PGAS 言語 XcalableMP における coarray 機能の実装と評価

中尾昌広[†] Tran Minh Tuan[††,☆] 李珍泌[††,☆☆]
朴泰祐[†,††] 佐藤三久[†,††,†††]

XcalableMP は分散メモリ環境を対象とした PGAS モデルに基づくプログラミング言語である．XcalableMP では生産性を高めるために，片側通信を簡易に記述できる coarray 機能を提供している．本論文では C 言語版の XcalableMP における coarray 機能の実装を高速通信ライブラリ GASNet と ARMCI を用いて行った．XcalableMP の coarray 機能を用いて ping-pong プログラム，NAS Parallel Benchmarks の Integer Sort および Conjugate Gradient の実装を行い，それぞれの性能評価を行った．その結果，それぞれのベンチマークにおいて GASNet を用いた実装は，ARMCI を用いた実装よりも性能が高くなることを示した．一方，ARMCI は Accumulate 通信に直接対応しているため，Accumulate 通信が発生するアプリケーションにおいては，ARMCI を用いた実装は GASNet を用いた実装よりも性能が高いことがわかった．

Implementation and Performance Evaluation of Coarray Function in XcalableMP

Masahiro Nakao [,†] Tran Minh Tuan [,††] Jinpil Lee [,††,☆☆]
Taisuke Boku [†,††] and Mitsuhisa Sato[†,††,†††]

XcalableMP is a parallel programming language based on a PGAS model for distributed memory systems. To facilitate development of parallel applications, XcalableMP has a "coarray function" which provides programmers to use one-sided communication easily. In this paper, we implement the coarray function of C language version XcalableMP by using GASNet and ARMCI, which are high-performance communication layers. We program ping-pong, Integer Sort and Conjugate Gradient of NAS Parallel Benchmarks by using the coarray function. These results show that performances of implementation by using GASNet are higher than those by using ARMCI. Besides, in an application which needs communication for accumulation, performance of implementation by using ARMCI is higher than that by using GASNet because ARMCI supports the communication for accumulation directly.

1. はじめに

分散メモリ環境を対象とした並列プログラミングモデルに XcalableMP（XMP）[1)~3)] がある．XMP は Partitioned Global Address Space（PGAS）言語の一種であり，指示文を用いて高性能な並列アプリケーションを簡易に開発することが可能である．XMP には C 言語版と Fortran 言語版が存在し，それぞれで用いられる指示文は同一になるように考慮されている．

XMP では，グローバルビューモデルとローカルビューモデルという 2 つのメモリ抽象化モデルを提供している．グローバルビューモデルではプログラマは大域アドレス空間を用いることで，分散されたデータの配置を意識することなくデータの操作と並列処理を行うことができる．一方，ローカルビューモデルは Fortran 言語の並列拡張である Coarray Fortran[4)] をベースとしており，プログラマはノード番号を指定することで，異なるノードに配置されたデータを片側通信を用いて操作することが可能である．この機能を coarray 機能と呼ぶ．ローカルビューモデルはグローバルビューモデルと比較して，各ノードの振る舞いを詳細に記述するアプリケーションの開発に適している．

本論文の目的は，C 言語版の XMP のローカルビュー

† 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba
†† 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
☆ 現在，ANS, Inc.
☆☆ 現在，MACROGRAPH, Inc.
††† 理化学研究所 計算科学研究機構
RIKEN Advanced Institute for Computational Science

モデルを実現するために必要な coarray 機能の実装とその評価を行うことである．coarray 機能を高速通信ライブラリである Global-Address Space Networking (GASNet)[5),6)] と Aggregate Remote Memory Copy Interface（ARMCI)[7),8)] を用いて実装し，各通信ライブラリに適したアプリケーションについて考察する．

本論文の以降の構成は下記の通りである．2 章では関連研究をまとめ，3 章では XMP の coarray 機能の仕様および実装について述べる．4 章ではベンチマークプログラムを用いて XMP の coarray 機能における各通信ライブラリの性能について評価し，5 章では 4 章の結果について考察する．

2. 関連研究

片側通信は PGAS 言語で現れる通信パターンとよく合致するため，PGAS 言語で用いられることが多い．また，ハードウェアによる Remote Direct Memory Access（RDMA）の支援により，性能向上につながる場合もある[5)]．一般に並列アプリケーションの作成には C，C++，Fortran 言語と MPI ライブラリとの組合せが用いられることが多いが，MPI の仕様は非常にプリミティブな機能しか定義されていないため，文献 9) でも指摘されている通り，MPI ライブラリを PGAS 言語の通信レイヤとして用いることは難しい．

GASNet は University of California, Berkeley（UCB）と Lawrence Berkeley National Laboratory が開発した通信ライブラリであり，Berkeley Unified Parallel C[10)]，Rice Coarray Fortran[4),11)]，Titanium[12)]，Chapel[13)] など様々な PGAS 言語の通信レイヤとして用いられている．GASNet はポータビリティと高性能性を両立させるため 2 つのレイヤで構成されている．ハードウェアに近い下位レイヤを Core API と呼び，UCB で開発された高速メッセージ通信ライブラリ Active Messages（AM）が用いられている．ユーザに近い上位レイヤを Extented API と呼び，多機能な片側通信のインタフェースが定義されている．

ARMCI は Pacific Northwest National Laboratory が開発した通信ライブラリであり，Global Arrays[14)] や GPSHMEM[15)]，また GASNet と同様に Rice Coarray Fortran の通信レイヤとして用いられている．ARMCI と GASNet の違いとして，ARMCI では計算とは別に通信用のサーバスレッドを起動させることで，不連続なデータ（stride，scatter/gather，I/O vector）の put/get/accumulate 操作に直接対応している点が挙げられる[16)]．一方，下位の通信レイヤについては明確に定義されていないため，GASNet と比較してポータビリティは低いと言える．

PGAS 言語に利用可能な他の通信ライブラリとしては，Global Address Space Programming Interface (GPI)[17)] や Telmem[18)] があるが，提案されて間もないため実際に利用された例は少ない．

PGAS 言語で用いられる片側通信ライブラリに関する論文には，文献 11)，18)，19) などがある．文献 11) はマルチプラットフォーム性を持つ Coarray Fortran の実装に関する論文であり，その通信レイヤとして GASNet と ARMCI が用いられている．ストライド転送については，ARMCI はパイプライン処理を行うことで AM を直接操作した GASNet よりも性能が高いことが紹介されているが，それ以外の比較については行われていない．なお，GASNet のストライド転送機能は次期バージョンから対応する予定であるため[6)]，ARMCI と同様の実装を行うことで GASNet の性能が文献 11) よりも向上する可能性は高いと考えられる．文献 18) では Telmem，MPI および ARMCI の性能比較が行われているが，GASNet との性能比較は行われていない．文献 19) は GASNet を通信レイヤに用いた UPC と Titanium に関するものであり，様々な計算機環境上で GASNet の基本性能を計測し，GASNet の方が MPI よりも優位であることを示している．

3. XcalableMP における coarray 機能の実装

3.1 coarray 機能の仕様

XMP の実行モデルは MPI と同様に Single Program Multiple Data（SPMD）である．XMP では，メモリを共有する 1 つ以上の CPU コアから構成される計算リソースの単位をノードと呼ぶ．XMP において，あるデータを任意のノードに転送したい場合，coarray 機能もしくは XMP 指示文を用いて明示的に指定する．

本節では，C 言語版の XMP の coarray 機能の仕様を XMP の仕様書 version 1.0[1)] を基にまとめる．coarray 機能を用いて他のノードに存在する配列を参照するための書式を下記に示す．このような書式で参照できる配列を coarray と呼ぶ．下記は 1 次元配列の例であるが，多次元配列にも対応している．

array-name[*base*:*length*:*step*]:[*node-index*]
array-name[*base*:*length*]:[*node-index*]
array-name[:]:[*node-index*]

通常の配列とコロンの後に，角括弧と ***node-index*** を用いて参照先のノード番号を指定する．また，参照される要素の範囲もコロンを用いて表す．***base*** は開始要素番号を示し，***length*** は参照される要素数を示す．また，***step*** は参照される要素のステップ幅を示す．2 行目のように ***step*** が省略された場合は，連続データが参照される．3 行目のようにコロンのみの場合は，すべての要素が参照されることを示す．

また，coarray 機能に対する同期機構として，***sync_memory*** 指示文が定義されている．***sync_memory*** 指示文はローカルで実行される同期命令であり，***sync_memory*** 指示文の前までに発行された coarray に対する全ての変更が完了することを保証する．

3.2 通信ライブラリに求められる機能

本節では，3.1 節でまとめた XMP の coarray 機能に加え，XMP に対して有用な機能についても言及し，それらを実現するために必要な通信ライブラリの機能について述べる．

- put/get の片側通信操作およびストライド通信
  これらに関連する機能として，現時点では XMP ではメモリコンシステンシモデルについては規定されていないが，将来的な拡張を考慮すると，片側通信操作はブロッキング/ノンブロッキング通信のどちらにも対応可能であることが望ましい．
- ***sync_memory*** 指示文のための同期機構
  sync_memory 指示文を実装するためには，送り先にデータが届き，そのデータが利用できる状態になったことを調べる仕組みが通信ライブラリに必要である．なお文献 3) では，XMP の coarray 機能を MPI で定義されている片側通信を用いて実装を行ったとあるが，***sync_memory*** 指示文は最新の仕様書 version 1.0 から定義された指示文であり，この機能は MPI が提供する片側通信の同期機構では実装することはできない．

以上の項目が必須要件である．これらに加え必須ではないが，XMP に対して有用である通信ライブラリに対する機能について下記に述べる．

- put 方向の Accumulate 通信
  例えば下記のコードのような，put 方向の Accumulate 操作を伴う通信パターンに対応していることが望ましい．下記コードの配列 *a[]* は coarray，配列 *b[]* は通常の配列，変数 *me* はノード番号を示している．

```
if(me == 1) a[:]:[2] += b[:];
if(me == 1) a[:]:[2] *= 5;
```

  ただし，通常の put/get 通信のみでも実装は可能である．上記の場合，まずノード番号 1 はノード番号 2 が持つ配列 *a[]* を get し，ローカルで演算を行った後，ノード番号 2 に put すればよい．しかしながら，通信を 2 回行う必要があるため非効率な実装となる．本項目については 5.2 節で考察する．
- 通信性能
  ネットワークアーキテクチャの性能を最大限に利用できる通信ライブラリが望ましい．また，高性能クラスタで利用されているマルチレールにも対応していることが望ましい．
- ポータビリティ
  XMP で作成したアプリケーションのポータビリティは，通信ライブラリに大きく依存するため，通信ライブラリのポータビリティは非常に重要である．

3.3 coarray 機能の実装

3.2 節で述べた必須要件を満たし，かつ多くの PGAS 言語において動作実績を持つ通信ライブラリ GASNet と ARMCI を用いて XMP の coarray 機能の実装を行う．なお GASNet と ARMCI は，Infiniband，Myrinet，MPI，UDP，IBM 社の BlueGene/P や Cray 社の XE/XT など，現在多くの並列分散環境で用いられる通信レイヤに対してサポートが行われている．現時点で最新である GASNet-1.18.0 と Global Arrays 5.0.2 と共に配布されている ARMCI における 3.2 節で述べた要件に対する対応を**表 1** に示す．2 章で述べた通り，GASNet のストライド通信機能は GASNet の次期バージョンで対応する予定になっている．また，GASNet は Accumulate 通信に直接の対応はしていないが，下位レイヤで用いられている AM のハンドラを定義することによって対応可能である．

筑波大学で開発が行われている Omni XMP Compiler[3] に対して，coarray 機能の実装を行った．Omni XMP Compiler は，source-to-source のコンパイラであり，ベース言語（C または Fortran）と XMP 指示文で記述されたソースコードを XMP コンパイラが生成する実行時ルーチンとして呼び出される関数の呼び出しに変換した後，ネイティブコンパイラ（例えば mpicc）を用いて実行コードに変換する．なお，coarray 機能については，ユーザが XMP プログラムをコンパイルする際，オプションを用いることで通信レイヤを切り替えることが可能になるように実装を行う．この機能により，ユーザは作成するアプリケーション

```
int a[1024][1024], b[1024][1024];
#pragma xmp coarray a:[3][3]
if (me == 3){ b[p:q][r:s:step] = a[t:q][u:s:step]:[1][2]; }
```

↓

```
if(me== (3)) {
_XMP_coarray_rma_ARRAY(700, _XMP_COARRAY_DESC_a, &(b),      // basic information
        (int)(2),                                           // number of coarray dimensions of a[][]
        (int)(t), (int)(q-t+1), (int)(1), (unsigned long long)(0x000000400ll),   // information for 1st dimension of a[][]
        (int)(u), (int)(s-u+1), (int)(step), (unsigned long long)(1),            // information for 2nd dimension of a[][]
        (int)(2),                                           // number of coarray dimensions of b[][]
        (int)(p), (int)(q-p+1), (int)(1), (unsigned long long)(0x000000400ll),   // information for 1st dimension of b[][]
        (int)(r), (int)(s-r+1), (int)(step), (unsigned long long)(1),            // information for 2nd dimension of b[][]
        (char)(NULL), (void *)(NULL),                       // information for accumulation
        (int)(1), (int)(2));                                // information for node
}
```

図 1 coarray 操作の変換例
Fig. 1 Example of translation of coarray operation

表 1 XcalableMP の coarray 機能の要件に対する GASNet と ARMCI のサポート
Table 1 Support of GASNet and ARMCI for functional requirements of XcalableMP coarray function

	GASNet	ARMCI
Blocking put/get	○	○
Non-blocking put/get	○	○
Stride communication	work-in-progress	○
Synchronism system for ***sync_memory*** directive	○	○
Communication for accumulation	△	○
Support for multi-rail	○	×
Portability	○	△

に適した通信レイヤを用いることが可能になる．

XMP のソースコードに現れる coarray は，下記の 4 つに分類できる．下記コードの配列 *a[]* は coarray，変数 *b, i, j* は通常の変数である．

```
b = a[i]:[j];      // get
a[i]:[j] = b;      // put
b += a[i]:[j];     // get-accumulate
a[i]:[j] += b;     // put-accumulate
```

XMP における 4 種類の coarray 機能を呼び出すための関数を作成する．**図 1** に XMP のソースコード中の coarray を関数（_XMP_coarray_rma_ARRAY）に変換する例を示す．その関数のプロトタイプ宣言は下記の通りである．

```
void _XMP_coarray_rma_ARRAY(int rma_code,
XMP_coarray_t *coarray, void *rma_addr, ...);
```

第 1 引数 *rma_code* が 700 の時は get，701 の時は put，702 のときは get 方向の Accumulate，703 のときは put 方向の Accumulate の操作を表す．第 2 引数 *coarray* は coarray の descriptor であり，配列の実体へのポインタや 1 要素のバイト数などの情報が格納されている．第 3 引数 *rma_addr* はローカル変数の先頭アドレスである．以降の引数は，coarray の次元数，右辺の coarray の 1 次元目の開始要素，終了要素，ステップ幅，要素間のメモリの距離を示している．以降は各次元の情報および左辺の情報である．その次は Accumulate が行われる場合必要となる情報であり，最初は四則演算を表す演算子，次は計算される値のポインタである．なお Accumulate 操作の右辺は，coarray を含む配列もしくは 1 要素からなる変数のみを想定している．最後の 2 つの引数は，通信先のノード番号を示している．

関数_XMP_coarray_rma_ARRAY の内部では，コンパイル時にユーザが指定した通信ライブラリが呼び出される．今回の実装では，put 操作はノンブロッキング通信，get 操作はブロッキング通信の GASNet および ARMCI の通信 API を用いて実装を行った．

なお，get 方向の Accumulate 通信の実装は非常に容易である．まずローカルに一時領域を作成し，その領域に get して得た送信元のデータ *a[i]:[j]* を一時保存し，そのデータとローカル変数 *b* とを演算させるという実装になる．

また，仕様書には記載はないが，今回の実装における coarray の制約として，関数の引数として用いることや，右辺で複数の演算を同時に行うことができないことが存在する．

4. 性能評価

各通信レイヤの基本性能およびその特性を調べるため，XMP を用いて ping-pong プログラムおよび NAS Parallel Benchmarks（NPB）[20] の Integer Sort（IS）と Conjugate Gradient（CG）を実装し，性能評価を行う．なお，IS と CG は Omni Compiler Project が

表 2 実験環境
Table 2 Experimental environment

CPU	AMD Opteron Quad-Core 8356 2.3GHz (4 sockets)
Memory	DDR2 667MHz 32GB
Network	Infiniband DDR(4rails) 4x2GB/s
OS	Linux 2.6.18
Compiler	gcc 4.6.0 Omni XcalableMP Compier 0.5.3
Communication Library	mvapich2-1.7a, GASNet-1.18.0 ARMCI(in Global Arrays 5.0.2)

提供している C 言語版の NPB version2.3[21] をベースに作成した．実験に用いた問題サイズは共に Class C である．

実験には T2K Tsukuba System を用いた（**表 2**）．T2K Tsukuba System は 1 ノードにつき 16CPU コアを持つが，今回の実験ではネットワークの性能に重点を置いて計測したいため，1 ノードにつき 1 プロセスを割り当てた．また，T2K Tsukuba System のネットワークは 4 レールで利用可能であるが，ARMCI はマルチレールに対応していないため，すべての実験は 1 レールのみを利用している．実験は 5 試行行い，その最大値で評価する．

4.1 ping-pong ベンチマーク

基本的な通信性能を評価するため，ping-pong プログラムを XMP で実装し，その性能評価を行う．**図 2** に put の ping-pong プログラムを，**図 3** に get の ping-pong プログラムを示す．図 3 では ***coarray*** 指示文の宣言などは図 2 と同じなので，get 操作を行う箇所のみ記述している．

図 2 では，まず ***nodes*** 指示文により 2 ノードを計算に用いることを宣言する．次に put 操作により相手（変数 *dest* は相手のノード番号を示す）にデータを送り，自ノードは相手からデータが送られてくるまで busy wait で待つ動作を繰り返す．図 3 では，get 操作により相手にデータを送っている．get の場合，ブロッキング通信であるので busy wait の代わりにバリア同期を行う ***barrier*** 指示文を用いている．図 2 と図 3 中の *LOOP* は繰り返しの回数であり，今回の実験では 1000 に設定した．変数 *size* は int 型の整数であり，送信する要素数を設定する．

結果を**図 4** に示す．put/get のどちらにおいても GASNet を用いた実装が優れていることがわかる．なお，一般的に get よりも put の方が性能は高いが，ARMCI を用いた実装では転送データサイズによっては get の方が性能が高くなっている．この原因は，ARMCI が提供しているバリア同期の性能が低いからであり，put 自体の転送能力は get より 30%程度性能が高いことを確認している．

```
#pragma xmp nodes p(2)    // 2 nodes are used
char loc_buf[MAX], rmt_buf[MAX];
#pragma xmp coarray rmt_buf:[*]
   :
memset(loc_buf, '0' , MAX);
memset(rmt_buf, '1' , MAX);
for(i=0;i<LOOP;i++){
  if(me == 1){
    rmt_buf[0:size]:[dest] = loc_buf[0:size];
#pragma xmp sync_memory
    while (rmt_buf[size-1] != '0') {
#pragma xmp sync_memory
    }
    rmt_buf[size-1] = '1';
  } else {
    while (rmt_buf[size-1] != '0') {
#pragma xmp sync_memory
    }
    rmt_buf[size-1] = '1';
    rmt_buf[0:size]:[dest] = loc_buf[0:size];
#pragma xmp sync_memory
  }
}
```

図 2 ping-pong（put）のプログラム例
Fig. 2 Example of ping-ping(put)

```
for(i=0;i<LOOP;i++){
  if(me == 1){
    loc_buf[0:size] = rmt_buf[0:size]:[dest] ;
#pragma xmp barrier
  } else {
#pragma xmp barrier
    loc_buf[0:size] = rmt_buf[0:size]:[dest] ;
  }
#pragma xmp barrier
}
```

図 3 ping-pong（get）のプログラム例
Fig. 3 Example of ping-ping(get)

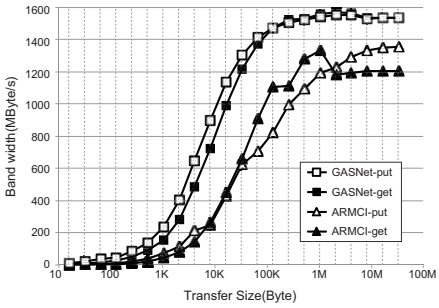

図 4 ping-pong の結果
Fig. 4 Result of ping-ping

4.2 Integer Sort ベンチマーク

IS は並列化バケツソートの実装であり，各プロセスでソートされた要素を交換する際に Alltoallv 型の通信が発生する．**図 5** にソースコードの一部を示す．オリジナルの IS で用いられている関数 MPI_Alltoallv の箇所を put 操作を用いて実装している．注意点として，put 操作の前に ***gmove*** 指示文を用いて put 操作

```
#pragma xmp coarray key:[*]
  :
#pragma xmp gmove
   dst_displ[:][:] = rcv_displ[:][:];

for( i=0; i<comm_size; i++ ){
   key[dst_displ[i][me]:count[i]]:[i] = loc[src_displ[i]]:count[i]];
}
_xmpc_sync_all();
```

図 5 XcalableMP による Integer Sort プログラム
Fig. 5 Integer Sort by using XcalableMP

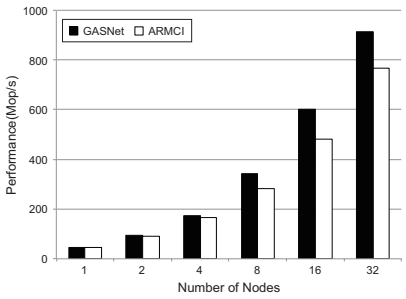

図 6 Integer Sort の結果
Fig. 6 Result of Integer Sort

を行うために必要な転送先の保存場所を転送先から取得する必要がある．この操作はオリジナルの IS では必要のない操作である[22]．

結果を**図 6** に示す．図 6 より，GASNet を用いた実装は ARMCI を用いた実装よりも性能が高いことがわかる．

4.3 Conjugate Gradient ベンチマーク

CG は疎行列の最小固有値の近似値を共役勾配法を用いて解くアルゴリズムである．文献 22) では，XMP のグローバルビューモデルを用いた実装について記されており，その問題点として，計算に用いるプロセス数によってはオリジナルの CG と比較して通信量が増えることも示されている．CG をローカルビューモデルで作成することにより，オリジナルの CG と同じ通信量のアルゴリズムを実装できる．Unified Parallel C で作成された CG[23] を参考にして XMP で CG を実装した．**図 7** にソースコードの一部を示す．

各ノードは他のノードが持つ *w[]* の値を配列 *q[]* に加算集約している．この通信パターンはオリジナルの CG では MPI の双方向通信を用いて実装されているが，XMP の CG では get 操作を用いて実装している．

結果を**図 8** に示す．図 8 より，GASNet と ARMCI を用いた実装はほぼ同じ性能であることがわかる．

```
#pragma xmp nodes pros(NUM_COLS, NUM_ROWS)
#pragma xmp coarray w:[*]
#pragma xmp coarray w1:[*]
  :
for( i=(l2npcols-1); i>=0; i-- ){
      :
  w[l:count[k][i]] += w1[m:count[k][i]]:[k];
#pragma xmp barrier
  if(i != 0){
     memcpy( &w1[l], &w[l], sizeof(double)*(count[k][i]));
#pragma xmp barrier
  }
}

if( l2npcols == 0 )
  memcpy( &q[0], &w[0], sizeof( double )*length[me] );
else
  q[0:length[dest]] = w[start[dest]:length[dest]]:[dest];
  :
```

図 7 XcalableMP による Conjugate Gradient プログラム
Fig. 7 Conjugate Gradient by using XcalableMP

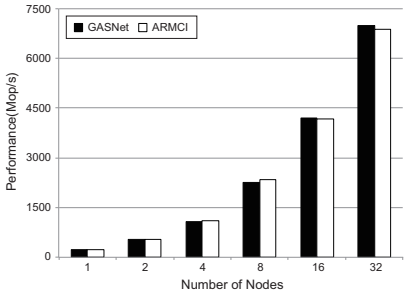

図 8 Conjugate Gradient の結果
Fig. 8 Result of Conjugate Gradient

表 3 全体の処理時間中の通信時間の割合（%）
Table 3 Percentage of communication time in the overall processing time(%)

Nodes	Integer Sort		Conjugate Gradient	
	GASNet	ARMCI	GASNet	ARMCI
2	12.17	12.76	0.91	1.00
4	19.43	21.15	2.19	2.60
8	21.08	32.95	5.11	5.76
16	31.01	42.87	9.94	10.98
32	49.01	55.55	20.34	21.46

5. 考　　察

5.1 Integer Sort と Conjugate Gradient についての性能詳細

本節では，前章で行った IS と CG の実験結果の詳細を調べ，GASNet と ARMCI の coarray 機能の通信性能について考察する．ノード間通信が発生する 2 ノード以上を計算に用いた各ベンチマークの通信時間を**図 9** と**図 10** に示し，各ベンチマークにおける全体の処理時間中の通信時間の割合を結果を**表 3** に示す．

図 9 と図 10 の結果から，すべての結果において

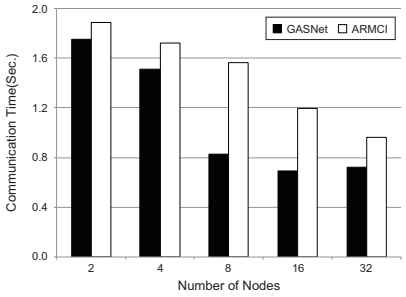

図 9 Integer Sort における通信時間
Fig. 9 Communication Time in Integer Sort

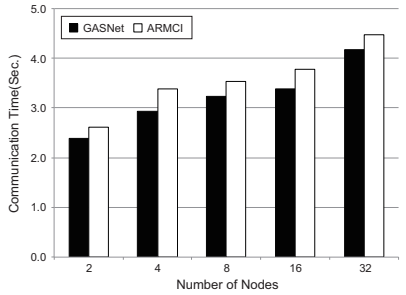

図 10 Conjugate Gradient における通信時間
Fig. 10 Communication Time in Conjugate Gradient

GASNet を用いた実装の方が ARMCI を用いた実装よりも少ない通信時間でベンチマークを実行できることがわかる．また表 3 の結果から，IS の方が CG よりも通信性能がベンチマークの性能に大きく関係していることがわかる．そのため，IS の性能結果において，GASNet を用いた実装の方が ARMCI を用いた実装よりも性能が高くなったと考えられる．CG の性能結果においては，今回の計算環境では通信が全体時間に占める割合が小さいため，GASNet を用いた実装と ARMCI を用いた実装との間に大きな性能差は確認できなかったが，表 3 の結果から，計算に用いるノード数が増えると GASNet を用いた実装の方が性能が高くなると考えられる．

5.2 Accumulate 通信の性能評価

片側通信を用いる場合，一般に get 通信よりも put 通信の方が性能が高いため，どちらを用いてもアプリケーション開発が行える場合は，put 操作がよく利用される．しかし，put 方向で Accumulate 通信が発生する場合は，3.2 節で述べたように通信ライブラリが Accumulate 通信に直接対応していなければ，非効率な実装となる．

本節では，Accumulate 通信が現れるアプリケーションに対する性能評価を行う．ベンチマークプログラムには，get 操作で記述した 4.3 節の CG ベンチマーク

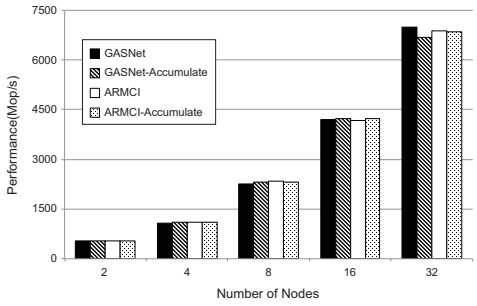

図 11 Accumulate 通信を用いた場合の CG の結果
Fig. 11 Result of Conjugate Gradient with communication for accumulation

を，put および Accumulate 通信で実行するように変更したプログラムを用いる．なお，図 7 では ***barrier*** 指示文を用いて同期を行っているが，変更したプログラムでは ***barrier*** 指示文の代わりに ***sync_memory*** 指示文を用いている．ノード間通信が発生する 2 ノード以上について，性能評価を行った．

結果を**図 11** に示す．図 11 中の項目である GASNet および ARMCI は図 8 の値と同じであり，GASNet-Accumulate および ARMCI-Accumulate は変更を行ったプログラムの性能値である．この結果より，32 ノードを用いた場合，GASNet を用いた実装は Accumulate の通信パターンを利用すると性能が 5%程度低くなることがわかる．一方，ARMCI を用いた実装では Accumulate 通信を用いても性能はほとんど変わらないことがわかる．また，Accumulate 通信同士の比較では，ARMCI を用いた実装の方が GASNet を用いた実装よりも性能は高くなることがわかる．

6. ま と め

本論文では，C 言語版の XMP のローカルビューモデルを実現する上で必要となる coarray 機能の仕様についてまとめ，その実装方法について説明した．そして，高速通信ライブラリである GASNet と ARMCI を用いて coarray 機能の実装を行い，ベンチマークを用いてそれぞれの性能評価を行った．

ping-pong プログラムおよび IS ベンチマークの性能評価を行った結果，GASNet を用いた実装の方が ARMCI を用いた実装よりも性能は高いことがわかった．CG の結果については，GASNet および ARMCI を用いた実装はほぼ同等の性能を示したが，計算に用いるノード数が増えた場合，GASNet を用いた実装の方が性能は高くなると考えられることを示した．次に Accumulate 通信が必要となるアプリケーションの性能を評価するため，Accumulate 通信を用いた CG

の作成を行った．性能測定を行った結果，ARMCI は Accumulate 通信に直接対応しているため，ARMCI を用いた実装の方が GASNet を用いた実装よりも性能は高くなることを示した．

結論として，基本的にはユーザは GASNet を XMP の coarray 機能の通信レイヤとして選択する方が良いと言える．しかし，Accumulate 通信や不連続なデータ操作が多いアプリケーションを作成する場合は，ARMCI を選択した方が良いと言える．

今後の課題として，GASNet の AM を直接用いた Accumulate 通信および次期 GASNet で対応予定のストライド通信を用いた実装，実アプリケーションを用いた coarray 機能の性能評価を行うことなどが挙げられる．

謝辞 本研究の一部は，文部科学省「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発，高生産・高性能計算機環境実現のための研究開発，シームレス高生産・高性能プログラミング環境」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 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.: XcalableMP Implementation and Performance of NAS Parallel Benchmarks, Fourth Conference on Partitioned Global Address Space Programming 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) Numwich, R. and Reid, J.: Co-Array Fortran for parallel programming, Technical report ral-tr-1998-060, Rutherford Appleton Laboratory (1998).
5) Bell, C., Bonachea, D., Nishtala, R. and Yelick, K.: Optimizing bandwidth limited problems using one-sided communication and overlap, In The 20th Int'l Parallel and Distributed Processing Symposium (IPDPS) (2006).
6) Dan Bonachea: GASNet Specification Version 1.8, http://gasnet.cs.berkeley.edu/dist/docs/gasnet.pdf (2008).
7) Nieplocha, J., Tipparaju, V., Krishnan, M. and Panda, D.: High Performance Remote Memory Access Comunications: The ARMCI Approach, *International Journal of High Performance Computing and Applications*, Vol. 20, No. 2, pp. 233–253 (2006).
8) Nieplocha, J. and Ju, J.: ARMCI: A Portable Aggregate Remote Memory Copy Interface, http://www.emsl.pnl.gov/docs/parsoft/armci/publications/armci1-1.pdf (2000).
9) Bonachea, D. and Duell, J.: Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations, In 2nd Workshop on Hardware/Software Support for High Performance Scientific and Engineering Computing, SHPSEC-PACT03 (2003).
10) Chen, W.-Y., Bonachea, D., Duell, J., Husbands, P., Iancu, C. and Yelick, K.: A Performance Analysis of the Berkeley UPC Compiler, ICS '03 Proceedings of the 17th annual international conference on Supercomputing (2003).
11) Katherine Yelick et. al.: Productivity and performance using partitioned global address space languages, PASCO '07, Proceedings of the 2007 international workshop on Parallel symbolic computation (2007).
12) Yelick, S. and et al: Titanium: A High-Performance Java Dialect, ACM 1998 Workshop on Java for High-Performance Network Computing, Stanford, California (1998).
13) Chamberlain, B. L., Callahan, D. and Zima, H.P.: Parallel Programmability and the Chapel Language, *International Journal of High Performance Computing Applications*, Vol. 21, No. 3, pp. 291–312 (2007).
14) Jarek, N.: Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit, *International Journal of High Performance Computing Applications*, Vol. 20, No. 2, pp. 203–231 (2006).
15) K. Parzyszek, J. Nieplocha, and R. A. Kendall: Generalized Portable SHMEM Library for High Performance Computing, Proceedings of the Twelfth IASTED International Conference Parallel and Distributed Computing and Systems, M. Guizani and X. Shen, Eds. Las Vegas, Nevada: IASTED (2000).
16) Buntinas, D., Saify, A., Panda, D.K. and Nieplocha, J.: Optimizing Synchronization Operations for Remote Memory Communication Systems, IPDPS ' 03 (2003).
17) Fraunhofer Institut for Industrial Mathematics ITWM: Global Address Space Programming Interface. http://www.gpi-site.com/cms/.
18) 堀敦史, 李珍泌, 佐藤三久: 片方向通信の実装方式の違いによる比較, 情報処理学会研究報告, Vol. 2010-HPC-126, No. 12, pp. 1–8 (2010).

19) Coarfa, C.: *Portable High Performance and Scalability of Partitioned Global Address Space Languages*, PhD Thesis, Rice Univ. (2007).
20) Bailey, D.H. et al: THE NAS PARALLEL BENCHMARKS, Technical Report NAS-94-007, Nasa Ames Research Center (1994).
21) Sato, M.: Download Omni OpenMP benchmarks. `http://www.hpcs.cs.tsukuba.ac.jp/omni-openmp/download/download-benchmarks.html`.
22) 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, Vol.2010-HPC-126, No. 9, pp. 1–7 (2010).
23) upc developers: UPC NAS Parallel Benchmarks. `http://threads.hpcl.gwu.edu/sites/npb-upc/`.