Vol.2012-HPC-135 No.15
2012/8/1

PGAS言語XcalableMPによるHigh Performance Linpackの作成と評価

中尾 昌広[1,a)]　朴 泰祐[1,2]　佐藤 三久[1,2,3]

概要：PGAS 言語 XcalableMP（XMP）を用いて High Performance Linpack（HPL）の作成を行い，その性能と生産性の評価を行った．HA-PACS ベースクラスタを用いて性能評価を行った結果，1 ノード内においては XMP を用いた実装はオリジナルの実装とほぼ同じ性能であるが，8 ノード（128 プロセス）を用いた場合，XMP を用いた実装はオリジナルの実装よりも約 30%低い値であることがわかった．XMP を用いた HPL の実装は，特にパネル転送の実装が非効率なので，その点を改善することで性能向上が見込める．また，生産性においては，XMP を用いると MPI を用いるよりも約半分のコード量で HPL を実装できることを示した．

1. はじめに

分散メモリ環境を対象とした新しい並列プログラミング言語として XcalableMP（XMP）[1–3] が注目されている．XMP は Partitioned Global Address Space（PGAS）言語の一種であり，プロセス間で透過的にアクセス可能なメモリ空間を持つ．また，XMP は分散メモリプログラミングで頻発する操作であるデータマッピングやワークマッピングを簡易に行う仕組みを有しているため，並列アプリケーション開発に対して高い生産性を発揮することが期待されている．さらに，XMP のプログラミングモデルはユーザにローカルメモリを強く意識させるモデルであるため，性能面においても優れていると考えられる．

本稿の目的は，C 言語版の XMP を用いて並列ベンチマーク High Performance Linpack（HPL）[4] の作成を行い，XMP の性能と生産性の評価を行うことである．また，XMP を用いた並列アプリケーション開発のノウハウの蓄積も行う．

2. High Performance Linpackの概要

Linpack ベンチマークは，非対称実数密行列の連立一次方程式を解く際の浮動小数点演算性能を評価するものである．HPL [4] は分散メモリ環境に対応した Linpack ベンチマークの一種である．HPL を利用するには Message Passing Interface（MPI）および数値計算ライブラリである Basic Linear Algebra Subprograms（BLAS）[5] もしくは Vector Signal Image Processing Library（VSIPL）[6] が必要である．HPL には様々なパラメータ（例：データマッピングに用いるブロックサイズ，パネル転送時の通信方法など）があり，測定対象のシステムに応じてパラメータを選択することができる．

HPL のアルゴリズムは，係数行列の LU 分解を行った後，前進消去および後退代入を行うことで解を算出するというものである．HPL の実行時間は LU 分解が多くを占めており，また LU 分解の実行時間は数値計算ライブラリを用いて行う計算が多くを占める．そのため，HPL では数値計算ライブラリが提供する計算を高効率に行うことが重要であると言える．

3. XcalableMP を用いた High Performance Linpack の実装

3.1 方針

XMP のプログラミングモデルにはグローバルビューモデルとローカルビューモデルがある．グローバルビューモデルは，逐次プログラムに近いイメージで XMP が定義する分散配列を扱うことができるモデルである．ローカルビューモデルは，片側通信を簡易に行うためのモデルであり，グローバルビューモデルよりも細かいデータの授受を行う操作に適している．HPL のように大規模配列を複数の小配列に分割する場合は，グローバルビューモデルを用

[1] 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba
[2] 筑波大学 大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
[3] 理化学研究所 計算科学研究機構
RIKEN Advanced Institute for Computational Science
[a)] mnakao@ccs.tsukuba.ac.jp

いる方が簡易に記述できるため，本稿では XMP のグローバルビューモデルを用いて HPL の実装を行う．

また前章で述べた通り，HPL では数値計算ライブラリが提供する計算を効率的に行うことが重要である．そこで，XMP で定義した分散配列に対して数値計算ライブラリを直接用いることを試みる．

XMP の実装には，筑波大学と理化学研究所が開発している Omni XcalableMP Compiler[*1]（TXMP）を用いる．しかし，XMP 仕様書 Version1.0 [1] に記載されている非同期通信機能は，TXMP ではまだ実装されていない．そのため，本稿では非同期通信機能を用いずに HPL の実装を行う．

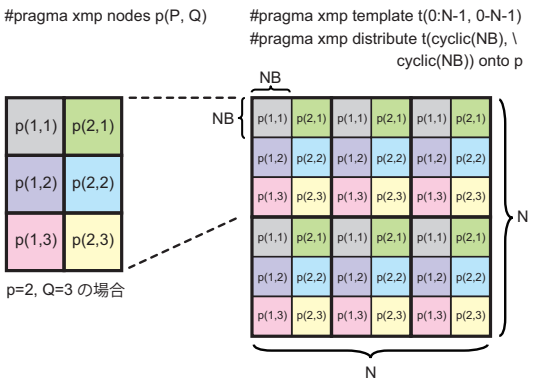

図 1 ブロックサイクリック分割
Fig. 1 Block cyclic distribution

3.2 実装

本節では，XMP を用いた HPL の LU 分解の実装について説明する．本節以降では，2 章で説明した HPL を “オリジナルの HPL” と呼び，XMP で実装した HPL を “XMP-HPL” と呼ぶ．

3.2.1 データマッピング

オリジナルの HPL では，連立一次方程式で用いる係数行列をブロックサイクリック分割で各プロセスに配置する．XMP-HPL においても同様の分割方法を用いる．下記に XMP のソースを示す．

```
#pragma xmp nodes p(P, Q)
#pragma xmp template t(0:N-1, 0:N-1)
#pragma xmp distribute t(cyclic(NB), cyclic(NB)) \
                                          onto p
double A[N][N];
#pragma xmp align A[i][j] with t(j, i)
```

1 行目ではデータマッピングに用いる 2 次元プロセスグリッドを作成している（**図 1** 左）．2 行目では 0 から N-1 までのインデックスを持つ 2 次元テンプレートを作成し，3 行目ではそのテンプレートをブロックサイズ NB 毎に 1 行目で宣言した 2 次元プロセスグリッドにサイクリックに割り当てる（図 1 右）．4 行目では通常の C 言語のように係数行列 A[][] を宣言し，5 行目では 3 行目までで定義したテンプレートに用いて係数行列 A[][] を各プロセスに割り当てる．

3.2.2 軸選択

オリジナルの HPL と同様に，解の精度を高めるために部分軸選択を行う．部分軸選択では，A[k][k] から A[N-1][k] までの列中の要素の最大値を求める．下記に XMP のソースを示す．

```
#pragma xmp loop on t(*,i) \
                reduction(FIRSTMAX:max/max_index/)
for(max=0,i=k;i<N;i++){
  if(max < A[i][k]){
    max = A[i][k];
    max_index = i;
  }
}
```

各プロセスでは，対象列中の最大値を変数 max に，その最大値を持つインデックスを変数 max_index に保存している．なお，変数 max_index に保存されるインデックスは，分散前の係数行列に対応するインデックス（いわゆるグローバルインデックス）であり，0 から N-1 までの整数値である．そして，for ループ文終了後に，全プロセスで最も大きな変数 max の値を持つプロセスが，自身が持つ変数 max_index を全プロセスにブロードキャストする．この操作により，全プロセスが同じ値の変数 max_index を持つことになる．

3.2.3 選択軸の行交換

係数行列 A[][] の max_line 行目と現在処理を行っている k 行目とを交換する．

```
double tmp_swap[N];
#pragma xmp align tmp_swap[j] with t(j,*)
...
#pragma xmp gmove
tmp_swap[:] = A[k][:];
#pragma xmp gmove
A[k][:] = A[max_index][:];
#pragma xmp gmove
A[max_index][:] = tmp_swap[:];
```

行交換を行うため，テンプレートの列に対してブロックサイクリック分割した作業用の配列 tmp_swap[] を作成する．gmove 指示文を用いて，下記の操作を行っている．

(1) 係数行列 A[][] の k 行目の要素全てを tmp_swap[] にコピー
(2) 係数行列 A[][] の max_index 行目の要素全てを係数行列 A[][] の k 行目にコピー
(3) tmp_swap[] の要素全てを係数行列 A[][] の max_index

[*1] http://www.xcalablemp.org/download.html

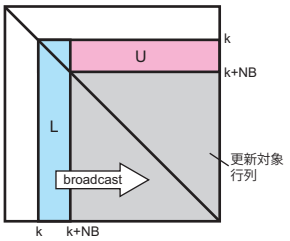

図 2 パネル転送
Fig. 2 Panel broadcast

行目にコピー

3.2.4 パネル転送

パネル転送の概要を**図 2** に示す．図 2 中の L に対してパネル分解を行った後，そのパネルを更新対象行列を持つプロセスに転送する．下記に XMP のソースを示す．

```
double tmp_L[N][NB];
#pragma xmp align tmp_L[i][*] with t(*,i)
...
#pragma xmp loop on t(i,*)
for(i=k; i<N; i++)
  for(j=0; j<NB; j++)
    tmp_L[i][j] = A[i][j];

#pragma xmp bcast (tmp_L) from t(k,*) on \
                              t(k:N-1,k:N-1)
```

作業用の分散配列 tmp_L[][] にパネルの情報を保存し，bcast 指示文を用いて更新対象行列を持つプロセスに tmp_L[][] を転送している．

その後は，図 2 中の U の更新を行い，U も L と同様に更新対称行列を持つプロセスに転送する．なお，U の更新については，BLAS ライブラリの三角行列ソルバー DTRSM を用いることができる．分散配列に対する DTRSM の適用方法は，次節で述べる行列積 DGEMM と同じであるため省略する．

3.2.5 更新計算

更新計算では分散配列 A，L および U に対する行列積演算（$A = A - LU$）を行う．高効率に計算を行うために，逐次用の数値計算ライブラリである BLAS を分散配列に対して適用させる．XMP における分散配列がどのようにローカルメモリにマッピングされるかについては，次期 XMP 仕様書において分散配列に対する推奨メモリレイアウトが記載される予定である．基本的には，**図 3** にあるように，連続した領域が確保される．また XMP では，分散配列に対するポインタはその分散配列に対するローカル配列のポインタを指すという規則がある．これらの情報から，BLAS に限らず，逐次用のライブラリを分散配列に対して直接用いることが可能である．下記に XMP のソースを示す．

```
cblas_dgemm(CblasRowMajor, CblasNoTrans,
            CblasNoTrans, local_y_length, local_x_length,
            NB, -1, &tmp_L[local_start_y][0], NB,
             &tmp_U[0][local_start_x], N/P, 1,
            &A[local_start_y][local_start_x], N/P);
```

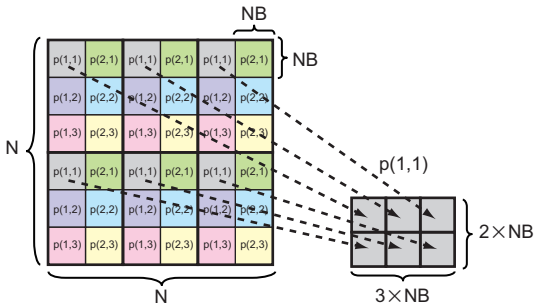

図 3 ブロックサイクリック分割のメモリレイアウト
Fig. 3 Memory layout of block cyclic distribution

表 1 HA-PACS ベースクラスタのスペックとソフトウェア環境
Table 1 Specification and software environment of HA-PACS Base Cluster

CPU	Intel Xeon E5-2670 2.60GHz (8 コア x 2 ソケット/ノード)
Memory	128 GB DDR3 1600MHz 4 チャネル/ソケット
Network	Infiniband QDR × 2 レール
C Compiler	gcc-4.7.0
MPI Library	MVAPICH2-1.7
BLAS Library	ATLAS-3.9.78

変数 local_y_length と local_x_length は手動で計算した行列積を適用させるローカル行列の各次元のサイズ，変数 local_start_y と local_start_x も同様に手動で計算した行列積を開始するローカル行列の先頭インデックスである．

4. 性能と生産性の評価

4.1 実験環境

実験環境には HA-PACS ベースクラスタを用いた．HA-PACS ベースクラスタは GPU を搭載しているが，CPU 部のみで計測を行った．今回の測定に関係する HA-PACS ベースクラスタのスペックおよびソフトウェア環境を**表 1** に示す．

XMP-HPL の実装には TXMP を用いたが，非同期通信以外の機能についても，3 章で説明した機能のいくつかはまだ実装されていない．そのため，TXMP のソースコードを解析し，TXMP が変換すると考えられるコードを手動で挿入することで，HPL を作成した．

4.2 性能評価

今回の実験では，1 プロセスに割り当てるメモリ量を約 512MB 程度になるように問題サイズ（N）を設定した．1 プロセスは 1CPU コアに対して割り当てる（シングルスレッド）．なお，1〜8 プロセスまでは 1 ソケット内に，16 プロセスの場合は 1 ノード 2 ソケットにプロセスを割り当てる．また，ブロックサイズ（NB）は事前評価の結果から 48 に設定した．

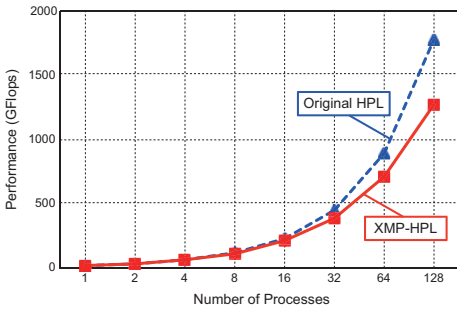

図 4 性能測定結果
Fig. 4 Performance result

表 2 生産性の評価（XMP-HPL の括弧は指示文の数を示す）
Table 2 Productivity result

	XMP-HPL	MPI-HPL
軸選択	12(1)	38
選択軸の行交換	8(3)	34
パネル転送	5(2)	15
更新計算	12(0)	12
前進消去および後退代入	60(12)	88
その他	80(27)	178
合計	177(45)	365

XMP-HPL の性能結果を**図 4** に示す．性能比較を行うために，XMP-HPL と可能な限り同じパラメータを用いたオリジナルの HPL の結果も示している．この結果から，1 ノード内の結果（1〜16 プロセス）は XMP-HPL とオリジナルの HPL との性能差はほとんどないが，2 ノード以上（32 プロセス以上）を計算に用いると XMP-HPL の性能はオリジナルの性能と比較して低くなることがわかる．128 プロセス時の性能差は約 30%である．

4.3 生産性の評価

HPL を実装するのに必要なコード量で生産性を評価する．なお，オリジナルの HPL のコード量は 8800 であるが，オリジナルの HPL では複数のアルゴリズムが実装されているため，比較対象として相応しくない．そのため，XMP-HPL と同等のアルゴリズムを MPI を用いて実装し，それと XMP-HPL とを比較する．

結果を**表 2** に示す．表中の MPI-HPL は MPI を用いた実装を示す．なお，空行，初期化処理，出力結果，時間計測，verification に関する行はカウントしていない．また，3.2 節で示した XMP-HPL のソースには，説明の都合上，グローバル領域で宣言している内容も記述してあるため，3.2 節で示したコードの行数と表 2 の数値とは必ずしも一致しない．この結果より，XMP-HPL は MPI-HPL の約半分の行数で記述できていることがわかる．

5. 考察

4.2 節の結果より，XMP-HPL とオリジナルの HPL との性能差の原因は通信部分にあると考えられる．そこで 3.2.4 節のパネル転送に着目する．XMP で実装を行ったパネル転送は，本来は必要でない要素まで更新対称行列を持つプロセスに転送している．この理由は，XMP の bcast 指示文は送信対象データの転送範囲を指定できないからである．この解決手段として，下記のように gmove 指示文を用いる方法があるが，不連続領域に対する gmove 指示文は TXMP ではまだ実装されていない．

```
#pragma xmp gmove
tmp_L[k:N-k][k:NB] = A[k:N-k][k:NB];
```

また，パネル転送において，オリジナルの HPL では隣接プロセスにデータを優先的に送信することにより高速化を図るなどの工夫がなされている．XMP において同様の工夫を行うためには，分散配列に対してローカルビューモデルによる片側通信を適用する必要があるが，その機能は TXMP ではまだ実装されていない．

6. まとめと今後の課題

本稿では XMP を用いて HPL を作成し，その性能と生産性を評価した．性能においては，HA-PACS ベースクラスタを用いて計測を行い，1 ノード内の結果においては XMP-HPL とオリジナルの HPL とでは性能差はないこと，しかし複数ノードを用いた結果においては XMP-HPL はオリジナルの HPL よりも性能が低くなることがわかった．生産性においては，XMP を用いると MPI を用いた場合の約半分のコード量で HPL を実装できることを示した．

今後の課題として，XMP-HPL を高速化するために TXMP に必要な機能である，非同期通信機能，不連続領域に対応した gmove 指示文，分散配列に対する片側通信機能などの作成が挙げられる．

参考文献

[1] XcalableMP Specification Working Group: XcalableMP Specification Version 1.0 (2011). http://www.xcalablemp.org/xmp-spec-1.0.pdf
[2] Nakao, M., Lee, J., Boku, T. and Sato, M.: Productivity and Performance of Global-View Programming with XcalableMP PGAS Language, The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (2012)
[3] 李珍泌, 朴泰祐, 佐藤三久: 分散メモリ向け並列言語 XcalableMP コンパイラの実装と性能評価, 情報処理学会論文誌コンピューティングシステム, Vol. 3, No. 3, pp. 153–165 (2010)
[4] Petitet, A., Whaley, R. C., Dongarra, J. and Cleary, A.: HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers (2010). http://www.netlib.org/benchmark/hpl/
[5] http://www.netlib.org/blas/
[6] http://www.vsipl.org/