ポストペタスケール環境における大規模疎行列解法のための数値計算・通信ライブラリに関する研究

林雅江　中島研吾　石川裕

東京大学情報基盤センター

1 はじめに

有限要素法，有限差分法等の科学技術計算は最終的には大規模な疎行列を係数とする線形方程式を解くことに帰着される．大規模問題向けの解法として，クリロフ部分空間法に基づく前処理付反復法が広く使用されている．並列計算においては，隣接領域境界における一対一通信と Allreduce（内積）による通信が発生し，コア数（MPI プロセス数）の増加とともに，通信によるオーバーヘッドは無視できないものになる．本研究では，並列有限要素法による三次元弾性静解析プログラム(GeoFEM)ならびに並列マルチグリッド法による三次元地下水シミュレーションプログラム(pGW-MG)の二つのアプリケーションを具体例とし，計算科学と計算機科学の両分野の緊密な協力のもとに，数値計算アルゴリズムおよび通信の両方の観点から，ポストペタスケール環境における疎行列解法の確立を目標として実施した．具体的には 1 対 1 通信オーバーヘッドを削減した並列前処理付共役勾配法アルゴリズムを開発するとともに，石川らが開発中の Fujitsu FX10 のインターコネクトが有する RDMA（Remote Direct Memory Access）機能に基づく Persistent Communication をサポートする MPI ライブラリを適用し，更なる最適化を図る．本稿では 2 章で RDMA 機能に基づく MPI ライブラリについて述べ，3 章で実験に用いた二つのアプリケーションプログラムの概要を述べ，4 章で Oakleaf-FX を利用した大規模数値実験について示す．最後に 5 章で本稿をまとめる．

2 RDMA 機能に基づく Persistent Communication をサポートする MPI ライブラリ

2.1 Global Persistent Communication

多くの科学技術計算の並列化では，プロセスローカルな計算とプロセス間通信を繰り返す．繰返し中のプロセス間通信ではその相手先と送受信データ領域は固定していることが多い．

MPI 通信ライブラリにおける Persistent Communication はこのような通信パターンにおいて使われることが想定されて規格化されたものであり，MPI 1.0 通信規格から存在する．図 1 に Persistent Communication の利用例を示す．

```
for (i = 0; i < N; i++)    MPI_Recv_init(rbuf[i], count, MPI_DOUBLE, src[i], tag, com, &req[i]);
for (I = 0; i < N; i++)    MPI_Send_init(sbuf[i], count, MPI_DOUBLE, dest[i], tag, com, &req[i+N]);
do {
        /* Computation */
        MPI_Startall(N*2, req);
        /* Computation */
        MPI_Waitall(N*2, req, stat);
        / **** /
} while (…);
```

図 1　Persistent Communication 使用例

主ループ内で同じ送信パターン，受信パターンを発行する通信処理に対して，主ループに入る前にそれら要求を初期化し(MPI_Send_init, MPI_Recv_init)， MPI_Request 構造体の配列である req に要求を格納している．複数の要求を発行する MPI_Startall 関数を使い，MPI_Waitall 関数で処理の完了を待つ．ループの中で MPI_Isend/MPI_Irecv 関数を使うよりも簡素に記述できプログラムの見通しが良くなるのがわかる．Persistent Communication では，MPI_Send_init および MPI_Recv_init 時に通信バッファを獲得しておくなどの最適化が可能となるが，現在のハードウェアではその効果は限定的である．MPI_Send_init，MPI_Recv_init によって送受信間が関連付けられないため，送受信端それぞれローカルにできる最適化のみとなる．しかし，実際のアプリケーションは通信相手が固定されていることが多く，また，MPI_Startall 時には集団通信と同様コミュニケータに属する全てのプロセスが関与することが多い. そこで以下を仮定する MPI Persistent Communication を Global Persistent Communication と呼ぶことにする （[1]では Persitent Remote DMA と呼んでいる）.

1） MPI_Send_init, MPI_Recv_init によって通信相手が固定される．MPI_Recv_init では，送信元が特定される（MPI_ANY_SOURCE を使用しない）.
2） MPI_Startall は集団通信同様コミュニケータが属する全てのプロセスは MPI_Startall を呼び出す.

Global Persistent Communication では以下のような最適化が可能となる.

1） MPI_Startall 時，ひとたび送受信の通信データ領域が通知されれば，リモート DMA 通信機構を利用することができる. 最初の MPI_Startall 時には通信データ領域の授受が必要になるが 2 回目以降はその情報を再利用できる．ループ内で MPI_Isend/MPI_Irecv を使い大量のデータを通信する場合，MPI 通信ライブラリはランデブプロトコルという通信処理を行う．このプロトコルでは，受信側は受信アドレスを送信側に伝え，送信側はそのアドレスを受け取るとリモート DMA 通信機構を用いて直接受信メモリ領域にデータを転送する. 現在のほとんどの MPI_Startall 実装は, この実装を踏襲しているため, Persistent Communication においてもランデブプロトコルが使われ, 性能に大きな差がない．一方 Global Persistent Communication の実装では，同期は必要であるが，ランデブプロトコルのオーバヘッドを生じさせない.
2） MPI_Startall 時に同時に通信する相手と通信量が確定しているため，通信ハードウェアを効率よく使用することができる. FX10 では計算ノードあたり通信 DMA を 4 基搭載している．通信量の少ない順にこれら 4 基を利用して送信し，4 基の DMA の使用率を上げるよう送信スケジューリングを行うことが可能となる.

2.2 Global Persistent Communication の実装

富士通 MPI が有する MPI 拡張機能の一つである RDMA 機能を用いて Global Persistent Communication を実装した．富士通 MPI で利用できる RDMA 機能を表 1 に示す.

表 1　富士通 MPI の RDMA 拡張機能

FJMPI_Rdma_init	RDMA 機能の初期化
FJMPI_Rdma_finalize	RDMA 機能の終了
FJMPI_Rdma_get_remote_addr	メモリ ID のリモートアドレスの取得
FJMPI_Rdma_reg_mem	RDMA 可能メモリ領域の登録（メモリ ID が返値）
FJMPI_Rdma_put	リモートメモリに DMA による書き出す
FJMPI_Rdma_get	リモートメモリのデータを DMA により取り出す
FJMPI_Rdma_poll_cq	RDMA 完了確認

MPI 通信ライブラリは，ユーザが MPI 関数を上書きできる機能を有している．上書きされた関数の中で MPI 通信ライブラリのオリジナル関数が呼び出せるように, MPI_XXX の代わりに PMPI_XXX という名前の関数が定義されている．本実装では，以下の関数を再定義している．

- MPI_Init/MPI_Finalize
  Global Persistent Communication 実装のために必要な初期化，終了処理の記述

- MPI_Send_init/MPI_Recv_init
  メッセージサイズが小さい場合にはオリジナル関数を呼び出し，そうでなければ，Global Persistent Communication のための初期化を行う．本拡張のために独自の MPI_Request 構造体が定義される．アプリケーションプログラムに渡す MPI_Request 構造体は，オリジナル構造体と本拡張の構造体で区別できるようにする必要がある．オリジナル MPI_Request 構造体はアドレス値として渡される．本拡張の構造体のアドレスを渡すのではなく，アドレス値よりも小さい整数値を渡し区別する．本拡張では，メモリ領域が RDMA できるように登録するとともに, 送信側受信側の同期のためのメモリ領域の確保およびその領域が RDMA できるように登録する．
  - ➢ MPI_Recv_init は MPI_Send 関数を使って送信側に受信メモリ領域およびデータ送信完了を受け取るための同期メモリ領域を送信する．MPI_Send_init 側から送信された同期メモリ領域アドレス通知メッセージが到着していれば受信する．
  - ➢ MPI_Send_init は MPI_Send 関数を使って受信側との同期メモリ領域アドレスを送信する．受信側からのメモリ領域情報が受信できれば受信する．受信できなければ，MPI_Start/MPI_Startall 時に受信する．

- MPI_Startall/MPI_Start
  MPI_Request 構造体がオリジナルであれば，オリジナル関数を呼び出す．そうでなければ，以下の処理をする．
  - ➢ 受信の場合は，送信側からの同期メモリ領域アドレス通知メッセージが受信されていなければ，その到着を待つ．既にアドレスが分かっていれば，RDMA 機能を用いて送信側の同期メモリ領域に Ready to Send フラグを立てる．
  - ➢ 送信の場合，受信側からのメモリ領域情報が到着していなければ，その到着を待つ．

既にアドレスが分かっていれば，送信側の同期メモリ領域に Ready to Send フラグが立った後, RDMA 転送行い, その後受信側同期メモリ領域に送信終了フラグを立てる.

- MPI_Wait/MPI_Waitall/MPI_Test/MPI_Testall
  - MPI_Request 構造体がオリジナルであれば，オリジナル関数を呼び出す．そうでなければ，送信側は送信終了の確認を，受信側は同期メモリに送信終了フラグが立ったかの確認を行う.

Persistent Communication を使用したアプリケーションコードを変更することなく，本ライブラリをリンクするだけで Global Persistent Communication を利用することができる.

3 テストアプリケーションの概要

3.1 並列有限要素法による三次元弾性静解析プログラム「GeoFEM」

ターゲットアプリケーションの一つ目は 3 次元立方体形状における静弾性問題を扱う並列有限要素解析 GeoFEM[2] で，境界条件を図 2 の(1)に示す．また，ヤング率・ポアソン比は一様とする．各節点が 3 つの自由度を持つことから，剛性マトリクスにおいては 3x3 ブロック化がなされている．対称・正定な係数行列であることから解法には 3x3 ブロック対角スケーリング付き共役勾配法（Conjugate Gradient Method，CG）を用い，残差ノルム $|\{b\}-[A]\{x\}|/|b|$ が 10^{-8} 未満となるときを CG 法の収束条件とする．並列プログラミングモデルには FlatMPI と OpenMP/MPI Hybrid 並列の両方を実施した．また，4，800 ノードを用いた大規模並列計算に向け，三次元弾性静解析のメッシュデータをあらかじめ用意するのではなく，計算の始めに各 MPI プロセスが自身の解析領域データおよび隣接領域との情報（通信テーブル）を生成し並列計算を開始するようにした．一辺が 100 要素の立方体（100 万要素，約 300 万自由度）を一ノードに割り当てるとし，図 2 (2) で示すように利用するノード数に合わせて立方体を x, y, z の 3 方向に並べたものが全解析領域となる.

FlatMPI や Hybrid 並列のケースによっては一ノード当たりに割り当てる MPI プロセス数が異なるため，ノード当たりに起動する MPI プロセス数によって 100^3 の立方体がさらに分割され，各 MPI プロセスに割り当てられる．各部分領域は直方体となるため，隣接領域との接し方は面・辺・頂点の 3 パターンとなり，隣接数は最大 27，最少 7 となる．通信量は面で接する隣接ペアの間で最大となり, 本モデルでは最も大きいケースでも 240KB 程度の通信量となる．OpenMP レベルの並列化については，CG 法同様に並列化が容易な対角スケーリングを用いていることからソルバー全体を通し，行に関するループをスレッド数で分割している.

本プログラムにおける計算コストの 90%以上は連立一次方程式の解法部分（対角スケーリング付き CG 法）で占められ，そのほとんどが疎行列ベクトル積部分となっている．そこで疎行列ベクトル積部分の最適化を主眼とし，それに係る計算と通信のオーバーラップを実施した．その疑似プログラムを図 3 に示す．計算と通信のオーバーラップは，疎行列ベクトル積において局所データのみで計算できる行(indep_row)と演算を完結させるには通信が必要な行(dep_row)にあらかじめわけ，indep_row の計算と通信をオーバーラップさせる.

さらに，MPI_Isend・MPI_Irecv を使用した通信部分を 2 章で述べた Persistent Communication 通信である MPI_Send_init，MPI_Recv_init，MPI_Start に置き換えたもの

を図 4 に示す．MPI_Send_init，MPI_Recv_init は，これに 2 章で開発されたライブラリをリンクさせ実行ファイルを作成する．反復のループの中では常に同じ送信パターン，受信パターンで送受信が実行されるため，反復ループに入る前に MPI_Send_init, MPI_Recv_init にてそれら要求を初期化している．ループの内部で MPI_Start 関数を使い Receive 側の reqest, Send 側の request の順に発行している．MPI_Waitall 関数で処理の完了を待つ．

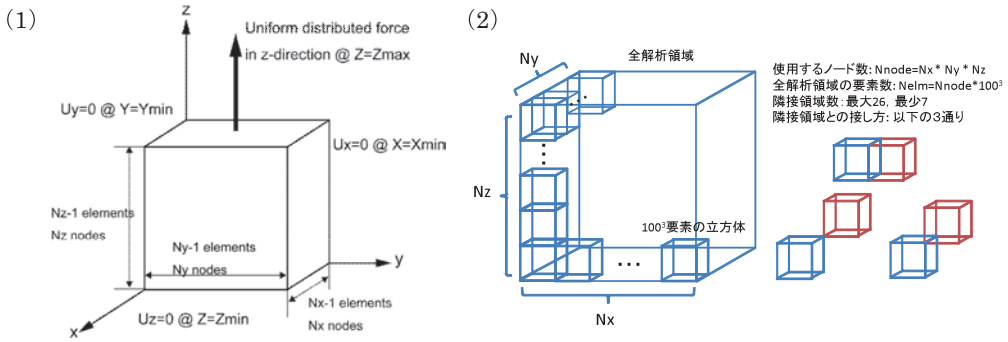
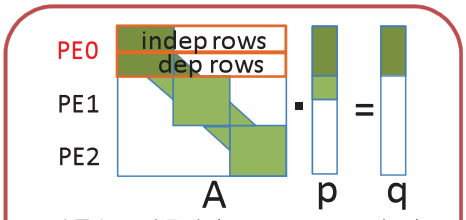


図 2　テストモデルの境界条件（1）と全体解析領域の形状（2）

```
//SEND
for( neib=1;neib<=NEIBPETOT;neib++){
        istart=EXPORT_INDEX[neib-1];
        inum  =EXPORT_INDEX[neib]-istart;
        for( k=istart+1;k<=istart+inum;k++){
            ii=3*EXPORT_ITEM[k-1];
            WS[3*k-3]=WW[P][ii-3];
            WS[3*k-2]=WW[P][ii-2];
            WS[3*k-1]=WW[P][ii-1];
        }
        MPI_Isend(&WS[3*istart], 3*inum, MPI_DOUBLE,
        NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req1[neib-1]);
}
//RECEIVE
for( neib=1;neib<=NEIBPETOT;neib++){
        istart=IMPORT_INDEX[neib-1];
        inum  =IMPORT_INDEX[neib]-istart;
        ii=3*(IMPORT_ITEM[istart]-1);
        MPI_Irecv(&WR[3*istart], 3*inum, MPI_DOUBLE,
          NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req2[neib-1]);
}

//BEGIN calculation for independent rows
#pragma omp parallel for private(ip, iS, iE, j, X1, X2, X3, WVAL1, WVAL2, WVAL3,
k, i, in)
for(ip=1;ip<=PEsmpTOT;ip++){
        iS=STACKindep[ip-1]+1;
        iE=STACKindep[ip ];
    for(in=iS;in<=iE;in++){
        j=row_indep[in-1];
        X1=WW[P][3*j-3];
        X2=WW[P][3*j-2];
        X3=WW[P][3*j-1];

            (indep_rowについてq=A･pの演算)

        WW[Q][3*j-3]=WVAL1;
        WW[Q][3*j-2]=WVAL2;
        WW[Q][3*j-1]=WVAL3;
    }
}
```

```
MPI_Waitall (NEIBPETOT, req2, sta2);
MPI_Waitall (NEIBPETOT, req1, sta1);

//BEGIN calculation for dependent rows
#pragma omp parallel for private(ip, iS, iE, j, X1, X2, X3, WVAL1, WVAL2, WVAL3,
k, i, in)
for(ip=1;ip<=PEsmpTOT;ip++){
        iS=STACKdep[ip-1]+1;
        iE=STACKdep[ip ];
    for(in=iS;in<=iE;in++){
        j=row_dep[in-1];
        X1=WW[P][3*j-3];
        X2=WW[P][3*j-2];
        X3=WW[P][3*j-1];

            (dep_row についてq=A･pの演算)

        WW[Q][3*j-3]=WVAL1;
        WW[Q][3*j-2]=WVAL2;
        WW[Q][3*j-1]=WVAL3;
    }
}
```



上図はPE0に注目した時のdep_row/indep_rowを示す．
割り当て行（濃い緑）の計算のうちdep_rowの計算には
ベクトルpの袖領域成分が必要となる。

図 3　CG 法における疎行列ベクトル積部分にて実施した計算と通信のオーバーラップ

```
//SEND
for( neib=1;neib<=NEIBPETOT;neib++){
        istart=EXPORT_INDEX[neib-1];
        inum  =EXPORT_INDEX[neib]-istart;
        for( k=istart+1;k<=istart+inum;k++){
            ii=3*EXPORT_ITEM[k-1];
            WS[3*k-3]=WW[P][ii-3];
            WS[3*k-2]=WW[P][ii-2];
            WS[3*k-1]=WW[P][ii-1];
        }
        MPI_Send_init(&WS[3*istart], 3*inum, MPI_DOUBLE,
        NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req1[neib-1]);
}
//RECEIVE
for( neib=1;neib<=NEIBPETOT;neib++){
        istart=IMPORT_INDEX[neib-1];
        inum  =IMPORT_INDEX[neib]-istart;
        ii=3*(IMPORT_ITEM[istart]-1);
        MPI_Recv_init(&WR[3*istart], 3*inum, MPI_DOUBLE,
          NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req2[neib-1]);
}


for(ITER=1; ITER<=MAXIT; ITER++){  //ITERATION START

                    (CG法アルゴリズム中略)

/***SpMV***/
//SEND&RECV
for( neib=1;neib<=NEIBPETOT;neib++){
        MPI_Start(&req2[neib-1]);
}
for( neib=1;neib<=NEIBPETOT;neib++){
        istart=EXPORT_INDEX[neib-1];
        inum  =EXPORT_INDEX[neib]-istart;
        for( k=istart+1;k<=istart+inum;k++){
            ii=3*EXPORT_ITEM[k-1];
            WS[3*k-3]=WW[P][ii-3];
            WS[3*k-2]=WW[P][ii-2];
            WS[3*k-1]=WW[P][ii-1];
        }
        MPI_Start(&req1[neib-1]);
}
```

```
//BEGIN calculation for independent rows
#pragma omp parallel for private(ip, iS, iE, j, X1, X2, X3, WVAL1, WVAL2, WVAL3,
k, i, in)
for(ip=1;ip<=PEsmpTOT;ip++){
        iS=STACKindep[ip-1]+1;
        iE=STACKindep[ip ];
    for(in=iS;in<=iE;in++){
        j=row_indep[in-1];
        X1=WW[P][3*j-3];
        X2=WW[P][3*j-2];
        X3=WW[P][3*j-1];
            (途中略)
        WW[Q][3*j-3]=WVAL1;
        WW[Q][3*j-2]=WVAL2;
        WW[Q][3*j-1]=WVAL3;
    }
}
MPI_Waitall (NEIBPETOT, req2, sta2);
MPI_Waitall (NEIBPETOT, req1, sta1);
//BEGIN calculation for dependent rows
#pragma omp parallel for private(ip, iS, iE, j, X1, X2, X3, WVAL1, WVAL2, WVAL3,
k, i, in)
for(ip=1;ip<=PEsmpTOT;ip++){
        iS=STACKdep[ip-1]+1;
        iE=STACKdep[ip ];
    for(in=iS;in<=iE;in++){
        j=row_dep[in-1];
        X1=WW[P][3*j-3];
        X2=WW[P][3*j-2];
        X3=WW[P][3*j-1];
            (途中略)
        WW[Q][3*j-3]=WVAL1;
        WW[Q][3*j-2]=WVAL2;
        WW[Q][3*j-1]=WVAL3;
    }
}

                    (CG法アルゴリズム中略)

/*** check torelance***/
if (RESID< eps) break;

}
```

図 4　CG 法における疎行列ベクトル積部分の通信に Persistent Communication の MPI 関数を適用

3.2 三次元地下水シミュレーションプログラム「pGW-MG」

本研究では，図 5 に示すような不均質な多孔質媒体中の三次元地下水流れを並列有限体積法（Finite Volume Method，FVM）によって解くアプリケーションを扱う．対象とする問題は以下に示すような，ポアソン方程式および境界条件である：

$$\nabla \cdot \left(\lambda(x, y, z)\nabla\phi\right) = q, \phi = 0 \; at \; z = z_{\max}$$

ここで，ϕは水頭ポテンシャル，$\lambda(x,y,z)$は透水係数で位置座標の関数であり，セル（cell）ごとに異なっている．透水係数は，地質統計学の分野で使用される Sequential Gaussian アルゴリズム[3]により発生させた値を使用した（図 5（a））．q は体積フラックスであり，本研究では

一様（=1.0）に設定されている．透水係数の最小値，最大値，平均値はそれぞれ 10^{-5}，10^{+5}，10^{0} となるように設定されている．有限体積セルは一辺長さ 1.0 の立方体である．
このような問題設定では，条件数が 10^{10} のオーダーとなるような対称，正定な悪条件マトリクスを係数とする線形方程式を解く必要がある．本研究で対象とするモデルは，各々 128^3 セルから構成される同じ不均質場に基づく部分モデルの集合である．したがって，x，y，z 各方向に周期的に同じ不均質パターンが繰り返される．

（a）　　（b）

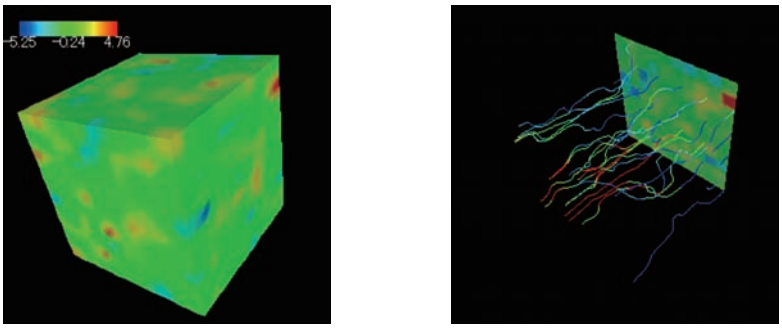


図 5　不均質多孔質媒体中の地下水流れの例，（a）透水係数分布，（b）流線

本研究では，ポアソン方程式を有限体積法によって離散化して得られる対称，正定（Symmetric Positive Definite，SPD）な行列を係数行列とする連立一次方程式を，多重格子法（Mulrigrid）による前処理を施した共役勾配法（Conjugate Gradient Method，CG）によって解く．このような前処理付き共役勾配法を MGCG 法[4]と呼ぶ．残差ノルム $|\{b\}-[A]\{x\}|/|b|$ が 10^{-12} 未満となるまで反復が繰り返される．多重格子法は大規模問題向けのスケーラブルな解法として注目されている．Gauss-Seidel 法などの古典的反復法はセルサイズに相当する波長をもった誤差成分の減衰には適しているが，誤差の成分のうち，長い波長の成分は緩和を繰り返しても中々収束しない．多重格子法は，長い波長の成分が粗い格子上で効率的に減衰するという考え方に基づいている[4]．多重格子法は，細かい格子において対象とする線形方程式の残差を計算し，修正方程式を粗い格子へ補間（制限補間，restriction）して解き，その結果を細かい格子に補間（延長補間，prolongation）して誤差を補正するというプロセスを，再帰的に多段階に適用することによって構築可能である．各レベルの計算が適切に実施されれば，誤差のあらゆる長さの波長をもった成分を一様に減衰させることができるため，計算時間が問題規模に比例するいわゆる「scalable」な手法の実現が可能である．本研究では，図 6 に示すように，8 個の「子（children）」セルから 1 個の「親（parent）」セルが生成されるような等方的な幾何学的多重格子法に基づき，格子間のオペレーションとしては，最密格子と最疎格子の間を直線的に動く V サイクル[4],[5],[6]を採用した．本研究では，各レベルにおける多重格子法のオペレーションは並列に実施されるが，最も粗い格子レベル（図 6 における Level=k）では 1 コアに集めて計算を実施する．

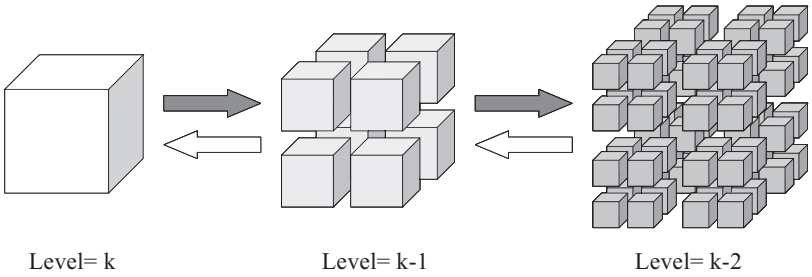


図 6　幾何学的多重格子法のプロセス（8 children=1 parent）

多重格子法では，各レベルにおける線形方程式を緩和的に計算するための演算子を緩和演算子（smoothing operator，smoother）と呼んでいる．緩和演算子として代表的なものはGauss-Seidel 法であり多くの研究で使用されているが，悪条件問題向けには不完全 LU 分解，不完全コレスキー分解が有効である[4]．本研究では，フィルインを生じない不完全コレスキー分解（IC(0)）を緩和演算子として採用した．IC(0)のプロセス（分解，前進後退代入）は大域的な処理を含むため，並列化は本来困難である．各領域において独立に IC(0)処理を実施するような，ブロック Jacobi 型の局所処理によって並列化は可能であるが，特に悪条件問題の場合，領域数が増えると収束が悪化する．ここで，加法シュワルツ法（Additive Schwartz Domain Decomposition，以下 ASDD）[4]を組み合わせることにより，並列計算においても安定した解を得ることが可能となる（図 7）．

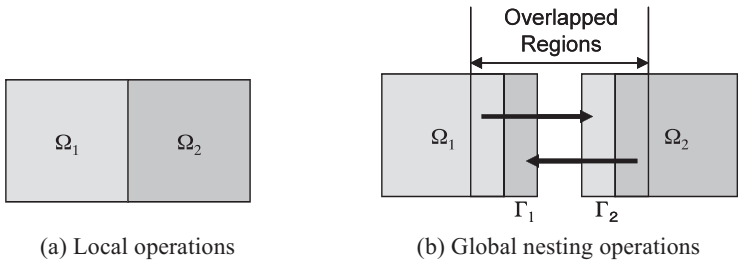


図 7　加法シュワルツ法（Additive Schwartz Domain Decomposition，ASDD）

OpenMP/MPI ハイブリッド並列プログラミングモデルで，FVM によるアプリケーションを並列化する場合，領域分割された各領域に MPI のプロセスが割り当てられ，各領域内で OpenMP による並列化が行われる．各領域においては，不完全コレスキー分解のように大域的な依存性を含むプロセスについては，各要素の並べ替え（reordering）により依存性を排除し，並列性を抽出する手法が広く使用されている[4]．本研究では，並列性が高く悪条件問題に対して安定な CM-RCM 法による並び替えを適用している[4]．本手法は，Reverse Cuthill-McKee（RCM）法とサイクリックに再番号付けする Cyclic マルチカラー法（cyclic multicoloring，CM）を組み合わせたものである．CM-RCM 法では各「色」内の要素は独立で，並列に計算を実行することが可能である．CM-RCM 法の色数の最大値は RCM におけるレベル数の最大値である．本研究では多重格子法の各レベルにおいて CM-RCM 法を適用している．

4 数値実験

4.1 GeoFEM

実験では 1200 ノード，2400 ノード，4800 ノードの三つの規模で性能評価を行った．一ノードあたりに割り当てる要素数は 100^3 要素に固定され，使用するノード数に対して規模を増加させていく Weak Scaling による性能評価を行った．Oakleaf-FX の最大 4800 ノードでの実行時には約 48 億要素（約 144 億自由度）の問題規模となった．OpenMP/MPI ハイブリッド並列プログラミングとしては，

- **Hybrid 2×8（HB 2×8）**:Oakleaf-FX 各ノードにスレッド数 2 の MPI プロセスを 8 つ起動する
- **Hybrid 4×4（HB 4×4）**:各ノードにスレッド数 4 の MPI プロセスを 4 つ起動する
- **Hybrid 8×2（HB 8×2）**：各ノードにスレッド数 8 の MPI プロセスを 2 つ起動する
- **Hybrid 16×1（HB 16×1）**：各ノードにスレッド数 16 の MPI プロセスを 1 つ起動する

のパターンを実施し，Flat MPI と比較した．

（1）最適化前

図 8 は，最適化前のプログラムを用いて 1200 ノードから 4800 ノードまでの Weakscaling による実験結果である．前処理付き反復解法における 1 コア当たりの Flops 値を示す．ノードあたりで適用するプログラミングモデルとしては Hybrid16x1（以下 Hybrid を HB と省略），HB8x2，HB4x4，HB2x8，FlatMPI の全ケースを実施した．4800 ノードでは最も性能がよいケース HB2x8 でコア当たり 7.75GFlops に達し，全体では 61TFlops が得られた．

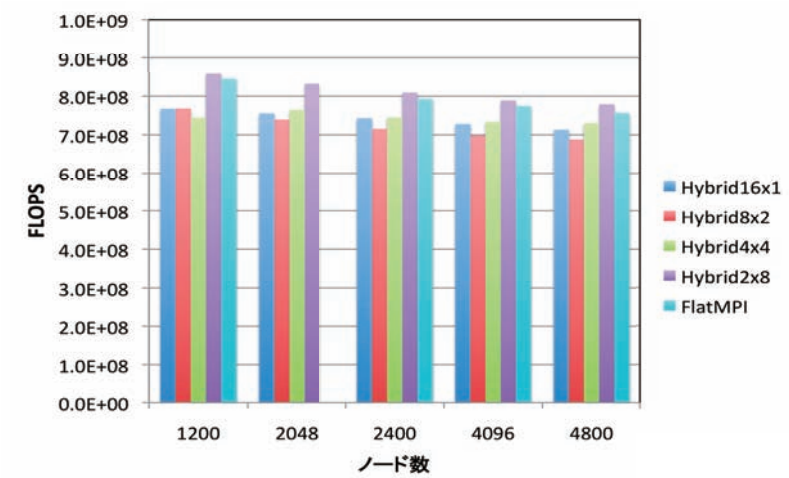


図 8　並列プログラミングモデルによるテストアプリケーションの性能評価

（2）通信のオーバーラップと RDMA 機能に基づく MPI ライブラリの適用

次に，計算と通信のオーバーラップさせた最適化プログラムを用い，4800 ノードで実施した際に RDMA 機能に基づく MPI ライブラリの適用効果を見たものを図 9 に示す．FlatMPI および全ハイブリッドプログラミングモデルのケースで実施し，1 コア当たりの FLOPS 値を測定した．微小ではあるが RDMA 機能をもつ通信ライブラリの効果が得られた．その効果が大きく現れなかった一因としては， 3.1 節で述べたように，本テストモデルにおいては通信ペア

間での通信量が比較的小さいことが挙げられる．RDMA 対応 MPI ライブラリがその効果を十分に発揮すると考えられる通信量に対して小規模であったためと考えられる．

また，通信量が小さいことで，今回実施した計算と通信のオーバーラップも大きな効果には結びつかず，結果的には RDMA 通信ライブラリを有効にした HB2x8 のケースが最高で 6.57GFlops となった．この性能後退の原因には，通信と計算のオーバーラップの実装において，indep_row と dep_row についてのリオーダリングを実施しなかったために間接参照を増やす実装となっていたこともある．そのため，Oakleaf-FX 上において RDMA 通信ライブラリの効果は確認できたものの，最適化前より Flops 値を下げることになった．リオーダリングにより間接参照を行わないような改善の後，再度検証実験が不可欠とされる．

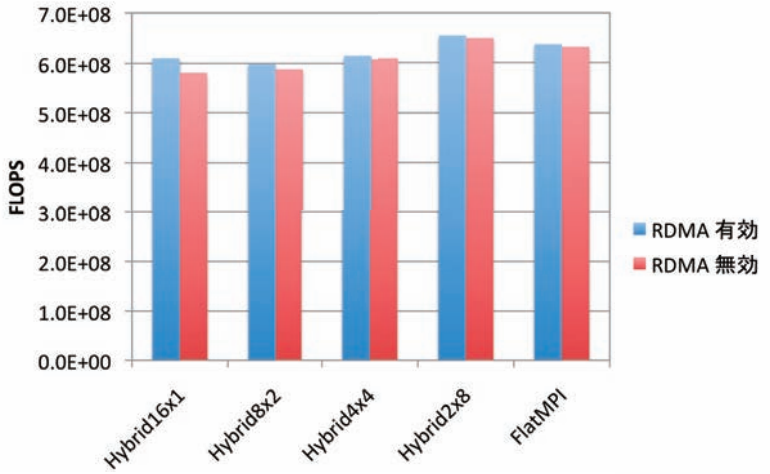


図 9　RDMA 通信ライブラリの効果

4.2　pGW-MG

（1）Coarse Grid Aggregation

既存研究[5]では V サイクルによる多重格子法は図 10 に示すような手順で実施されている．図 11 は図 10 の 1)，2)，3) のプロセスをまとめたものである．

1) Starting from finest level (level=1), a smoothing operation by IC(0), and *restriction* to coarser levels are applied. Operations at each MPI process are done by a parallel manner with some communication.
2) At the coarsest level, information on each process is gathered into a single MPI process, and a “coarse grid problem” is formed. The size of this coarse grid problem corresponds to the number of MPI processes.
3) **(Coarse Grid Solver):** A serial multigrid solver by IC(0) is applied for solving the coarse grid problem on a single core.
4) After the coarse grid problem is solved, the results of the coarse grid solver are scattered to each MPI process.
5) Starting from the coarsest level, *prolongation* to a finer level with smoothing at each level, is applied, until the finest level (level=1).

図 10　V サイクルによる多重格子法

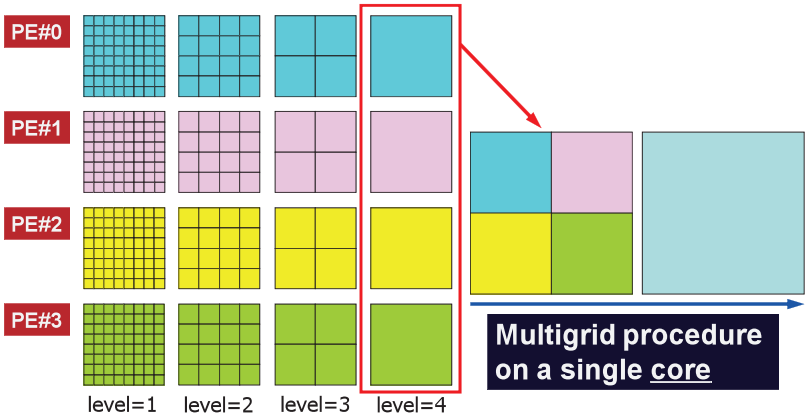


図 11　V サイクルによる並列多重格子法（4 MPI プロセス）における Coarsening／Restriction

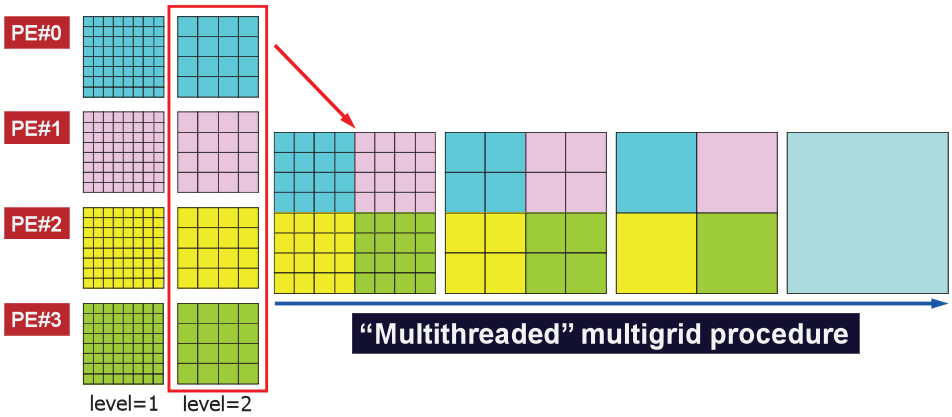


図 12 Coarse Grid Aggregation の例：level=2 の計算の後，各プロセスの結果は 1 MPI プロセスに集められる

中島によって提案された「Coarse Grid Aggregation」[6]では図 10 の 2)，4）における MPI プロセスの統合（aggregation）／分割（disaggregation）がより細かい（finer）レベルで実施されている.「Coarse Grid Solver」[4]への移行をより細かいレベルで実施することによって，収束の安定性，通信オーバーヘッドの削減が期待されるが，Coarse Grid Solver の解く問題サイズはオリジナルの設定と比較してより大きくなる．既存研究[5]では Coarse Grid Solver は 1 コアのみを使用しているが，[6]では各 MPI プロセス上の全コアを使用して OpenMP による並列化を実施している．ポストペタ／エクサスケールシステムでは，各ノードは $O(10^2)$規模のコアを含んでいるものと予測され，各ノードの多数のコアの計算能力を利用することを考慮する必要がある．図 12 は Coarse Grid Aggregation の例であり，ここでは level=2 の計算後に各 MPI プロセスにおける情報は 1 つの MPI プロセスに集められるため，Coarse Grid Solver は図 11 の場合よりも早い段階で開始している．

（2）Weak Scaling

各コアにおける問題サイズ（セル数）は 262,144（=643）を固定した Weak Scaling による性能評価を Oakleaf-FX の最大 4,096 ノード（65,536 コア）を使用して実施した．最大問題サイズは 17,179,869,184 である．以下に示す，3 種類の OpenMP/MPI ハイブリッド並列プログラミングモデルを適用し，全コアに独立に MPI プロセスを発生させる Flat MPI と比較した：

- **Hybrid 4×4（HB 4×4）**:Oakleaf-FX 各ノードにスレッド数 4 の MPI プロセスを 4 つ起動する
- **Hybrid 8×2（HB 8×2）**:各ノードにスレッド数 8 の MPI プロセスを 2 つ起動する
- **Hybrid 16×1（HB 16×1）**:各ノードにスレッド数 16 の MPI プロセスを 1 つ起動する

図 13（a），（b）は HB 8×2（CM-RCM（90））及び HB 16×1（CM-RCM（100））の 4,096 ノード使用時の Coarse Grid Aggregation の効果である．Coarse Grid Solver へ移行するレベルの効果が示してある．表 2 は各ケースにおける収束までの反復回数である．

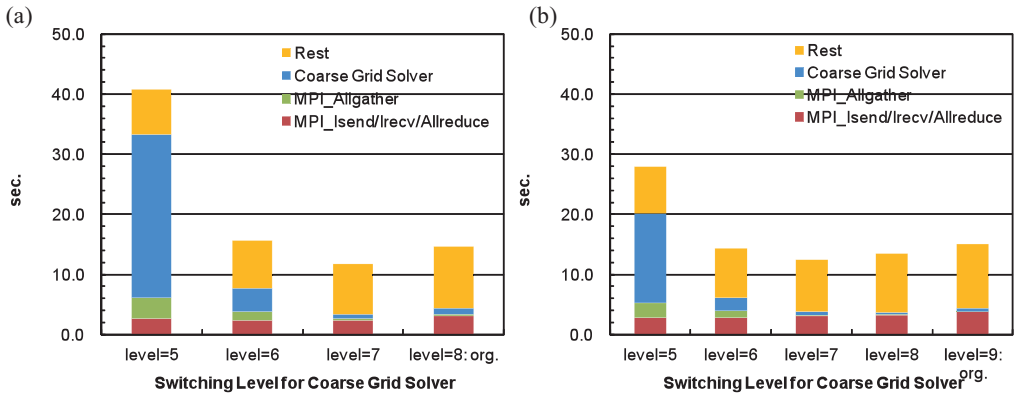
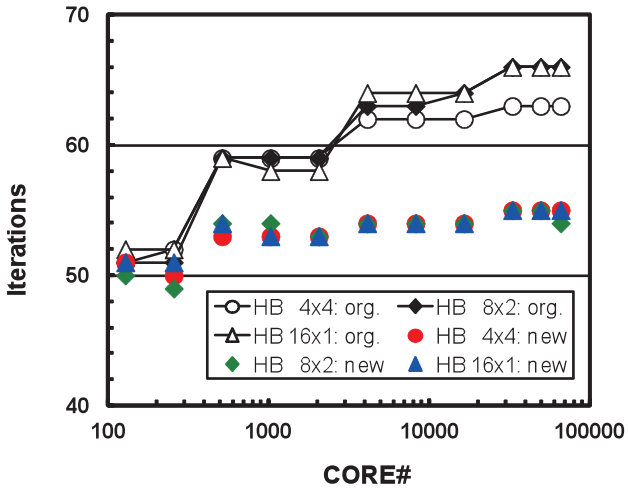


図 13　Oakleaf-FX における CM-RCM 法による MGCG 法の計算時間，4,096 ノード（65,536 コア），最大問題サイズ：17,179,869,184，Coarse Grid Solver への移行レベルの効果（a）HB 8×2（CM-RCM（90）），（b）HB 16×1（CM-RCM（100））

表 2　Oakleaf-FX における CM-RCM 法による MGCG 法の反復回数，4,096 ノード（65,536 コア），最大問題サイズ：17,179,869,184，Coarse Grid Solver への移行レベルの効果

Switching level for coarse grid solver	HB 8×2	HB 16×1
level=9	-	66
level=8	66	61
level=7	54	55
level=6	51	51
level=5	49	50

(a)



(b)

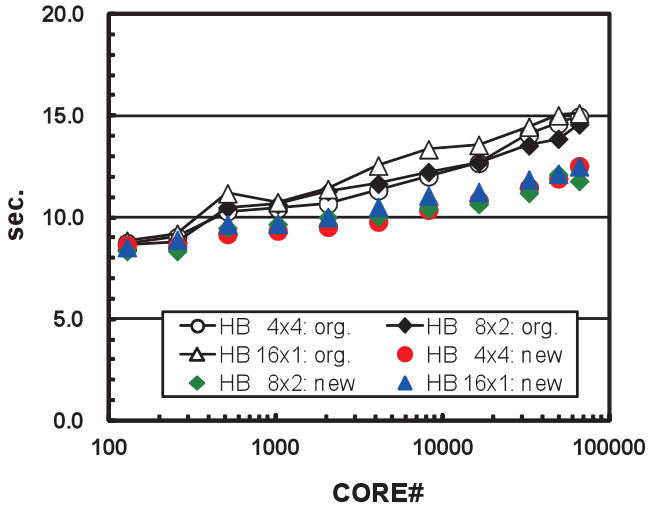


図 14　Oakleaf-FX における CM-RCM 法による MGCG 法の計算性能，Coarse Grid Aggregation の効果，最大 4,096 ノード（65,536 コア），Weak Scaling：コア当たり問題サイズ=262,144（=64^3），最大問題サイズ：17,179,869,184（a）MGCG 法の反復回数，（b）MGCG 法の計算時間 HB 4×4（CM-RCM（70）），HB 8×2（CM-RCM（90）），HB 16×1（CM-RCM（100））

HB 8×2 の level=8，HB 16×1 の level=9 では Coarse Grid Solver は，[5]では 1 コア上で動いている．[5]では ASDD による「局所型」IC(0)が各 MPI プロセスにおいて適用されている．本研究では Coarse Grid Aggregation により Coarse Grid Solver が 1 MPI プロセス上で適用されているため，IC(0)による緩和が局所型 IC(0)と比較してより効果的に行われている．

図 14 に示すように，より細かいレベル（level が小さい場合）において Coarse Grid Solver

へ移行することにより収束は安定し，反復回数は減少しているが，level=5, level=6 では Coarse Grid Solver の問題サイズが大きくなり，むしろ計算時間がかかっている．HB 8×2，HB 16×1 ともに level=7 の場合が最も計算時間が短い．また，HB 4×4 においても level=7 が最適である．

図 14（a），（b）は Weak Scaling における Coarse Grid Aggregation の効果である．全ケースで level=7 としている．図 14（a）に示すように，Coarse Grid Aggregation による収束の改善効果は顕著である．MGCG 法の Scalability は図 14（b）に示すように改善されている．4,096 ノード（65,536 コア）における計算時間は 12.5 秒（HB 4×4），11.8 秒（HB 8×2），12.5 秒（HB 16×1）である．既存手法[5]と比較して計算時間は 20%程度改善されている．

5 まとめ

ポストペタスケール環境における疎行列解法の確立を目標として，計算科学と計算機科学の両分野の緊密に協力し，数値計算アルゴリズムおよび通信の両方の観点から，大規模 HPC チャレンジに臨んだ．GeoFEM, pGW-MG の二つのアプリケーションが最大で 4,800 ノードおよび 4,096 ノードで行われた．一つ目のアプリケーション GeoFEM においては FX10 上で RDMA 機能に基づく Persistent 通信の利用が試みられた．アプリケーション側からだけのアプローチであれば，計算と通信のオーバーラップをはじめとするプログラム上での最適化が限界になってしまうことも多く，Isend/Irecv などの既存の通信関数以外を試みようとすることもなかなか難しい．MPI1.0 通信規格からあり，想定される通信パターンが適確であるにも関わらず使われていなかった Persistent Communication を実装に取り入れられたこと，さらにハードウェアに最適化された通信機能をもつ MPI ライブラリを活用しながらアプリケーションの最適化が試みられた事は大変有意義なことであったと同時に，今回のような通信分野とアプリケーション分野の緊密な連携が不可欠であると感じた．二つ目のアプリケーション pGW-MG では収束の安定性，通信オーバーヘッドの削減が期待される提案手法 Coarse Grid Aggregation が実施され既存手法と比較された．提案手法による収束の改善効果が得られ，20%程度の計算時間の改善が得られた．

最後に，Oakleaf-FX の全 4800 ノードが 24 時間利用できたことは規模・時間ともに大変恵まれた機会であった．HPC チャレンジの実施期間中，Oakleaf-FX が稼働開始してから間もなかったこともあり，昼夜を問わず富士通のサポートの方に対応いただいた．心より御礼を申し上げて，本稿のまとめとする．

参　考　文　献

[1] Yutaka Ishikawa, Kengo Nakajima, and Atsushi Hori, “Revisiting Persistent Communication in MPI,” EuroMPI 2012, 2012 (Poster).

[2] GeoFEM プロジェクト http://geofem.tokyo.rist.or.jp/

[3] Deutsch, C.V., Journel, A.G., GSLIB Geostatistical Software Library and User's Guide, Second Edition. Oxford University Press (1998).

[4] Nakajima, K., Parallel Multigrid Solvers using OpenMP/MPI Hybrid Programming Models on Multi-Core/Multi-Socket Clusters, Lecture Notes in Computer Science 6449, pp.185-199 (2011).

[5] Nakajima, K., New strategy for coarse grid solvers in parallel multigrid methods using OpenMP/MPI hybrid programming models, ACM Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores, ACM Digital Library (DOI: 10.1145/2141702.2141713) (2012).

[6] Nakajima, K., OpenMP/MPI Hybrid Parallel Multigrid Method on Fujitsu FX10 Supercomputer System, IEEE Proceedings of 2012 International Conference on Cluster Computing Workshops, 199-206, IEEE Digital Library: 10.1109/ClusterW.2012.35 (2012).