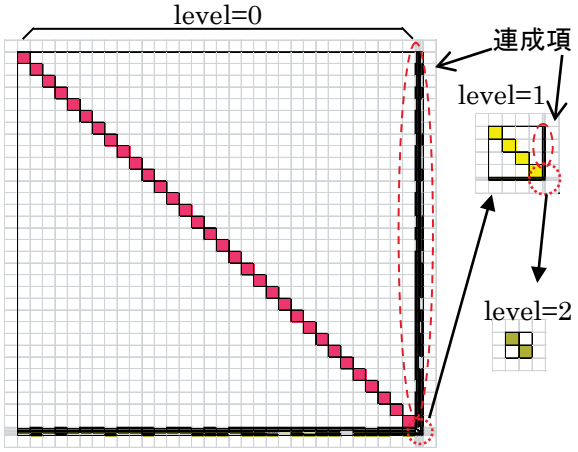
場合の係数行列のイメージ図である。非零項が存在するブロックが色付けされている。格子の 99%以上が level=0 と level=1 に含まれる。不完全 LU 分解を smoother に使用する場合は前進代入計算と後退代入計算が各 level で並列化できる。指定したグループ数に制御できる利点がある。

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多いなら、GOTO 1
7. END

図－1　格子のグループ分けのアルゴリズム [10)]

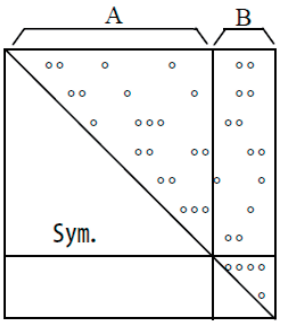


図－2　32×32 格子のグループ分け例 [10)]



図－3　係数行列の非零項の分布イメージ [10)]

更に、対角ブロック内の計算効率を高める為に図－4 の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。本報告では「2. reordering」は Sloan[8)]の方法を用いている。このようにすると、図－4 の A 部分は図－5 のように計算できる為、計算効率が良い。



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. reordering
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
5. 3 で最大値を与えた行を B の領域に移動
6. GOTO 3
7. END

図－4　対角ブロック内のオーダリング [10)]

```
for(int j=0; j<n1; j++) {
  int i0=iL[j][0], i1=iL[j][1], i2=iL[j][2];
  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];
}
```

(a) 前進代入計算

```
for(int i=n1-1; i>=0; i--) {
  int j0=iu[i][0], j1=iu[i][1], j2=iu[i][2];
  x[i] -= rD[i]*( U[i][0]*x[j0]
                 +U[i][1]*x[j1]
                 +U[i][2]*x[j2]);
}
```

(b) 後退代入計算

図－5 対角ブロック内 A 部分の計算(m1=3)[10)]

2.1.2 CG 法のアルゴリズム

図－6 (a)、(b)は夫々、通常版、Chronopoulos and Gear[9)]のアルゴリズムである。(b)は作業ベクトルの本数は多いが、通常の CG 法よりも全体通

信回数が少ない。また、前処理計算後、すぐに行列ベクトル積の計算が行える為、前進後退代入計算を行う前処理の場合は、後退代入計算を行いながら行列ベクトル積の計算も行うことで計算量を減らすことができる。尚、AMG 法を前処理とした場合に、(a)や(b)は計算が破綻する場合がある為、前 iteration 時と直交する方向ベクトルを生成することだけを考えた(c)を使用する。前処理は線形変換のはずであるが、それが乱れていると考える。

1: $\boldsymbol{r}_0 = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_0$;
2: for i=0, …. do
3: $\boldsymbol{u}_i = \mathbf{M}^{-1}\boldsymbol{r}_i$
4: $\gamma_i = (\boldsymbol{r}_i, \boldsymbol{u}_i)$
5: $\beta = \gamma_i / \gamma_{i-1}$
6: $\boldsymbol{p}_i = \boldsymbol{u}_i + \beta \boldsymbol{p}_{i-1}$
7: $\boldsymbol{s} = \mathbf{A}\boldsymbol{p}_i$
8: $\delta = (\boldsymbol{s}, \boldsymbol{p}_i)$
9: $\alpha = \gamma_i / \delta$
10: $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha \boldsymbol{p}_i$
11: $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \alpha \boldsymbol{s}$
12: Residual $= |\boldsymbol{r}_{i+1}|$
13: end for

(a)通常版

1: $\boldsymbol{r}_0 = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_0$;
2: for i=0, …. do
3: $\boldsymbol{u}_i = \mathbf{M}^{-1}\boldsymbol{r}_i$
4: $\boldsymbol{w}_i = \mathbf{A}\boldsymbol{u}_i$
5: $\gamma_i = (\boldsymbol{r}_i, \boldsymbol{u}_i)$
6: $\delta = (\boldsymbol{w}_i, \boldsymbol{u}_i)$
7: $\beta = \gamma_i / \gamma_{i-1}$
8: $\alpha_i = \gamma_i / (\delta - \beta\gamma_i / \alpha_{i-1})$
9: $\boldsymbol{p}_{i+1} = \boldsymbol{u}_i + \beta \boldsymbol{p}_i$
10: $\boldsymbol{s}_{i+1} = \boldsymbol{w}_i + \beta \boldsymbol{s}_i$
11: $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i \boldsymbol{p}_i$
12: $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \alpha_i \boldsymbol{s}_i$
13: Residual $= |\boldsymbol{r}_{i+1}|$
14: end for

(b) Chronopoulos and Gear[9)]

1: $\boldsymbol{r}_0 = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_0$;
2: for i=0, …. do
3: $\boldsymbol{u}_i = F_{\mathrm{AMG}}(\boldsymbol{r}_i)$
4: $\boldsymbol{w}_i = \mathbf{A}\boldsymbol{u}_i$
5: $\gamma = (\boldsymbol{r}_i, \boldsymbol{u}_i)$, $\eta = (\boldsymbol{w}_i, \boldsymbol{u}_i)$, $\zeta = (\boldsymbol{p}_{i-1}, \boldsymbol{w}_i)$
6: $\beta = -\zeta / \delta_{i-1}$ ($\beta = 0$ for i=0)
7: $\delta_i = \eta + \beta\zeta$
8: $\alpha_i = \gamma / \delta_i$
9: $\boldsymbol{p}_{i+1} = \boldsymbol{u}_i + \beta \boldsymbol{p}_i$
10: $\boldsymbol{s}_{i+1} = \boldsymbol{w}_i + \beta \boldsymbol{s}_i$
11: $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i \boldsymbol{p}_i$
12: $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \alpha_i \boldsymbol{s}_i$
13: Residual $= |\boldsymbol{r}_{i+1}|$
14: end for

(c)本報告のアルゴリズム

図－6　CG 法のアルゴリズム [10)]

2.1.3 運動方程式の連立方程式解法

DILU（対角項のみ更新した不完全 LU 分解)を前処理とした BiCG 法と Additive Schwartz を用いる。少ない収斂計算回数で収斂する安定な問題の場合は，Additive Schwartz が計算時間の点で有利である。係数行列は流速 3 成分で対角項以外は同一である為、3 成分同時に計算することで計算効率を向上させる。

2.1.4 vector，tensor の展開

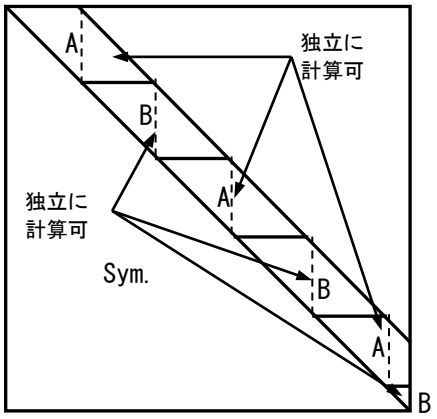
OpenFOAM 内では、3 成分の vector 型と 9 成分の tensor 型があり、それらに対して演算子が定義されている。C++で書かれている利点を損なうことになるが、これらの型は夫々、[*][3]、[*][3][3]の型に cast して使用し，更に、ループ内のテンポラリーの配列は成分ごとのスカラー変数に置き換えると高速化される。

2.1.5 良く現れるループの並列化（バンド幅狭い）

図－7 の形のループが良く現れる。変数 nFaces は face の数であり、行列イメージでは上三角、又は下三角部分の非零項の数であり、対角項は含まない、非零項の配置は対称である。**図－7** は i 行と j 行に対する演算であるが、上三角部分を考え、i 行と j 列に対する演算と解釈して並列化を考える。**図－8** のようにブロック化すると A 部分と B 部分の block multicolor となり，スレッド並列可能となる。ブロックサイズは可変とし、最小サイズを設定してブロック分けを行う。

```
for(label fi=0; fi<nFaces; fi++) {
  label i=owner[fi],
j=neighbor[fi];
  scalar tmp = (some computation);
  a[i] += tmp;
  a[j] += tmp;
}
```

図－7　良く現れる loop 形 [11)]

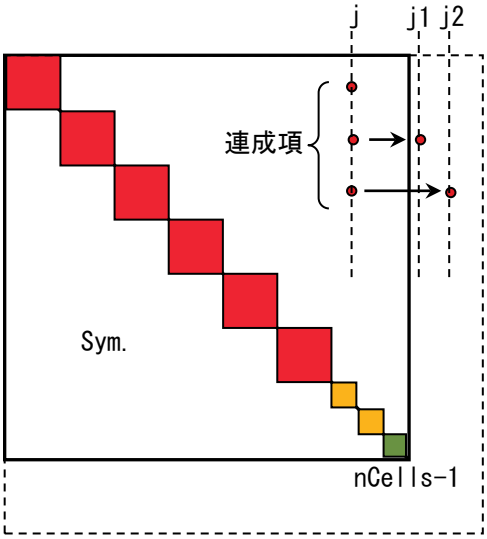


図－8　2 色の block multicolor[11)]

2.1.6 良く現れるループの並列化（バンド幅広い）

バンド幅が広い場合は前項の方法だと十分なブロック数が得られない。この場合は、**2.1.1 項**と同様のリオーダリングを行う。尚，**図－1** の「グループを単位として reordering」の reordering では CM(Cuthill Mckee) を適用し、対角ブロック

内もCMを適用しておくとキャッシュメモリーの利用効率が上がる。連立方程式解法では，処理に入る前に作業領域に対角項，対角ブロック，連成項を別に格納しているが，この場合は row compressed で格納されている為、工夫を要する。**図－9** に示すように，各 level でグループ間の連成項に二回以上現れる列番号がある場合は、二回目からは nCells(格子数)以上の重複しない列番号をつける。新たなリスト配列と作業領域を必要とし、nCells 以上の番号に格納された結果を元の番号の場所に格納し直す処理が必要になる為、バンド幅が狭い場合に適用すると前項の方法に比べて計算が遅くなる。

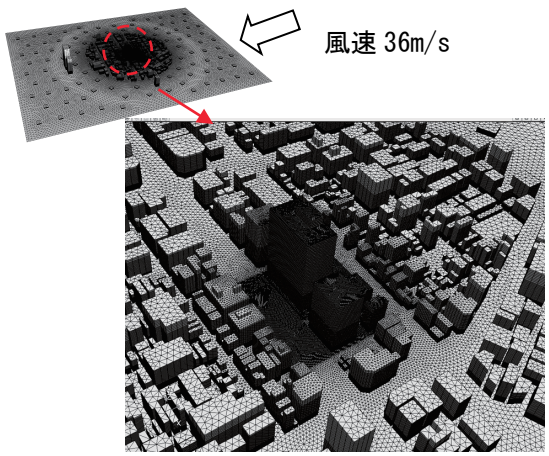


図－9 連成項の列番号変更 [11)]

2.2 計算例

2.2.1 解析モデル [10, 11)]

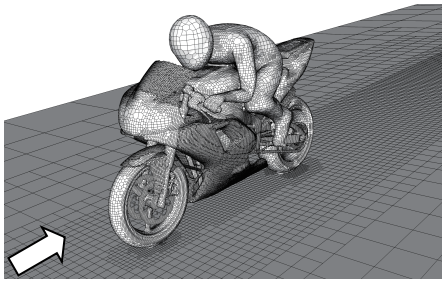
計算に用いるのは、**図－10** と**表－1**、**図－11** と**表－2** に示したモデルである。**図－10** のモデルは比較的綺麗な格子で、係数行列バンド幅が狭い。**図－11** のモデルは複雑な格子で、バンド幅が広い。



図－10 高層建物 [10,11)]

表－1 高層建物の計算領域と格子数 [10,11)]

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



風速 20m/s

図－11 motorBike[10,11)]

表－2 motorBike の計算領域と格子数 [10,11)]

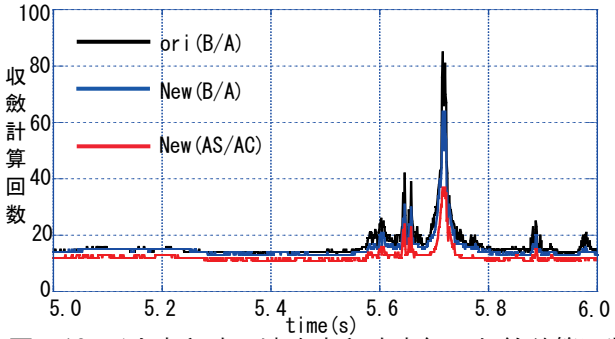
計算領域 (m)	20×8×8	
格子数	4,601,581	
格子種類と数	hexahedra	3,897,137
	prisms	107,368
	wedges	19,160
	pyramids	535
	tet wedges	21,499
	tetrahedral	444
	polyhedra	555,438

2.2.2 計算例 1 (Intel Xeon) [10)]

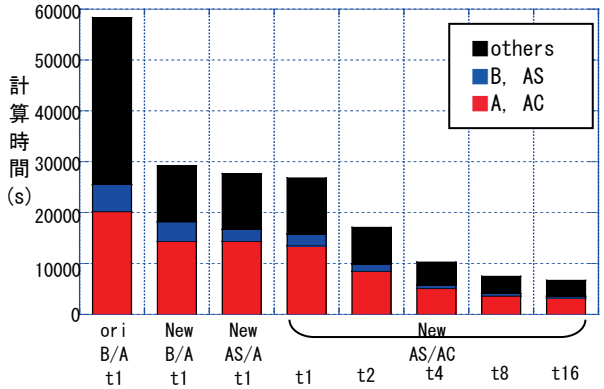
連立方程式解法は、運動方程式が B(DILUBiCG)と AS(Additive Schwartz)、圧力方程式は A(AMG)と AC(AMG-CG)を適用する。圧力方程式の修正回数は 3 回とした。原コード(ori)は Cuthill-Mckee ordering 適用の B/A、書換えた新コード(New)は本報告で提案の ordering 適用の B/A、AS/A、AS/AC である。AMG 法は mergeLevels=2、各レベルの sweep 回数は 1、smoother は DIC(対角項のみ書換える不完全分解)を用いた Additive Schwartz である。Coarsest Grid の格子数と解法は、ori は 1,990 格子で DICCG，New は 16,568 格子で直接解法の multifrontal 法である。計算環境は、Intel Xeon E5-2687w v4 (3.0GHz, 12 cores, 30 MB Cache)を 2 基搭載した計算機を使用し、OS は CentOS-7.3、コンパイラは Intel C/C++ Compiler v16.0.4.258 である。

図－10 の高層建物の結果を示す。時間増分幅は 0.001 s である。**図－12** から A(AMG 法)よりも AC(AMG-CG 法)が安定であるのが分かる。計算時間を比較した**図－13** から、連立方程式求解部分は ori で 44%であり、連立方程式求解以外の高速化も重要であることが分かる。New は並列化しない状態

で、ori の 0.45-0.50 倍の計算時間となっており、書換えの効果が表れている。スレッド並列化による高速化率は、New(AS/AC)は 8 スレッドで 3.6 倍、16 スレッドで 4.0 倍であり、スレッド数が増えても高速化しにくいのが分かる。これは、NUMA であることによって、メモリー帯域が早くに飽和してくる為と考える。



図－12 圧力方程式の連立方程式求解の収束計算回数 [10)]



図－13 計算時間(5 s - 6 s, t*: thread 数)[10)]

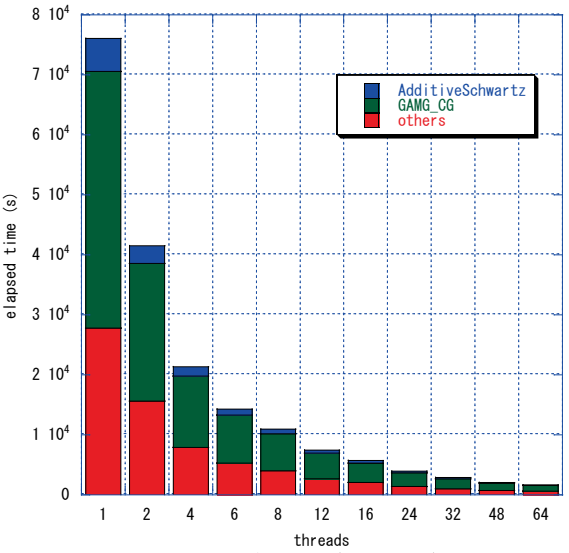
2.2.3 計算例２(Intel Xeon-Phi)[11)]

前項のプログラムから、更に手を加え、ほぼ全ての箇所をスレッド並列化している。連立方程式解法は流速に関する方程式が Additive Schwartz、圧力方程式は AMG-CG である。圧力方程式の修正回数は 3 回である。AMG 法は各レベルの sweep 回数は 1、smoother は DIC(対角項のみ書換える不完全分解)を用いた Additive Schwartz である。計算環境は、Oakforest-PACS(Knights Landing: Xeon-phi 7250 (1.4GHz, 68 cores/node))の 1 node を使用し、MCDRAM を cache memory として使う cache mode で計算を行う。コンパイラは Intel C/C++ Compiler v17.0.4.196 である。

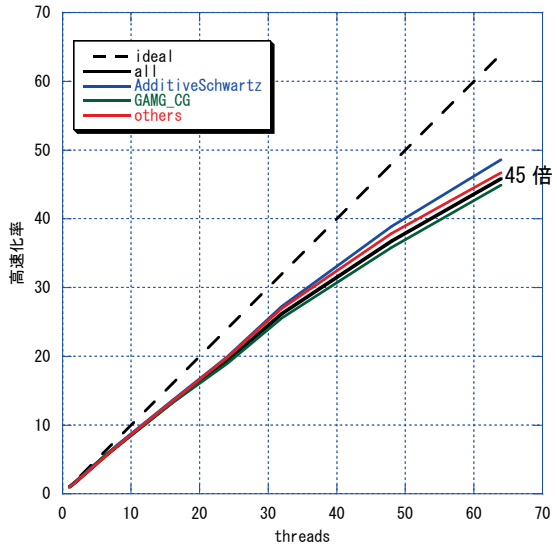
(1) 高層建物（図－10）

時間増分幅は 0.001 s、計測時間は 5 s から 6 s である。AMG 法の各レベルで, 2.1.1 における level=0 のグループ数は 128、coarsest grid 数は 1175 格子(skyline 法の直接解法)である。

図－14 に計算時間を、図－15 に高速化率示す。性能の上がり方が鈍ってはくるが、メモリー性能が高いことから、かなり良い性能となっている。



図－14 計算時間(5 s - 6 s)[11)]



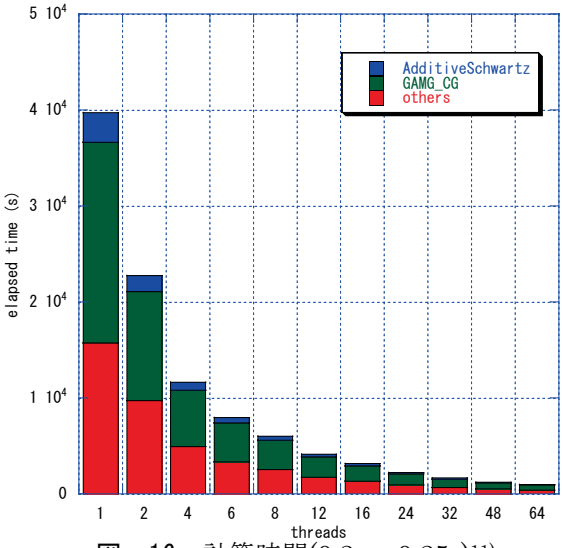
図－15 高速化率 [11)]

(2) motorbike（図－11）

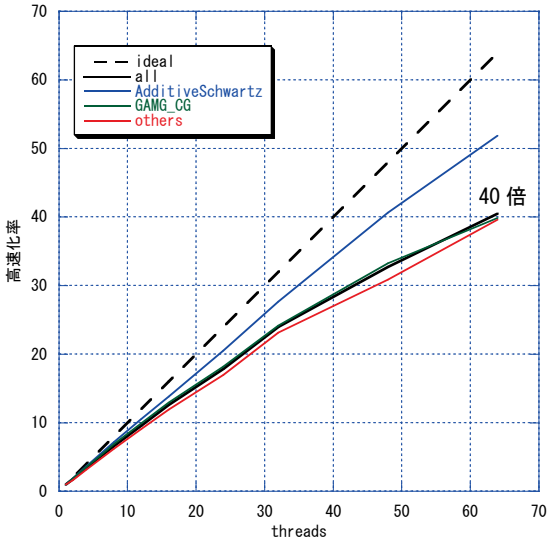
時間増分幅は 0.00005 s である。AMG 法の各レベルで, 2.1.1 における level=0 のグループ数は 128、coarsest grid 数は 969 格子(skyline 法の直接解法)である。連立方程式解法以外のループに関しては、バンド幅が広い場合に該当し、グループ数は level = 0,1 で 256 とした。

図－16 に計算時間を、図－17 に高速化率示す。高層建物の場合と同様の傾向であるが、やや性能が悪い。バンド幅が広いことで付加された処理があるこ

とと、一ブロック当たりの格子数が少ないことが原因と考える。



図－16　計算時間(0.2 s - 0.25s)[11)]



図－17　高速化率[11)]

3. FrontISTR の AMG 法前処理付き CG 法の高速化

3.1 手法

3.1.1 行列の持ち方

ソリッドモデルに限定すれば、各節点 3 自由度で near null vectors が 6 本であり、3×3、3×6、6×6 ブロックを単位としたブロック疎行列とすれば非零項へのアクセスを効率化できる。行列は流体解析システム OpenFOAM の方式をブロック化したものとし、上三角(CRS)、対角項、下三角(CCS、上三角と対称)、Halo 領域を分けて持つ。

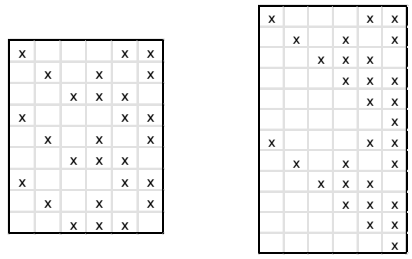
3.1.2 AMG 法：Coarsening

構造問題で考えれば剛体結合していくことと同値である。行列は対称とする。

図－18 の 4、5 では、図－19 のように 6 個の剛体変位モードの非零成分の配置は既知であることから、非零成分についてだけ計算を行えば良い。QR 分解はグラム・シュミットの正規直交化法で十分であり、プログラミングも簡単である。但し，剛体結合する節点が直線上に並んでいる場合はハウスホルダー変換を使用する。また、5、6 の計算は非零ブロックの発生位置の情報を作ってから計算すると速い。対称な場合は $\mathbf{A}_k$ は対角ブロックと上三角部分を作成すれば良い。

k=0, $\mathbf{A}_0 = \mathbf{A}$ (係数行列), $\mathbf{V}_0 = \mathbf{V}$ (null vectors)
1. k=k+1
2. $\mathbf{A}_{k-1}$ の非零項の配置から節点グループを作成
3. 剛体結合の変換行列 $\tilde{\mathbf{P}}$ を $\mathbf{V}_{k-1}$ から作成
4. QR 分解： $\tilde{\mathbf{P}} = \mathbf{QR}$, $\mathbf{V}_k = \mathbf{R}$
5. Smoothing： $\mathbf{P}_k = (\mathbf{I} - \omega \mathbf{D}_{k-1}{}^{-1} \mathbf{A}_{k-1})\mathbf{Q}$
6. $\mathbf{A}_k = \mathbf{P}_k{}^{\mathrm{T}} \mathbf{A}_{k-1} \mathbf{P}_k$　（対称な場合）
7. if(節点数が設定値よりも大きい) goto 1
8. end

図－18 Coarsening[13)]

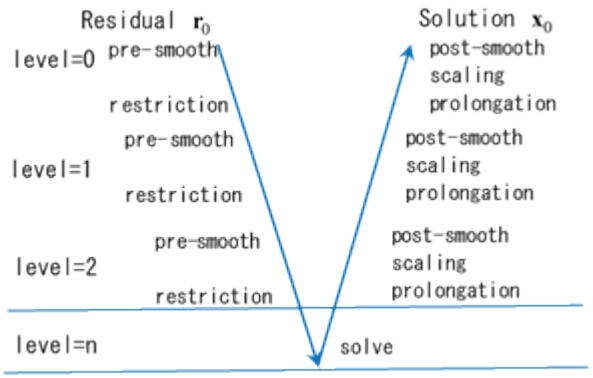


図－19 Null Vectors[13)]

Fine mesh を CM(Cuthill-Mckee) ordering することで cache memory の利用効率が上がるが、先頭の節点から aggregate すると coarse mesh の節点数が多くなる。そこで、接続節点数が多い節点から aggregate する方法を新たに作成した。

3.1.3 AMG 法：V-cycle, W-cycle

図－20 は V-cycle で計算する流れである。本報告では、Fine mesh での計算回数を増加させずに収斂性を向上させる為に level 1 以上をもう一度繰り返す W-cycle を行った。Scaling の処理は OpenFOAM の AMG 法に組込まれている傾斜法による解の改良を参考にしたものである[12)]。但し、今回の計算例では、その機能を使用せずに安定に計算できた。



Restriction: $\mathbf{r}_k = \mathbf{P}_k{}^T \mathbf{r}_{k-1}$, Prolongation: $\mathbf{x}_{k-1} = \mathbf{P}_k \mathbf{x}_k$
k=level

図－20 V-cycle[13)]

3.1.4 AMG法：Smoother

FrontISTRではSmootherにChebyshev反復を使用しているが収斂状況が良くない場合がある為、図－21のSymmetric Gauss-Seidel(SGS)を用いる。ここで，演算量の削減を考えると、Pre-smoothでは$\mathbf{x}^{(0)} = 0$であることから最初のsweepの前進代入計算では$\mathbf{U}\mathbf{x}^{(0)}$が不要である。後退代入計算では$\mathbf{U}\mathbf{x}^{(k+1)}$の計算が発生し、これを作業領域に保存して次の前進代入計算で使用すれば良い。最後のsweepでは後退代入計算と行列ベクトル積、又は残差計算を同時に行えば良い。Post-smoothでは，scalingの計算時に残差を計算しておけば、SOR法前処理(緩和係数=1.0)を用いたAdditive Schwartz(SGSと等価)とした方が計算しやすい。Pre-smoothと同様にSOR法の後退代入計算と行列ベクトル積を同時に行えば効率的である。図－6(c)のCG法のアルゴリズムを用いれば，前処理計算の直後に行列ベクトル積を計算できる為、level=0ではそのまま行列ベクトル積の結果を使用できる。尚、level>0では最後のsweepで行列ベクトル積は不要である。

for $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$, $\mathbf{x}^{(0)}$, $\mathbf{b}$ are given.
for k=1,2,…. do
Forward: solve $(\mathbf{L} + \mathbf{D})\mathbf{x}^{(k+1/2)} = \mathbf{b} - \mathbf{U}\mathbf{x}^{(k)}$ for $\mathbf{x}^{(k+1/2)}$
Backward: solve $(\mathbf{D} + \mathbf{U})\mathbf{x}^{(k+1)} = \mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1/2)}$ for $\mathbf{x}^{(k+1)}$
end for

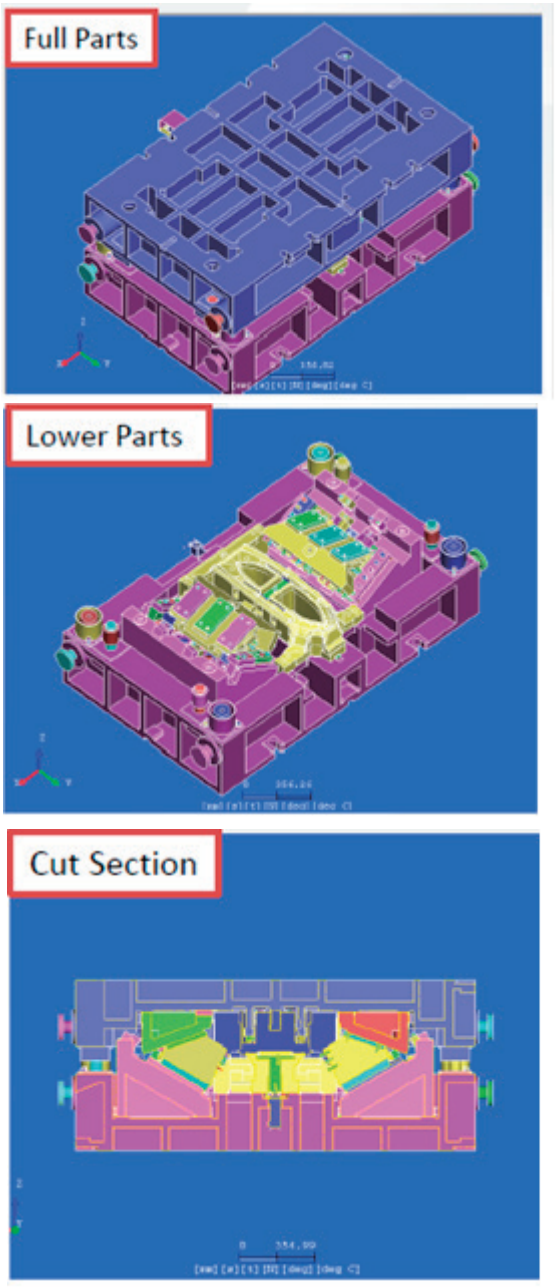
図－21 Symmetric Gauss-Seidel[13)]

Coarsest meshでの計算は単一領域の場合は直接解法、領域分割法の場合は各領域で係数行列を三角分解し、それを前処理として使用するCG法とした。この箇所のCG法は図－6(b)を用いた。

3.2 計算例[13)]

3.2.1 解析モデル

使用するデータは，図－22に示す複雑な形状をした1,015,778節点、638,041要素のMold_001milである[14)]。材料は一様である。4面体2次要素で離散化され、下面並進3成分が固定、上面に圧力が作用する。領域分割はKMETISで行い、AMG法のcoarseningはflat aggregation、uncoupled(隣接領域間でaggregationを行わない)とした。



図－22 Mold_001mil[13)]

使用した計算機のCPUはXeon Gold 6150 @2.70GHz (18 cores)×2，メモリーはDDR4-2666

の 6ch，OS は RedHat7.4、コンパイラは Intel Compiler v19.0.4.243 である。

表－3 に計算ケースを示した。FrontISTR v4.5 の設定で ML を使用したケースが Ori である。単一領域の場合の level=2,3 での行列サイズは夫々、3210 と 102 である。現在の計算機であれば 3210 のサイズの行列を skyline 法で三角分解するのに時間はかからない為、Level=3 まで行う必要は無いと考える。Lx における三角分解は，行列サイズを n、非零項数を nz として、$nz/n^2 > 0.5$ であれば密行列として計算し、それ以外は文献 [8]の方法で reordering して 6-by-6 block skyline 法 [15]で計算した。全体方程式の CG 法の収束判定値は 1.0e-6 とした。

表－3 計算ケース [13]

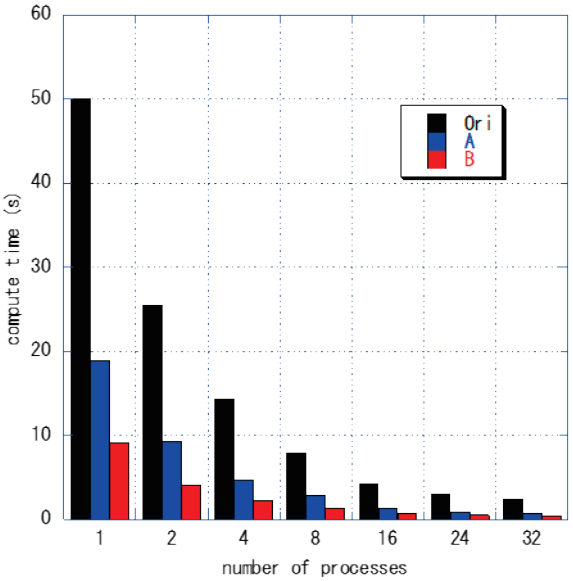
case		Ori	A	B	B2
ordering		-	-	CM	CM
agg_new		-	-	○	○
cycle		V	W	W	W
0	pre/post	Ch2	SGS1	SGS1	SGS1
1	pre/post	Ch2	SGS1	SGS1	SGS2
2	pre/post	Ch2	Lx	Lx	Lx
3	pre/post	Ch2	-	-	-

Ori: ML 使用（FrontISTR v4.5 の設定を使用）
A, B, B2: 新規作成
ordering: fine mesh を CM ordering
agg_new: 接続節点が多い節点から aggregate する
cycle: V-cycle, W-cycle (0-1-2-1-2-1-0)
ChN: Chebyshev 反復を N回
SGSN: Symmetric Gauss-Seidel を N回 sweep
Lx: 単一領域の場合は直接法（6-by-6 block 分解）．領域分割の場合は三角分解した行列を前処理とする CG 法（収束判定値は 1.0e-4）

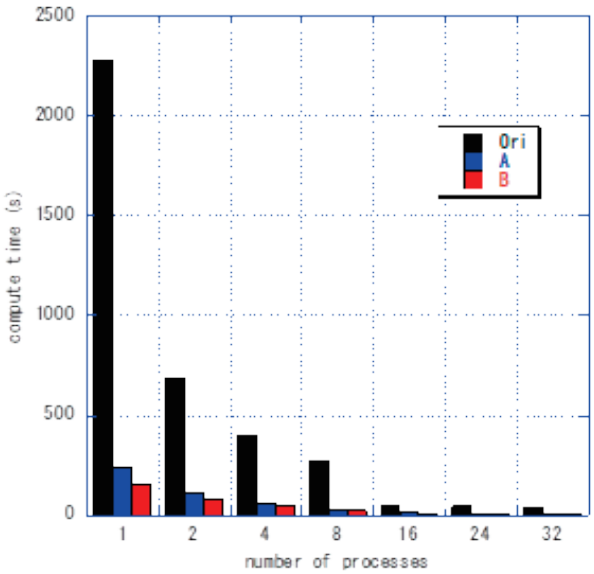
3.2.2 計算結果

図－23 は AMG 法の Setup (coarsening)の計算時間である。Aggregate の方法が同じである Ori と A では、新規に構築した A が 2.6 から 3.4 倍高速化している。B は A から概ね 2 倍高速化しており、Ori からは 5 倍以上の高速化である。B は Ori よりも coarse mesh の節点数がやや少なくなっている面もあるが、cache memory が効くようになった為と考える。

図－24 は Solve (CG 法による全体方程式求解)の計算時間である。尚、B の 24 process と 32 process は B2 である。計算効率が良くなったことと、各レベルの smoother を性質の良いものに変えたことで大幅に計算時間が短縮され、B2 は Ori から、32 process 時で 4.9 倍高速化された。



図－23 Setup の計算時間 [13]



図－24 Solve の計算時間 [13]

収斂計算回数は、1 process 時の Ori、A、B は夫々、908、143、142 であり、32 process 時の Ori、A、B2 は夫々、432、195、153 である。

4. おわりに

流体解析システム OpenFOAM に関しては、連立方程式解法のスレッド並列化の為のオーダリング方法を提案するとともに、他の部分の高速化とスレッド並列化の方法を示した。計算機の仕組みとメモ

リー性能から、Intel Xeon 上ではスレッド数を増やしても性能向上はすぐに鈍ってくるが、メモリー性能の高い Intel Xeon-phi では高い性能が得られた。また、限定した機能についてチューニングした結果ではあるが、原コードからの計算時間短縮は十分な成果と考える。

FEM 解析システムの FrontISTR については、AMG 法を新規に構築し、AMG 法を前処理とする CG 法を大幅に高速化した。行列の持ち方と計算方法の変更、改良に加え、aggregation 方法の工夫と AMG 法用に smoother をチューニングしたことで効果を上げた。

謝辞

Mold001mill のデータとモデル図の使用を許可して頂いた FrontISTR Commons に感謝いたします。

＜参考文献＞

1) T. Iwashita, M. Shimasaki : “Algebraic multicolor ordering for parallelized ICCG solver in finite-element analyses”, IEEE transactions on Magnetics, vol.38, No.2, pp.429-432, 2002
2) 内山学, ファム バン フック : “非構造格子に対応した PCG 法の thread 並列化手法”, 第 153 回ハイパフォーマンスコンピューティング報告発表会, 2016
3) https://www.openfoam.com/
4) https://github.com/FrontISTR
5) https://trilinos.github.io/ml.html
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), pp.395-396, 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, pp.2651-2679, 1989
9) P. Ghysels, W. Vanroose “Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm”, Parallel Computing vol.40, pp.224-228, 2014
10) 内山学, Pham Van Phuc : “非構造格子に対応した thread 並列化手法”, 計算工学講演会論文集, vol.22, 2017
11) 内山学, Pham Van Phuc : “Xeon-phi 上での非構造格子の thread 並列化”, 計算工学講演会論文集, vol.23, 2018
12) 内山学 : “FrontISTR における AMG 前処理の改良”,日本建築学会学術講演梗概集(関東), pp.193-194, 2020
13) 内山学 : “FrontISTR における AMG 法前処理の改良による CG 法の高速化”, 計算工学講演会論文集, vol.26, 2021
14) 稲垣和久, 奥田洋司:“ベンチマーク問題 FBench について(第２報)”, 第 61 回 FrontISTR 研究会, 2020
15) 内山学 : “大規模構造解析用高速 solver の検討”, 構造工学における数値解析法シンポジウム論文集, vol.18, pp.205-210, 1994