Vol.2013-HPC-140 No.7
2013/7/31

XcalableMPによるNAS Parallel Application Benchmarksの実装と評価

津金 佳祐[1,a)]　佐藤 三久[1,2,3]　中尾 昌広[2]　村井 均[3]

概要：XcalableMP は C，Fortran 言語の拡張であり，分散メモリ環境を対象とした指示文ベースの並列言語である．本稿では，XcalableMP の生産性と性能の評価を行うために NAS Parallel Benchmarks（NPB）の Computational Fluid Dynamics（CFD）アプリケーションである Block Tri-diagonal solver（BT），Scalar Penta-diagonal solver（SP）及び Lower-Upper Gauss-Seidel solver（LU）の 3 種類について，XcalableMP のグローバルビューモデルとローカルビューモデルを用いた 2 つの実装を行った．結果として，ローカルビューモデルによる実装は，NPB の MPI 版（NPB3.3-MPI）とほぼ同等の性能となった．また，グローバルビューモデルによる実装は，性能は及ばなかったが，XcalableMP の言語仕様上の機能を用いることでほぼ同等の結果を得られるだろうことがわかった．加えて，グローバルビューモデルによる実装は，MPI 版と比較してシリアル版のソースコードを大きく変更することなく実装することができたため，XcalableMP の生産性を示すこともができた．

1. はじめに

分散メモリ環境上では並列プログラミングモデルとして，Massage Passing Interface（MPI）が広く普及している．しかし，MPI はプロセス毎のデータの分散配置を考慮するといった，並列化を行う上での様々な手順を明示的に示す必要があるため，プログラミングコストが大きいことやソースコードが煩雑になってしまうことが問題となっている．そこで，分散メモリ環境上での並列プログラミングをより容易にするために開発されたのが，XcalableMP である [1][2][3]．XcalableMP は，C や Fortran 言語といった既存言語を拡張した言語であり，OpenMP のようにプログラム中に指示文の追加を行うだけでループの並列化やプロセス間通信を行うことが可能である．そのため，既存のプログラムを大きく変更することなく容易に並列プログラミングを行うことができる．また，現在 XcalableMP のような既存言語を拡張した言語によって様々な並列アプリケーションが開発されている [4][5]．

NAS Parallel Benchmarks（NPB）は，NASA Advanced Supercomputing Division よって開発された，並列コンピューティングのためのベンチマークである．NPB は，5 つのカーネルコード（IS，EP，CG，MG，FT）と 3 つの Computational Fluid Dynamics（CFD）アプリケーション（BT, SP, LU）で構成されており，それぞれ MPI, OpenMP, シリアル版が存在する．カーネルコードである Integer Sort（IS），Embarrassingly Parallel（EP），Conjugate Gradient（CG）の 3 種類については中尾氏 によって XcalableMP 化が行われた [6]．

そこで本稿では，NPB の CFD アプリケーションである Block Tri-diagonal solver（BT），Scalar Penta-diagonal solver（SP）及び Lower-Upper Gauss-Seidel solver（LU）の 3 種類に対して XcalableMP を用いて実装を行い，XcalableMP の生産性と性能を評価することを目的とする．

本稿の構成は下記の通りである．2 章では XcalableMP の概要及びプログラミングモデルの説明を行い，3 章では NPB の XcalableMP による実装の方法を述べる．4 章では 3 章で作成した NPB の評価を行い，5 章では 4 章の結果についての考察を行う．最後に 6 章で本稿のまとめを行う．

2. XcalableMP

XcalableMP は，次世代並列プログラミング言語検討委員会及び PC クラスタコンソーシアム並列プログラミング言語 XcalableMP 規格部会により，仕様検討及び策定されている分散メモリ型 SPMD（Single Program Multiple

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

```
#pragma xmp nodes P(*)
#pragma xmp template T(0:N-1)
#pragma xmp distribute T(block) onto P
 int a[N];
#pragma xmp align a[i] with T(i)
・・・
#pragma xmp loop on T(i)
 for (i = 0; i < N; i++) {
    total += a[i];
 }
#pragma xmp reduction(+:total)
```

図 1 グローバルビューモデルプログラミングの例
Fig. 1 Example of Global View Model Programming.

Data）を実行モデルとする並列プログラミング言語である．プログラミングモデルとしては，指示文により各ノードにおいてのデータの分散，ループの並列処理，通信・同期を行う方式をとっており，ユーザが指示文を挿入しない限りそれらの動作が起こることはない．よって，ユーザが意図しない通信が起こることはなく，プログラムの挙動や結果を予測しやすいという利点がある．

XcalableMP はプログラミングモデルとして，グローバルビューモデルとローカルビューモデルの 2 つのプログラミングモデルが実装されており，これらを用いることで柔軟に並列プログラミングを行うことが可能である．以下で 2 つのプログラミングモデルの説明を行う．

2.1 グローバルビューモデル

グローバルビューモデルは，問題全体を各ノードに分散する指示文を記述することで，並列実行を行うプログラミングモデルである．データの分散には，テンプレートと呼ばれる仮想的なアドレス空間を用い，テンプレートに対して分割方法（ブロック分割，サイクリック分割及び任意数による分割）の指定を行い，分散したい配列との対応付けをすることによって，各ノードへとデータの分散を行う．これらの動作は全て指示文によるものであるため，ユーザは各ノードへと分散されたデータの配置を意識することなく並列実行を行うことが可能である．また，グローバルビューモデルによる XcalableMP プログラムは，XcalableMP 指示文を無視することで通常の C，Fortran 言語によるプログラムとして解釈することが可能である．

2.1.1 グローバルビューモデルプログラミング

簡単な例を用いて，グローバルビューモデルを用いたプログラミング方法の説明を行う．**図 1** の例は，実行時に指定されたノード数を用い，サイズ N の配列 a に格納された値の総和を並列実行により求め，全てのノードがその結果を保持するプログラムを表している．XcalableMP プログラムは，最初に nodes，template，distribute 指示文により，実行時に使用するノード数 P（*が指定された場合は，プログラム実行時に指定したノード数となる）の決定，分散を行う配列に使用するテンプレート T の定義，テンプレート T の分割方法（例はノード数によるブロック分割）を指定している．次に align 指示文によって，分散対象である配列 a に対してテンプレートとの対応付けを行う．これにより，配列 a を各ノードへとブロック分割したことになるため，並列実行可能となる．

```
#pragma xmp nodes P(*)
#pragma xmp template T(0:N-1)
#pragma xmp distribute T(block) onto P
 int a[N];
#pragma xmp align a[i] with T(i)
・・・
#pragma xmp gmove
 b[0:N] = a[0:N]
```

図 2 gmove 指示文の例
Fig. 2 Example of ”gmove” Directive.

for ループの前行にある loop 指示文は，直後の for ループを並列実行するように指定する指示文である．並列実行後，各ノードは分散された配列 a の和の値を変数 total に保持しているため，変数 total の値の総和を求める必要がある．それには，reduction 指示文による集約演算を行う．reduction 指示文は括弧内のコロンの後で指定された変数，または配列に対して集約演算を行う指示文であり，コロンの前の”+”は総和を求めることを表す．また，他にも総乗”*”や最大値”MAX”といった集約演算も行うことが可能である．これにより，配列 a の総和の値を全てのノードが得ることができる．

2.1.2 同期・通信指示文

XcalableMP には定型的な通信指示文が多数存在するが，本節では 3 章の実装で用いる gmove，shadow，reflect 指示文についてのみの説明を行う．

gmove 指示文は，分散配列に対するデータ通信を行う指示文である．**図 2** のように記述され，この例の場合，分散配列 a のインデックス 0 から N-1 の値をローカル配列 b のインデックス 0 から N-1 へのコピーを表している．このように，分散配列であっても指示文を追加することでローカルな値を参照するようにデータを用いることが可能となる．また，gmove 指示文は，分散配列同士やローカル変数・配列に対しても使用可能である．

shadow 指示文は，分散配列の分散境界を挟んで隣接する上端・下端の要素の値を一時的に保持する領域である，袖領域の確保を行う指示文である．袖領域の確保には，分散配列の上端・下端に任意数の領域を定義することが可能であり，分散配列の端のデータを持つノードに対しては，片側のみに袖領域が作成される．shadow 指示文によって確保された袖領域の値の更新は，reflect 指示文によって行うことが可能である．

```
 int a[N];
#pragma xmp coarray a:[*]
・・・
if (node_id == 2) {
 b[0:3] = a[3:3]:[1];
}
#pragma xmp sync_memory
```

図 3 ローカルビューモデルプログラミングの例
Fig. 3 Example of Local View Model Program.

2.2 ローカルビューモデル

ローカルビューモデルは，各ノードが持つローカルデータに対して通信を行うプログラミングモデルである．XcalableMP では，Fortran 言語の拡張記法である Coarray Fortran[7] をベースとした coarray 記法 [8] が実装されており，ノード番号を指定することによるデータの片方向通信を行うことができ，MPI のようなノード毎の振る舞いを個別に記述することが可能である．**図 3** の例では，ノード番号 1 が持つ配列 a のインデックス 0 から 3 個の値をノード番号 2 の配列 b のインデックス 3 から代入することを表している．coarray 記法は，配列括弧内のコロンの前の値が配列のインデックスをを表し，後の値が個数を表している．配列の後に記述されているコロンの後の括弧は，通信相手のノード番号となる．図 3 の最後の行にある sync_memory 指示文は，coarray 記法による通信の保証を行う指示文である．

3. 実装

XcalableMP は現在，C 言語版の開発が進んでいるためこちらを用いる．また，NPB は Fortran 言語での実装しか存在しないため，NPB のシリアル版（NPB3.3-SER）と MPI 版（NPR3.3-MPI）を C 言語を用いて書き換えを行い，XcalableMP 化を行う．また，実装には，グローバルビューモデルとローカルビューモデルを用いた 2 つのモデルでの作成を行う．グローバルビューモデルでの実装は，基本的にはシリアル版への指示文の追加のみでの実装を目指すが，必要であればソースコードの変更も行う．

3.1 BT

BT は，5 x 5 の非優位対角なブロックサイズを持つ 3 重対角方程式を解くベンチマークであり，3 次元の alternative direction implicit（ADI）法を用いて解かれる．

3.1.1 グローバルビューモデルによる実装

MPI 版による実装は，x，y 及び z 軸方向のデータを表す 3 次元配列に対して，3 次元全てで分割を行っており，それに加えてセルという概念を利用し，データの分散，並列化を行っている．この実装は，シリアル版と比較して，大幅にソースコードの変更を行っている．そのため，XcalableMP を用いた実装の方針としては，MPI 版とは異なり，3 次元配列の z 軸方向のデータに対して 1 次元のブロック分割を行う方針をとる．以下で指示文を追加する以外に変更を行った箇所の説明を行う．

```
#pragma xmp nodes P(*)
#pragma xmp template TZ(0:K-1) onto P
#pragma xmp template TY(0:J-1) onto P
 double rhs [K][J][I], zrhs[K][J][I]
#pragma xmp align rhs [k][*][*] with TZ(k)
#pragma xmp align zrhs[*][j][*] with TY(j)
・・・
#pragma xmp gmove
 zrhs[0:K][0:J][0:I] = rhs[0:K][0:J][0:I];
・・・
```

図 4 gmove 指示文による分散配列のコピー
Fig. 4 Copy Operation of Distributed Array by Using "gmove" Directive.

関数 rhs は，z 軸方向の分散配列である配列 u の値によって求められた配列 6 種類を用いて，x，y 及び z 軸方向とそれぞれの軸方向へと順に演算を行う関数である．各軸方向への演算処理中には，例えば，z 軸方向への演算ならば，配列 u も含んだ上記の配列全てに対して z+1，z-1 といったインデックスによる演算が存在する．分散配列の境界のインデックスの演算については，隣接ノードにマッピングされている値を必要とするため，shadow 指示文による袖領域の定義，reflect 指示文による袖領域の更新が必要となる．しかし，シリアル版の実装のままでは，配列 u も含めた上記の配列全てに対して reflect 指示文による袖領域の更新を行う必要があるため，ノード数が増えた場合に通信時間が大きなものとなる．そこで，配列 u の値により求められた配列を用いずに各軸方向による処理中に演算をさせることにより，各ノードでの演算量は増えてしまうが，reflect 指示文による同期を配列 u の 1 回のみとし，通信時間を減らすことで性能を改善することができる．

関数 x_solve，y_solve 及び z_solve は，3 次元配列の x，y 及び z 軸方向に対して順に走査を行う関数である．これらの関数では，各軸方向に対して運搬依存のある演算が交互に現れるため，ある軸方向に対して演算を行う場合には，他の 2 軸は並列実行可能となっている．そのため，z 軸方向で 1 次元のブロック分割を行った場合，関数 z_solve を並列実行することが不可能となる．そこで，関数 z_solve での並列実行を行うために，予め y 軸方向で分割を行った作業用配列を用意しておき，**図 4** のように gmove 指示文を用いることで分割領域の異なる配列同士のコピーを行う．これにより y 軸方向に並列実行を行うことが可能となる．また，y 軸方向に並列実行後，関数 z_solve 以外では z 軸方向へと分割されているため，その演算結果が格納された分散配列に対して再度 gmove 指示文を用いることで元の分割方向へと戻す．

```
for (k = 1; k < nz - 1; k++) {
 jacld(k);
 blts(k, ・・・);
}
```

図 5 関数 ssor
Fig. 5 Function of "ssor".

```
 node_id = xmp_all_node_num();
 node_num = xmp_all_num_nodes();
・・・
for (k = 1; k < nz + node_num; k++) {
#pragma xmp reflect (rsd)
 k_id = k - node_id;
 if (k_id <1 || nz - 1 <= k_id) continue;
     jacld(k_id);
     blts(k_id, ・・・);
}
```

図 6 関数 ssor（擬似的パイプライン実装）
Fig. 6 Function of "ssor"（Pseudo Pipelined Implementation).

3.1.2 ローカルビューモデルによる実装

BT の MPI 版の実装は，MPI_Isend，MPI_Irecv による非同期通信を行い，ある点で MPI_Wait により同期をとることで袖領域の更新を行っている．そのため，coarray 記法により片方向通信を記述し，sync_memory 指示文により同期をとることで MPI 版と同様の位置で片方向通信を行う実装をとる．

3.2 SP

SP は，5 x 5 の非優位対角なスカラ 5 重対角方程式を解くベンチマークである．主要な処理部分は，反復回数や通信に対する演算量に違いがある点以外，基本的なアルゴリズムとして BT と同様に ADI 法を用いている．よって XcalableMP により，3 次元配列の z 軸方向による 1 次元のブロック分割を行い，3.1 節と同様の実装をグローバルビューモデルとローカルビューモデルのそれぞれを用いて行う．

3.3 LU

LU は，実際には LU 分解を行わず，5 x 5 のブロックサイズを持つ上下三角行列に対して，Symmetric Successive Over-Relaxation（SSOR）法を用いて解くベンチマークである．

3.3.1 グローバルビューモデルによる実装

MPI 版の実装では，実行ノード数により分割対象である配列の次元数を変更していたが，XcalableMP の実装では分割を行う次元数を予め決定しておく必要があるため，実装の方針としては，y 軸方向のデータに対して 1 次元のブロック分割を行う方針をとる．以下で指示文を追加する以外に変更を行った箇所の説明を行う．

LU は x，y 及び z 軸方向と全ての軸方向に運搬依存性のあるループが存在する．XcalableMP では，このようなループ処理に対して現在の実装では，指示文を追加するだけでの並列実行を行うことは不可能である．そこで，村井氏による擬似的にパイプライン処理をする実装を行う [10]．以下に簡単な説明を示す．**図 5** は LU の中心処理である関数 ssor の処理の一部を表しており，z 軸方向 (ループ変数 k) のループ内の関数 jacld，blts で 3 軸方向に対しての運搬依存のある処理が存在するため，指示文による並列実行を行うことができない．そこで，z 軸方向の各ノードが実行するインデックス k に対してパイプライン処理を行うように変更を行った．**図 6** が変更を行ったソースコードである．パイプライン処理には，実行ノード数とノード番号が必要であるため，予め XcalableMP の API である xmp_all_num_nodes で実行ノード数, xmp_all_node_num でノード番号を求めている．z 軸方向のループ変数 k に対して，ノード番号との差の値を用い，z 軸方向のループ範囲内であれば実行を行うように変更することで擬似的にパイプライン処理を実行することが可能である．また，LU は，全体として y 軸方向への分散を行っているため，関数 jacld，blts 中の y 軸方向の運搬依存のある処理は袖領域で対応している．そのため reflect 指示文によりループ内で同期を実行している．これによりパイプライン処理の制御も行うことが可能である．

3.3.2 ローカルビューモデルによる実装

LU の MPI 版の実装は，MPI_Send，MPI_Recv による同期的な通信を行っている．そのため各ノードがそれぞれ送信側，受信側のどちらが先に通信待ち状態になるかわからず，現在の XcalableMP では実装を行うことができなかった．

表 1 実験環境
Table 1 Experimental Environment.

	Cray XK6m-200 System
CPU	Opteron Processor 6273 2.1GHz
Memory	32GB
Network	GEMINI Network 2D メッシュ
OS	CNL（Compute Node Linux）
C Compiler	cc 8.1.6 (Cray C)
Fortran Compiler	ftn 8.1.6 (Cray Fortran)

4. 評価

4.1 実験環境

3 章で説明をした BT，SP 及び LU のグローバルビューモデルとローカルビューモデルによる実装についての評価を行う．NPB の問題のサイズや反復回数を表す Class は，Class B とし，コンパイラオプションの最適化レベルは O3 を用いる．また，実験環境として Cray XK6m-200 システムを用いる．1 ノードの性能は**表 1** のようになっており，最大で 16 ノードを用いて計測を行う．CPU には，Opteron

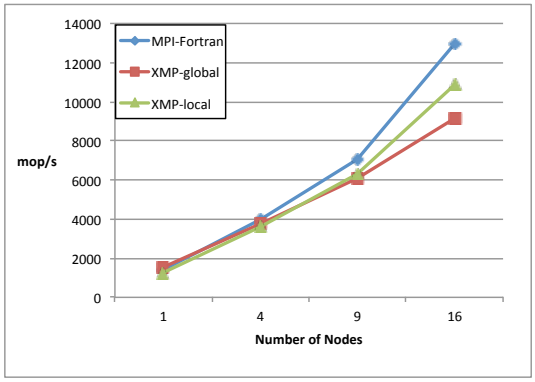

図 7 BT の評価結果
Fig. 7 Evaluation Result of BT.

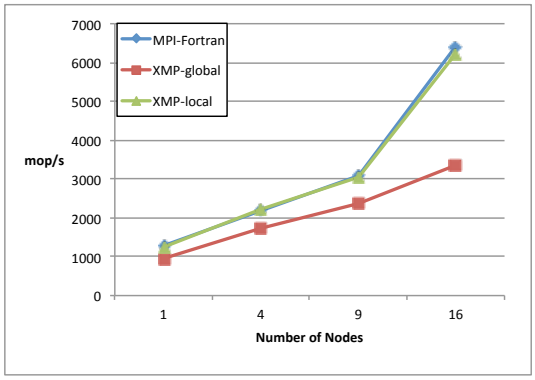

図 8 SP の評価結果
Fig. 8 Evaluation Result of SP.

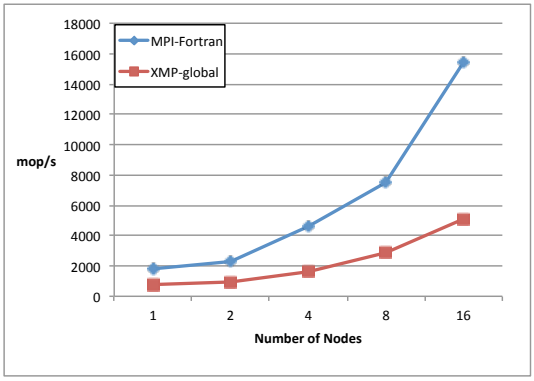

図 9 LU の評価結果
Fig. 9 Evaluation Result of LU.

表 2 BT の通信時間と処理時間（秒）
Table 2 Communication and Execution time of BT（sec）.

Nodes	gmove 指示文	反復処理時間
4	0.24	1.61
9	0.23	0.74
16	0.21	0.43

表 3 SP の通信時間と処理時間（秒）
Table 3 Communication and Execution time of SP（sec）.

Nodes	gmove 指示文	反復処理時間
4	0.17	0.55
9	0.17	0.41
16	0.16	0.24

Processor 6272 を用いているため 16 コアあるが，各ノードはノード内での並列化を行わずに 1 コアのみを使用する．評価の比較対象としては，NPB の MPI 版（NPB3.3-MPI）を用いる．NPB の MPI 版は制約上，実行ノード数に制限があり BT，SP は n の 2 乗，LU は 2 の乗数となるため，XcalableMP 版のノード数もこれに合わせる．評価回数は 5 回とし，実行した中の最も良い値を評価値とし，評価値には NPB の評価尺度である mop/s を用いる．

4.2 評価結果

図 7 に BT，**図 8** に SP の評価結果を示す． BT と SP のグローバルビューモデルによる実装は，gmove 指示文による分割次元の異なる分散配列のコピーによる通信の割合が実行時間の多くを占め，高速化を行えていない．同様の XcalableMP による実装を用いているにもかかわらず，BT と SP の結果に大きな差ができているのは，BT と比較し SP は，反復回数が多いことから，gmove 指示文による通信回数が BT よりも多いためである．また，ノード数が増える毎に袖領域の同期にかかる通信量も増加することから XcalableMP 版は，MPI 版のようにノード数 9 から 16 にかけての傾向は見られない．

ローカルビューモデルによる実装は，MPI 版と比較して BT，SP 共にだいたい同じ結果を得ることができた．また，実装の参考のために作成した C 言語による NPB の MPI 版と比較した結果，BT，SP 共に評価結果がほぼ同一であったため評価結果図 7，図 8 にある MPI 版と XcalableMP のローカルビューモデルによる実装との差は，C と Fortran 言語による実装の違いである．

図 9 に LU の評価結果を示す．LU は，擬似的にパイプライン処理を行う際に reflect 指示文による同期が必要となるため，ノード数を増やした場合に通信量が大きくなものとなり，MPI 版と比較して大きな差が出てしまった．

5. 考察

BT と SP のグローバルビューモデルによる実装である，gmove 指示文による分散次元の異なる作業用配列へのコピーについての考察を行う．この 2 つのベンチマークは，関数 z_solve 内で 2 つの配列に対して gmove 指示文を用いてコピーを行なっている．このコピーされる配列のうちの 1 つは，1 回の反復処理内で一度更新された後は，次の反復まで値の更新をされることがない．そのため配列が更新された後，次に関数 z_solve が呼ばれる間に非同期的に通信を行い，作業用配列へコピーを行うことが可能である．そこで，この gmove 指示文によるコピーの処理時間と配

列の値が更新されてから次に関数 z_solve が呼ばれるまでの実行時間の調査を行った．問題のサイズは，評価結果と同じ Class B を用いた．その結果，**表 2**，**表 3** のような結果を得た．表の gmove 指示文の項目は，関数 z_solve 内の gmove 指示文によって通信される 1 つの配列の実行時間に対して反復回数で割ったものであり，同様に反復処理時間の項目は，配列の値が更新されてから関数 z_solve が呼ばれるまでの実行時間に対して反復回数で割ったものである．この結果，BT，SP 共に 16 ノードまでは，非同期通信によって gmove 指示文による通信時間を隠すことが可能である．

LU の同期処理についての考察を行う．LU は，MPI 版の関数 rhs 内の処理では，配列 u に対する分割配列の袖領域の更新しか行なっておらず，XcalableMP 版では，それを加えた 2 つの配列 u，flux に対して reflect 指示文による同期で袖領域の更新を行なっている．ここで配列 flux は，配列 u の値を用いて求められている．MPI 版では，袖領域を 1 つ分多く通信を行い，各ノードで分散領域を上端・下端共に 1 つ分余分に演算することにより，演算量は増えるが，別配列の袖領域の更新による同期の通信回数を減らす方法をとっている．XcalableMP では分散配列に対して，分割領域を超えた領域には袖領域を定義することで使用することはできるが，loop 指示文による並列計算を袖領域を含めた範囲での演算を行うことができないため，この MPI 版同様の実装を行うことができなかった．また，3.1 節のように配列 flux を使わず，その都度配列 u の値を用いて演算を行い，同期を行わない方法もとったが計算量が多く同期に掛かる実行時間よりも削減することはできなかった．

また，関数 ssor で使用される reflect 指示文によって更新を行われる配列は，パイプライン処理に入る前に袖領域を上端・下端にそれぞれ 2 領域必要とする．しかし，パイプライン処理では，袖領域の上端 1 領域のみを必要とする．shadow 指示文によって確保された袖領域は，実行途中で領域の量を変えることができず，また，reflect 指示文は，現在の実装では部分的に袖領域の更新を行うことができないため，無駄な更新が多くなってしまっている．

これらのことから，XcalableMP による実装で良い結果が得ることができなかった．XcalableMP の仕様書によると gmove 指示文の非同期化と reflect 指示文による袖領域の部分的な更新については書かれているため，今後これらが実装されれば MPI 版に近い結果を得ることができると考える．

6. まとめ

本稿では，XcalableMP を用いて NPB の CFD アプリケーションである，BT，SP 及び LU の 3 種類について，グローバルビューモデルとローカルビューモデルを用いた 2 つの実装を行い，XcalableMP の生産性と性能の評価を行った．結果として，ローカルビューモデルによる実装は，NPB の MPI 版に対してほぼ同等の性能を得ることができた．また，グローバルビューモデルによる実装は，MPI 版の性能には及ばなかったが，仕様書に記載されている機能である gmove 指示文による通信の非同期化や，reflect 指示文による部分的な同期に加え，loop 指示文による袖領域を含めた範囲での並列計算を実現することが出来れば，MPI 版に近い結果を得られることがわかった．加えて，グローバルビューモデルによる実装では，MPI 版と比較してシリアル版のソースコードを大きく変更することなく実装をすることができたため，XcalableMP の生産性を示すことができた．

謝辞 本研究の一部は，JST-ANR 日仏戦略プロジェクト研究領域「情報通信技術」「ポストペタスケールコンピューティングのためのフレームワークとプログラミング」(FP3C) による．また，XcalableMP の仕様は，次世代並列プログラミング言語検討委員会によるものである．

参考文献

[1] XcalableMP Specication Working Group, version 1.1 : XcalableMP, http://www.xcalablemp.org/spec/xmp-spec-1.1.pdf (2012).
[2] Nakao, M., Lee, J., Boku, T. and Sato, M. : XcalableMP Implementation and Performance of NAS Parallel Benchmarks, Fourth Confer- ence on Partitioned Global Address Space Pro- gramming Model (PGAS10) (2010).
[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] Guohua Jin, Mellor-Crummey, J., Adhianto, L., Scherer, W.N. : Implementation and Performance Evaluation of the HPC Challenge Benchmarks in Coarray Fortran 2.0, Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International，pp. 1089-1100 (2011).
[5] A. Stone, J. M. Dennis, M. Strout : Evaluating Coarray Fortran with the CGPOP Miniapp，Proceedings of the Fifth Conference on Partitioned Global Address Space Programming Models (PGAS) (2011).
[6] 中尾 昌広，李 珍泌，朴 泰祐，佐藤 三久：XcalableMP による NAS Parallel Benchmarks の 実装と評価，情報処理学会研究報告書，Vol.2010-HPC-126，No. 9，pp. 1-7 (2010).
[7] Numwich, R. and Reid, J. : Co-Array Fortran for parallel programming, Technical report ral- tr-1998-060, Rutherford Appleton Laboratory (1998).
[8] 中尾 昌広，Tran Minh Tuan，李 珍泌，朴 泰祐，佐藤 三久：PGAS 言語 XcalableMP における coarray 機能の実装と評価，先進的計算基盤システムシンポジウム論文集，Vol. 41，No. 6，pp. 289 - 297 (2012).
[9] Bailey, D.H. and et al. : THE NAS PARALLEL BENCHMARKS，Technical Report NAS-94-007，Nasa Ames Research Center (1994).
[10] 村井 均，岡部 寿男：地球シミュレータ上の HPF による NAS Parallel Benchmarks の実装と評価，情報処理学会シンポジウム論文集，No. 6，pp. 389-369 (2004).