Vol.2015-HPC-151 No.20
2015/10/1

OpenFOAMによる流体コードのHybrid並列化の評価

内山学[†1]　ファム　バン　フック[†1]　千葉修一[†2]　井上義昭[†3]　浅見暁[†3]

本報告は流体コードOpenFOAMを基にして，MPI並列とThread並列を用いたHybrid並列の検討を行う．OpenFOAMはThread並列には対応していないため，CG法とBiCG法を対象にThread並列化を可能とする行列のオーダリング方法を示すとともに，計算効率を向上させる行列の格納方法を示す．更に，全体通信回数の少ないアルゴリズムを採用し，そのアルゴリズムの特徴を生かして行列演算の効率化を行う．CG法とBiCG法以外の部分に対してもThread並列化の方法を示し，最後に，Hybrid並列コードとMPI並列コード，元コードを「京」コンピュータ上で比較する．

Evaluation of Hybrid Parallelization of a CFD Code based on OpenFOAM

MANABU UCHIYAMA[†1]　PHAM VAN PHUC[†1]
SHUICHI CHIBA[†2]　YOSHIAKI INOUE[†3]　AKIRA AZAMI[†3]

This paper describes hybrid parallelization of OpenFOAM which is an open source CFD code. To parallelize CG and BiCG method in OpenFOAM by using threads, cell numbers are reordered and matrices are stored in special forms. Algorithms that have less synchronization points than those of the ordinary algorithms are used and ILU(0) preconditioning and matrix-by-vector multiplication are reconstructed and improved. Parallelizing other parts is also described. The hybrid parallelized code, the new code but flat MPI and the original code are run on K computer and their performance is shown.

1. はじめに

HPC分野でもC++を利用したシミュレーションコードの開発が増加している．目的の一つとして，シミュレーションコードの生産性を確保することが挙げられる．特に流体解析の分野では，オープンソースの流体解析ライブラリであるOpenFOAM (Open source Field Operation And Manipulation)の利用が拡大している．

OpenFOAMは，乱流，燃焼，混相流等の様々なモデルが用意されており，要件に合わせたソルバ，クラスライブラリを利用することができる．これにより，研究者は高度な計算科学の技術を必要とせずに，シミュレーションコードの生産性を確保しながら高い実行性能を享受できる．

高い実行性能を実現するため，OpenFOAMを構成するコンテナクラスでは，基底クラスにExpression Template技術を利用している．この技術により，コンパイラの最適化に頼ることなく一時的なメモリー利用を省略し，ソルバ演算の演算密度を向上させている．

しかしながら，OpenFOAMには幾つかの課題がある．

一つはコンパイラの最適化などの適用が困難となる仮想関数の利用である．これは，複数のモデルを実現するためにStrategyデザインパターンが採用されているためである．このデザインパターンで必要となるポリモーフィズムを実現するため，C++では仮想関数の利用が必要不可欠である．仮想関数は，コンパイラによるインライン展開最適化が難しく，演算密度の向上の妨げになっている．そのため，ハンドチューニングによる性能向上が必要となる．

もう一つはコードの並列性である．通常，ソルバをThread並列化する場合，コンパイラの自動並列化を期待するか，ハンドチューニングによる最適化指示を行うことになる．特に，自動並列化できないコードに対してはハンドチューニングが必要不可欠である．しかしながら，OpenFOAMは多階層のテンプレートで実現されており，Thread並列化を指示が可能なループ形状となっていない．

本稿では，これらの課題に対する取り組みを述べ，その評価結果を報告する．

多くの機能を持つOpenFOAMライブラリから，pisoFOAMを基に各部の計算の効率化とともにノード内の計算をThread並列化してHybrid並列の性能を検討する．CG法とBiCG法を対象に，Thread並列を行うための格子のオーダリングと行列成分の格納方法を示すとともに，全体通信回数の少ないアルゴリズムを採用し，そのアルゴリズムの特徴を生かして，ILU(0)前処理と行列ベクトル積の演算の効率化を図る．運動方程式を解くBiCG法に関しては，作業領域は増加するが，流速3成分を同時に計算することで効率化を試みている．ただし，これら3成分の連成までは考慮しない．CG法とBiCG法以外の部分のThread並列化の方法についても示す．Thread並列化ツールはOpenMPを使用する．なお，本報告では直交格子に限定している．また，演算性能の検討が目的であり，CG法とBiCG法の収斂性については論じない．使用する計算機は「京」コンピュータである．

†1 清水建設株式会社
SHIMIZU Corporation
†2 富士通株式会社
Fujitsu Ltd.
†3 一般財団法人高度情報科学技術研究機構
Research Organization for Information Science and Technology

Vol.2015-HPC-151 No.20
2015/10/1

2. 格子のオーダリングと行列格納方法

2.1 Block Multicolor

格子をマルチカラーオーダリングして並列化する方法は種々提案[1, 2, 3]されているが，格子ごとにオーダリングを行うと近接する格子が離れて並ぶことになる．そのため，ILU(0)の前処理行列の性質が悪化して収斂性が著しく悪くなる．そこで，本報告では図 1 に示すように，ノードに割当てられた格子を 4x4x4 のブロックに分割し，そのブロックを単位として，図 2 のように 2 色のマルチカラーにオーダリングする．各ブロック内は Cuthill-McKee オーダリングを行う．4x4x4 の分割で 2 色の場合，独立に計算できるブロックは 32 個である．OpenMP3.0 からは work pile 方式であり，schedule を dynamic に指定すれば多少の imbalance は吸収できる．「京」コンピュータは 8cores/node であるから，この分割で十分な並列度を確保可能と考える．

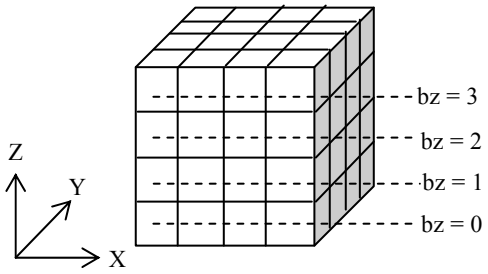


Figure 1　計算領域を 4x4x4 のブロックに分割

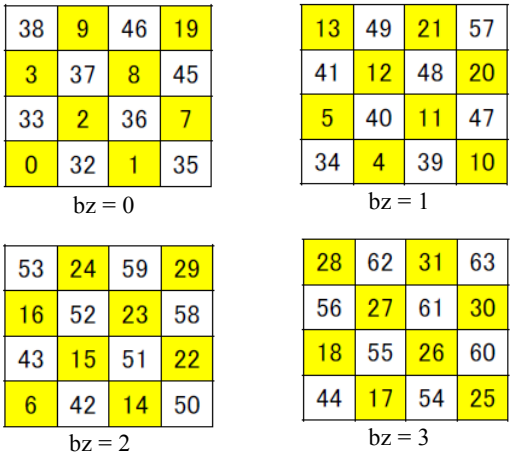


Figure 2　Multicolor (2 色)

2.2 行列格納方法

図 3 は図 2 の各ブロックを一単位として行列の非零ブロックを図示したものである．対角ブロック内の非零項は最大 3 個/行である．ブロック内の格子数を ni として，対角ブロックの行列を下三角行列 L[ni][3]，上三角部分 U[ni][3] とし，対応する列番号を夫々，jl[ni][3]，ju[ni][3]のように 2 次元配列として格納する．プログラムは図 4 のように 行番号でループを回し，列方向 3 成分は陽に展開する．

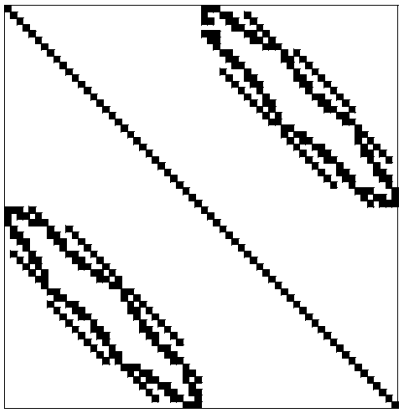


Figure 3　非零ブロックの分布

```
void
DICP_x2_f( double *rD    , double *L , double *y, double *x,
           Int     nCells, int    *jL )
{
    double (*__restrict__ LL)[3] = (double (*)[3]) L;
    int    (*__restrict__ jl)[3] = (int    (*)[3]) jL;

    /*--- Forward Substitution ---------------------------*/
    for (register int i=0; i<nCells; i++) {
      idata j0=jl[i][0], j1=jl[i][1], j2=jl[i][2];

      x[i] = rD[i]*(y[i]-LL[i][0]*x[j0]
                        -LL[i][1]*x[j1]
                        -LL[i][2]*x[j2]);
    }
}
```

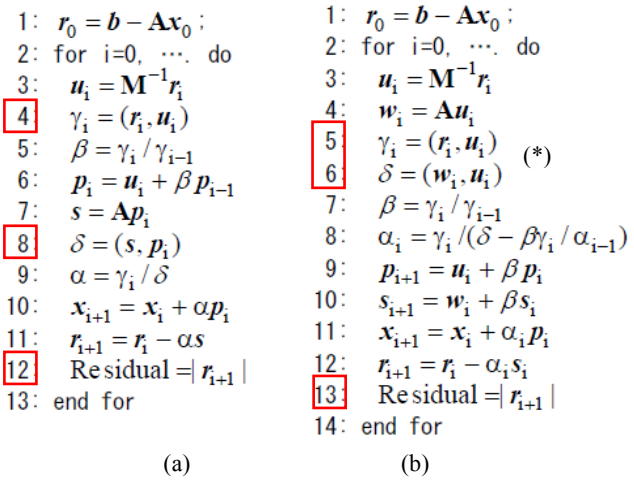
Figure 4　プログラム例（ILU(0)前処理の前進代入計算）

3. CG 法と BiCG 法

計算時間の大部分を占めるのは連立方程式を解く部分である．CG 法と BiCG 法には多くの変種が提案[4, 5, 6]されており，夫々特徴がある．本報告では，計算量の増加が少ない文献[4]の Chronopoulos and Gear の方法を試みる．

3.1 アルゴリズム

CG 法と BiCG 法の通常のアルゴリズムは図 5(a)と図 6(a)の通りである．一反復当たり全体通信が 3 回必要である．



(a)　(b)

Figure 5　Preconditioned CG（□：全体通信）

1: $r_0 = b - Ax_0$; $\bar{r}_0 = b - A^T x_0$
2: for i=0, …. do
3: $u_i = M^{-1} r_i$; $\bar{u}_i = M^{-T} \bar{r}_i$
4: $\gamma_i = (\bar{r}_i, u_i)$
5: $\beta = \gamma_i / \gamma_{i-1}$
6: $p_i = u_i + \beta p_{i-1}$; $\bar{p}_{i+1} = \bar{u}_i + \beta \bar{p}_i$
7: $s = Ap_i$; $\bar{s} = A^T \bar{p}_i$
8: $\delta = (s, \bar{p}_i)$
9: $\alpha = \gamma_i / \delta$
10: $x_{i+1} = x_i + \alpha p_i$; $\bar{x}_{i+1} = \bar{x}_i + \alpha_i \bar{p}_i$
11: $r_{i+1} = r_i - \alpha s$; $\bar{r}_{i+1} = \bar{r}_i - \alpha_i \bar{s}_i$
12: Residual = $| r_{i+1} |$
13: end for

(a)

1: $r_0 = b - Ax_0$; $\bar{r}_0 = b - A^T x_0$
2: for i=0, …. do
3: $u_i = M^{-1} r_i$; $\bar{u}_i = M^{-T} \bar{r}_i$
4: $w_i = Au_i$; $\bar{w}_i = A^T \bar{u}_i$
5: $\gamma_i = (\bar{r}_i, u_i)$
6: $\delta_i = (w_i, \bar{u}_i)$ (*)
7: $\beta = \gamma_i / \gamma_{i-1}$
8: $\alpha_i = \gamma_i / [\delta_i + \beta^2 \delta_{i-1} - (2\beta / \alpha_{i-1}) \gamma_i]$
9: $p_{i+1} = u_i + \beta p_i$; $\bar{p}_{i+1} = \bar{u}_i + \beta \bar{p}_i$
10: $s_{i+1} = w_i + \beta s_i$; $\bar{s}_{i+1} = \bar{w}_i + \beta \bar{s}_i$
11: $x_{i+1} = x_i + \alpha_i p_i$; $\bar{x}_{i+1} = \bar{x}_i + \alpha_i \bar{p}_i$
12: $r_{i+1} = r_i - \alpha_i s_i$; $\bar{r}_{i+1} = \bar{r}_i - \alpha_i \bar{s}_i$
13: Residual = $| r_{i+1} |$
14: end for

(b)

Figure 6　Preconditioned BiCG（□：全体通信）

図 5(b)は文献[4]による CG 法のアルゴリズムである．一反復当たり全体通信は 2 回で良い．図 6(b)は BiCG 法について同様に導いたアルゴルズムである．更に，残差ノルムに関する全体通信を図 5(b)と図 6(b)の(*)の位置に移動すれば，全体通信は一反復当たり一回で済む．但し，前処理と行列ベクトル積の計算は一回分無駄になる．

3.2 行列演算

図 5(b)と図 6(b)のアルゴリズムでは，前処理計算後，直ちに行列ベクトル積を計算できる．ILU(0)前処理使用時は，前処理の後退代入計算を行いながら行列ベクトル積の計算を行うことで，行列成分をメモリーから読込む回数を減らし，且つ演算密度を上げることができる．図 7 は CG 法の例である．

```
for (register int i=nCells-1; i>=0; i--) {
  int j0=ju[i][0], j1=ju[i][1], j2=ju[i][2];
  double s1  = U1[i][0]*x1[j0]+U1[i][1]*x1[j1]+U1[i][2]*x1[j2];
  x[i]      -= rD1[i]*s1;          ← 後退代入
  z[i]       = D1[i]   *x1[i] +s1;
  z[j0]     += U1[i][0]*x1[i];
  z[j1]     += U1[i][1]*x1[i];     行列ベクトル積
  z[j2]     += U1[i][2]*x1[i];
}
```

Figure 7 後退代入計算と行列ベクトル積の例

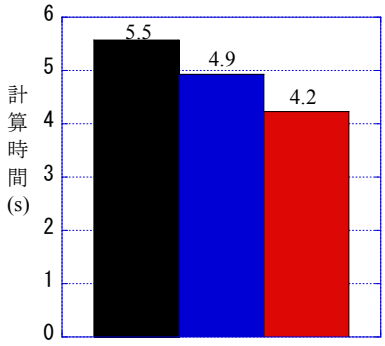
3.3 内積演算

図 5 と図 6 のアルゴリズムでは，δ は前処理計算後に計算される．ILU 前処理を使用する場合は式(1)のように表すことが出来るため，前進代入計算と同時に計算を行うことができる．

$$\delta = (\bar{r}, u) = (Uu)^T (L^T \bar{u}) \quad (1)$$

3.4 運動方程式の流速 3 成分を同時に計算（BiCG 法）

運動方程式は各流速成分を独立に計算している．各成分で，係数行列の非対角項は同じであるため，3 成分を同時に計算することを考える．(A) 夫々の成分を格子数長のベクトルとして扱う場合と(B) 3 成分を v[*][3]の 2 次元配列として扱う場合の二つの方法を検討する．図 8 は 172x172x172 格子を一領域としたモデルでの時間増分 10step（BiCG 法の総反復回数=30 回，8 threads 使用）の BiCG 法の計算時間の比較である．B の計算時間は，成分ごとに計算する場合に対しては 24%，A に対しては 14%短縮されている．OpenFOAM 内では流速に関する配列は B の形式で確保されているため，成分ごとの計算や A で必要となる並べ替え部分が B では不要になるメリットもある．



■：成分ごとに計算
■：(A) 夫々の成分を格子数長のベクトルとして同時計算
■：(B) 3 成分を v[*][3]の 2 次元配列として同時計算

Figure 8　流速 3 成分の計算方法の比較（BiCG 法）

4. 他の部分の Thread 並列化と高速化

CG 法と BiCG 法が大部分の計算時間を占めるが，他の部分も Thread 並列化を行って並列度を上げる必要がある．図 9 は OpenFOAM 内に良く現れるループパターンである．変数 x はループ内で複数回更新されるため，このままでは並列化できない．本報告では，2.1 節のように格子番号が付けられているので，図 10 のように色分けを行う．同じ色のブロックは独立に計算できる．

```
for (int fi=0; fi<nFaces; fi++) {
  double s1 = (some computation);
  int i=owner[fi], j=neighbor[fi];
  x[i] += s1;
  x[j] -= s1;
}
```

Figure 9　良く現れるループパターン

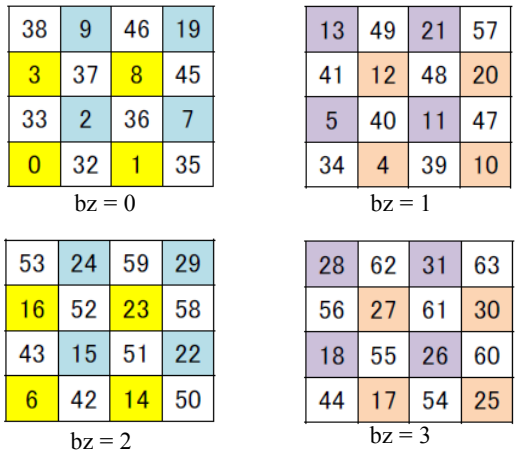


Figure 10　another Multicolor

OpenFOAM では図 11 のように vector（3 成分）や tensor（9 成分）の配列でループ内のテンポラリ変数を定義して使用している箇所が多く見られる．ループ内のテンポラリ変数として配列を使用すると計算性能が悪い場合があるため，本報告ではスカラー変数に置き換えている．該当箇所の計算時間は 1/3～1/6 に短縮された．

```
for (label fi=0; fi<nFaces; fi++) {
  vector ssf = iweights[fi]*(iU[P[fi]] - iU[N[fi]]) + iU[N[fi]];
  GradTypeXX Sfssf = iSf[fi]*ssf;

  igGrad[owner[fi]]     += Sfssf;
  igGrad[neighbour[fi]] -= Sfssf;

}
```

tensor と同じ 9 成分

Figure 11　テンポラリ変数に配列を使用した例

更に，作業領域が各部で確保／解放されており，非効率である．本報告では，時間増分ループ開始前に作業領域を確保して適宜割当てる方法とした．

5.　解析コードと解析モデル，解析条件

5.1 解析コード

次章で使用する解析コードを表 1 に示す．オリジナルの Piso は Kfast のオプションでコンパイルすると正しい計算結果とならないため, O3 でコンパイルしたコードのみとした．EK と HK は，展開されたコードを Kfast のオプションでコンパイルし，O3 のオプションでコンパイルした OpenFOAM ライブラリと link している．なお，HK は時間増分ループ内では OpenFOAM のクラスを殆ど使用していない．OpenFOAM のプログラム構造の維持よりも，性能の追求を試みたコードである．

なお，使用した OpenFOAM は 2.2,「京」コンピュータのコンパイラは K-1.2.0-18 である．

Table 1　解析コード

name	説明	コンパイラの最適化レベル
Piso	pisoFOAM．OpenFOAM 内のコード．	O3
E3	Piso のメインプログラム内に大部分のクラスを展開．計算の効率化も行う．	O3
EK	E3 の最適化レベルを Kfast に変更．	Kfast
HK	EK に対して本報告の内容を実装．次節表 2 の BMC2 と CM に対して，オーダリング方法に対応した計算方法を選択して実行．	Kfast

5.2 解析モデル

解析領域は $Lx \times Ly \times Lz = 16.0(m) \times 3.0(m) \times 2.0(m)$ である．風洞実験施設の測定部を模している．格子数等は表 2 の通りであり，約 500 万格子/ノードである．ノード形状は「京」コンピュータの tofu に合う形状とした．「order」の BMC2 は 2.1 の Block Multicolor を適用した Hybrid 並列用モデル，CM は Cuthill-McKee オーダリングで MPI 並列用である．Hybrid 並列用と MPI 並列用では 1 ノード当たりの格子形状を同じにしている．そのため，MPI 並列用のモデルでは，各プロセスに与えられる領域は X 方向に薄い形状である．

Table 2 解析モデル

ノード	MPI	order	格子形状	格子数	総格子数
32x6x4 (768)	768	BMC2	168x172x172	4,970,112	3,817,046,016
	6144	CM	21x172x172	621,264	3,817,046,016
16x6x4 (384)	384	BMC2	168x172x172	4,970,112	1,908,523,008
	3072	CM	21x172x172	621,264	1,908,523,008
16x3x4 (192)	192	BMC2	168x172x172	4,970,112	954,261,504
	1536	CM	21x172x172	621,264	954,261,504
16x3x2 (96)	96	BMC2	168x172x172	4,970,112	477,130,752
	768	CM	21x172x172	621,264	477,130,752
8x3x2 (48)	48	BMC2	168x172x172	4,970,112	238,565,376
	384	CM	21x172x172	621,264	238,565,376
4x3x2 (24)	24	BMC2	424x108x108	4,945,536	118,692,864
	192	CM	53x108x108	618,192	118,692,864
2x3x2 (12)	12	BMC2	712x84x84	5,023,872	60,286,464
	96	CM	89x84x84	627,984	60,286,464
1x1x1 (1)	1	BMC2	584x108x84	5,298,048	5,298,048
	8	CM	73x108x84	662,256	5,298,048

5.3 解析条件

本報告では，計算性能の検討が目的であるため CG 法と BiCG 法の収斂計算回数を固定している．時間増分ステップを 20 回とし，1 増分ステップ当たりの収斂計算回数は，圧力方程式を解く CG 法が 200 回（100 回+100 回），運動方程式を解く BiCG 法が各成分 5 回である．

6. 解析結果

解析コードと解析モデルの組み合わせは，表 1 の name と表 2 の order を用いて[name]_[order]としている．計算時間等の計測は時間増分ループを対象とし，「京」コンピュータで用意されている fipp コマンドを使用して計測した．

6.1 経過時間，GFLOPS 値，実行性能，実行性能比

図 12 は経過時間とノード数の関係である．オリジナルの Piso_CM に比べて，HK_CM と HK_BMC2 の経過時間は 1/2 以下であり，384 ノードの場合で 1/2.7 となっている。EK_CM と比べても 48 ノード以上では 1/2 以下である．なお, Piso_CM は unsigned int 型で保持している総格子数での除算があり，その上限値を超える 786 ノードのモデルの解析は実行できない。

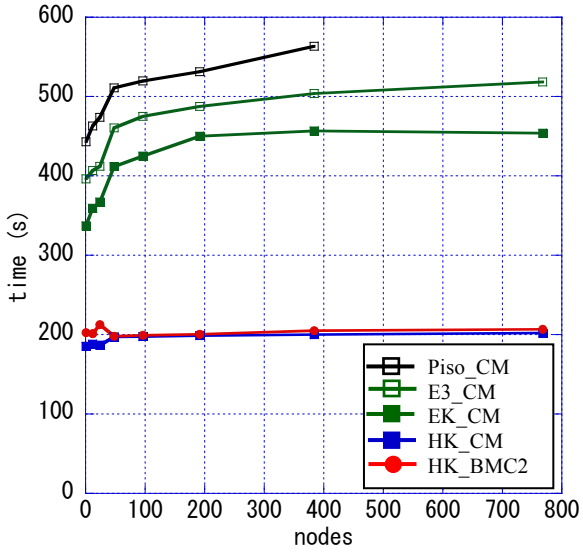


Figure 12　経過時間 vs. ノード数

図 13 は GFLOPS 値とノード数の関係である．EK_CM と HK_CM, HK_BMC2 はノード数が少ない場合を除いてほぼ線形に推移している．

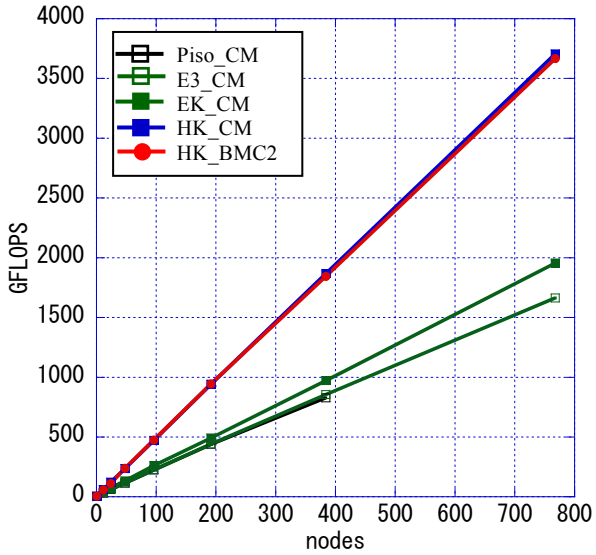


Figure 13　GFLOPS vs. ノード数

図 14 は実行効率とノード数の関係である．何れのケースもノード数が 48 までは急激に実行効率が低下している．EK_CM は 192～768 ノードでは実行効率の低下は殆ど見られないが，他は緩やかに低下して行くのが見られる．

予備解析として同じ格子数で serial job を実行したが，HK_BMC2 は HK_CM に比べて 10%程度遅い．また，HK_BMC2 の 8 threads での並列計算時の加速率は約 5.5 であった．ノード数が少ない場合の HK_BMC2 と HK_CM の差は，その影響が現れていると考える．なお，24 ノードでの HK_BMC2 の実行効率の低下は，一部の計算で thrashing が起こっているためと考えられる．1 プロセス当たりの格子数が多いほど，性能劣化時の影響は大きく現れる．

Piso_CM と E3_CM では経過時間（図 12）に差があるにも関わらず実行効率に差が見られない．このことより，E3 の計算量が Piso よりも減少していることが言える．

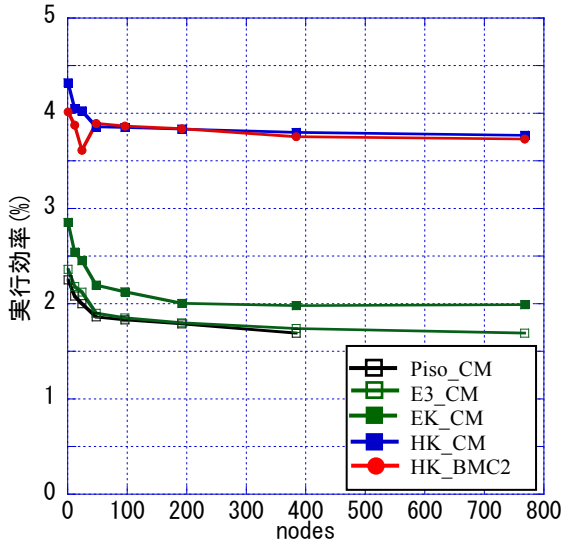


Figure 14　実行効率 vs. ノード数

図 15 は実行効率の変化の程度を見るために，実行効率を「1 ノードの解析モデル計算時の実行効率」で除した値を

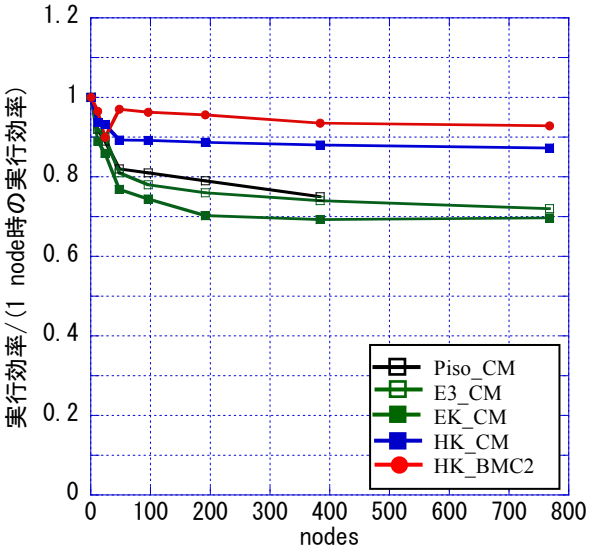


Figure 15　実行効率比 vs. ノード数

実行効率比として図示したものである．HK_BMC2 は HK_CM に対して，実行効率（図 14）ではやや劣るが低下割合は少ない．

これらの結果から，本報告で実装した方法は，性能向上に十分な効果があったと言える．一方，Hybrid 並列の優位性を示すまでには至らなかった．「京」コンピュータの通信機能が優れているために差が出なかったとも考えられる．

7. 結語

直交格子に限定しているが，多くの改良を行って十分な性能向上を実現した．Hybrid 並列の優位性を示すには至らなかったが，MPI 並列と同等の性能は実現できた．

今後，Hybrid 並列と MPI 並列での CG 法及び BiCG 法の収斂性の比較，多領域での収斂性の悪化への対処，非構造格子を対象とした性能向上と Hybrid 並列化を行う．最終的にはクラスライブラリ化して利用し易いものにしていく．

謝辞

本報告は，理化学研究所のスーパーコンピュータ「京」を利用して得られたものである（課題番号: hp150031）．ここに記して謝意を表する．

参考文献

1)中島研吾, マルチコア時代の並列前処理手法, 数理解析研究所講究録第 1733 巻, pp1-10, 2011.
2) 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.
3) 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.
4) P. Ghysels, W. Vanroose, Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm, Parallel Computing vol.40, pp224-228, 2014.
5) 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.
6) 藤野清次, 張紹良, 反復法の数理, 朝倉書店, 1996.