Vol.2015-HPC-148 No.22
2015/3/3

メニーコアプロセッサ向けプロセスモデルPVASのPGAS言語実行時ライブラリの設計

大川 千聡[1,a)]　堀 敦史[2]　島田 明男[2]　池井 満[1,3]　佐藤 三久[1,2]

概要：近年，これからの高性能計算システムのプロセッサ技術の一つとしてメニーコアプロセッサが注目されているが，メニーコアプロセッサではコア数が多いことからコアごとのデータの局所性を活用できるプログラミングモデルが望ましい．その一つとして，Partitioned Global Address Space (PGAS) プログラミングモデルが提案されている．我々は新たなモデルである Partitioned Virtual Address Space(PVAS) を用いて，PGAS 言語である XcalableMP の実行時ライブラリをメニーコアプロセッサ向けに設計，実装した．PVAS は，プロセスを同じアドレス空間に作成することができ，直接，異なるプロセスのデータにアクセスすることができることから，効率がよい通信を行うことが期待される．内部的に使用されていた MPI の代わりに PVAS を用いることで通信時間を抑えることができた．ステンシル計算で使用される reflect 通信では，通信時間を 45%に抑えることができ，分散配列の通信に使用される gmove 通信では，通信時間を 34%に抑えることができた．

1. はじめに

計算クラスタでは，プログラミングモデルの MPI が標準的に用いられている．しかし，MPI はプログラミングコストが高いことが問題となっている．そこで，仮想的にグローバルな配列を分散システムに対応させることで，逐次プログラミングと同様に記述することができる Partitioned Global Address Space(PGAS) 言語 [1] が注目されている．PGAS 言語の 1 つとして，XcalableMP(XMP) が提案されている [2][3]．逐次コードに指示文を追加することで，並列化することができる．指示文によって MPI と同等の分散が可能となり，プログラミングコストを減少させることができるとされている．

高性能計算分野において，新たなアーキテクチャのメニーコアプロセッサが注目されている．メニーコアプロセッサは，通常のプロセッサに比べ多くのコアを搭載することで，高い処理能力を実現している．しかし，多くのコアを持つために，従来よりもプロセッサ内の並列化が重要となる．そのため，コアごとのデータ局所性を利用することや効率的な通信方法が必要である．これらを解決するために，メニーコアプロセッサ向けのプロセスモデルである，PartitionedVirtualAddress Space(PVAS) が提案されている [4][5]．

以上を踏まえ，メニーコアプロセッサに適したプロセスモデルを用いた PGAS 言語実行時ライブラリの設計を行うことで，高い通信性能を得ることを目的とする．また，PGAS 言語を用いることで，クラスタを使用するときの MPI/OpenMP のようなハイブリッド並列化を一元的に扱うことができる．本稿では，袖通信と分散配列の通信を設計の対象とする．PGAS 言語として XMP，メニーコアプロセッサとして Intel 社が開発している Intel Many Integrated Core(MIC)，MIC 内プロセスモデルとして PVAS を使用する．本稿では，姫野ベンチマークと NAS Parallel Benchmarks CG(NPB-CG) の性能測定をおこなった．

2 章ではメニーコアアーキテクチャの MIC の概要，3 章では PGAS 言語の 1 つの XMP の概要，4 章ではメニーコア向けの PGAS 言語で用いる実行時ルーチンの候補を述べる．5 章では MIC 向け PGAS 言語の設計を行い，6 章で性能評価，7 章でまとめを行う．

2. Intel Many Integrated Core アーキテクチャの概要

MIC は Intel 社によって開発されている，Intel Xeon Phi Coprocessor を代表とするメニーコアアーキテクチャであ

[1] 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
[2] 理化学研究所 計算科学研究機構
Advanced Institute for Computational Science, RIKEN
[3] インテル株式会社
Intel K.K.
[a)] ohkawa@hpcs.cs.tsukuba.ac.jp

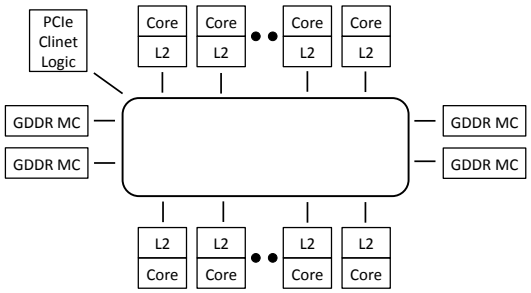

図 1 MIC の構造

り [6]，構造を**図 1** に示す．MIC は，Intel Xeon プロセッサのホストマシンに，PCI Express のボードとして使用する．Intel Xeon Phi Coprocessor 7120 には，61 個のコアがリングバスに接続されており，各コアには L2 キャッシュが用意されている．GDDR5 のメモリ 16GB がパッケージに搭載されており，2 チャネルのメモリコントローラ 8 個がリングバスに接続されている．また，リングバスに PCI Express ロジックが接続されており，MIC 外との通信はそれによって行われる．MIC の動作周波数は 1.3GHz に設定されており，比較的周波数の低いコアを多く搭載する．コア数が多く，メモリアクセスがボトルネックになることを防ぐため，メモリ帯域幅が 352GB/s と大きく設定されている．

3. PGAS 言語 XcalableMP

XMP は PC クラスタコンソーシアムで規格され，リファレンスコンパイラの Omni XcalableMP Compiler[3] が筑波大学と理化学研究所計算科学研究機構により実装されている．XMP は分散メモリ環境を対象とし，逐次アプリケーションに指示文を加えることで並列化を行うことができる並列言語である [2][3]．XMP は MPI と同様に SPMD モデルであり，各ノードで同一のプログラムが実行される．XMP 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存在する．グローバルビューモデルは，PGAS 言語に則しており，複数ノードに跨る配列データを宣言し，各ノードが処理する範囲を指示文によって定義する機能を提供する．ローカルビューモデルは各ノードが保持するローカルなデータを，配列参照に似た記述で通信することができる機能を提供する．本研究では，グローバルビューを使用しているため，グローバルビューについてのみ説明する．

3.1 グローバルビューモデル

グローバルビューモデルは，指示文で指定したデータを各ノードに分割して配置することができる．また，各ノードでの通信を指示文により行うことができる．**図 2** に XMP のソースコードの例を示す．nodes 指示文ではノード集合の名前と形状を指定することができ，p という名前の 2 次元の 2×2 ノード集合を定義している．template 指示文は，仮想的な index の集合を定義するものである．distribute 指示文では，template 指示文で指定した index をどのようにノードに配置するかを指定する．この場合，block を指定しているため，template の 100×100 要素を 4 等分した，50×50 要素が 1 ノードの割り当てとなる．distribute 指示文の後に，通常の C 言語の 100 行 100 列の 2 次元配列 A を宣言しているが，これはノード間においてグローバルな変数であるため，各ノードに全ての要素は確保されない．次の align 指示文で，template t によって A を分割するように定義されるため，各ノードには，50 行 50 列の配列が確保される．配列 B は行方向において分割され，列方向は*が指定されているため，重複した配列が確保される．したがって，p(1,1) と p(1,2) が B[0] から B[49] を持ち，p(2,1) と p(2,2) が B[50] から B[99] を持つ．

```
#pragma xmp nodes p(2, 2)
#pragma xmp template t(0:99, 0:99)
#pragma xmp distribute t(block, block) onto p

double A[100][100];
double B[100];
#pragma xmp align A[i][j] with t(j, i)
#pragma xmp align B[i] with t(i,*)
```

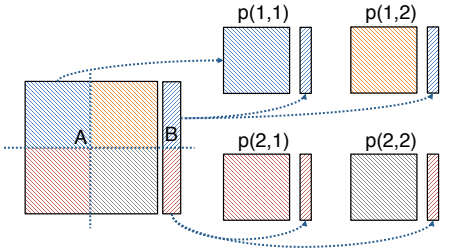

図 2 グローバルビューモデルのデータ並列化

3.2 reflect 指示文による通信

科学技術計算のステンシル型のアプリケーションなど，隣接ノードが保持する領域のデータを使用することで，各ノードの計算を行うアプリケーションは多く存在する．このため，隣接ノードのデータを重複する形で保持し，参照する前に値を更新する方法が取られている．重複する部分を袖領域と呼び，袖領域の同期を袖通信と呼ぶ．XMP では，袖領域の定義には，shadow 指示文を使用し，袖通信には reflect 指示文を使用する．図 2 のプログラムに，shadow 指示文の#pragma xmp shadow A[1][1] を追加すると，**図 3** のように，各ノードに分割された配列の上端・下端に 1 行 1 列分の袖領域が確保される．

袖領域の交換は reflect 指示文によって行われ，図 3 のように，隣接ノードに袖領域を送受信する．送信側のデータは pack によって送信側のバッファに蓄えられる．送信側のバッファの内容は，MPI 通信によって受信側のバッファに送信され，unpack によって袖領域に配置される．

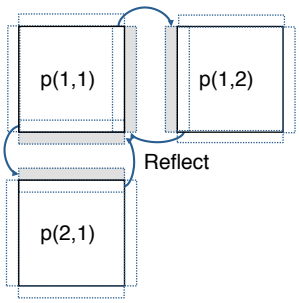

図 3 shadow，reflect 指示分による 2 次元袖通信

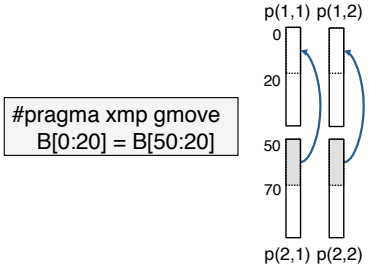

図 4 gmove 通信による分散配列の代入

3.3 gmove 指示文による通信

gmove 指示文では，B[0:20] = B[50:20]; のような代入文を記述することで，分散配置されたデータの通信を行うことができる．コロンの左側は，参照するグローバル配列のインデックス，右側は代入する要素数を表す．ユーザは B[50:20] がどのノードに配置されたデータであるかを意識する必要がなく，XMP コンパイラが内部的に判断する．

4. メニーコア向け PGAS 言語実行時ルーチンの検討モデル

メニーコアを対象とした PGAS 言語において，どのような実行時ルーチンを設計するかが重要となる．以下にその候補を示す．

(1) MPI 通信を行う方法．これは，ノード間に使用する MPI を MIC 内でも使用し，MIC の 1 コア (1 スレッド) を 1MPI プロセスと見なすことであり，実行時のオプションを変更するだけで実行可能となる．しかし，同じ共有メモリプロセッサでプロセス同士がメッセージ通信を使用することは無駄が多い．MPI 実装の 1 つである，OpenMPI では，MPI の通信に共有メモリを使用することができる．しかし，共有メモリを介した memcpy が必要となり，スレッド数の多い MIC にとっては，性能低下の原因となる．

(2) PGAS 言語が直接共有メモリを使用して通信する方法．PGAS 言語が共有メモリ領域を確保し，これを用いたデータ通信を行うことで，MPI を使用するよりも柔軟かつ高速なデータ転送が可能となる．この手法は池井らが実装している [7]．池井らの研究では，XMP の袖通信において，通常各 XMP プロセスで作成される 2 つの pack バッファ，unpack バッファを共有領域に 1 種類作成する．送信プロセスは memcpy によって，pack バッファに書き込み，受信プロセスは pack バッファからデータを memcpy によって読み込む．これらは pack/unpack と同等の処理であるため，MPI 通信を削減することができ，通信性能が向上した．

(3) PGAS 言語がコンパイラによってコードをスレッド並列に変換する方法．通常のスレッド並列であれば，参照データがスレッド間で共有されるため通信は必要ない．しかし，PGAS 言語に則したデータ分散の構造を持たない．そこで，PGAS 言語のコンパイラによる変換により，各スレッドでローカルなデータを確保させることで，データの局所性を高めることが期待できる．

(4) メニーコアに適したプロセスモデルを使用する方法．メニーコア向けの PVAS は，同じアドレス空間に複数のプロセスが存在するため，相互のデータを直接参照することができる．したがって，参照先のアドレスが判明していれば，memcpy1 回によりデータを取得することができる．

以上が挙げられる．データを局所的に持つことができることと，将来的に MPI によってクラスタで実行されノード内でも MPI プロセスを使用できることを考慮し，4 番目の PVAS を使用する手法を取る．

PVAS は理化学研究所計算科学研究機構で開発された，メニーコアを対象としたプロセスモデルである [4][5]．通常のプロセスはノード内であっても独立のアドレス空間を持つため，プロセス間では，互いのアドレス空間に存在するデータに直接アクセスすることができない．

一方，PVAS では**図 5** のように，複数のプロセスを単一のアドレス空間に生成する．PVAS アドレス空間に生成されるプロセスは，PVAS Task と呼ばれる．プロセスが単一のアドレス空間に存在するため，それぞれのプロセスの仮想アドレス空間が同一であり，仮想アドレスによってデータにアクセスすることができる．したがって，データの送受信は，受信側が送信側のデータをコピーすることで実現でき，データコピーを 1 回に抑えることができる．これにより，通信時間の減少が期待される．

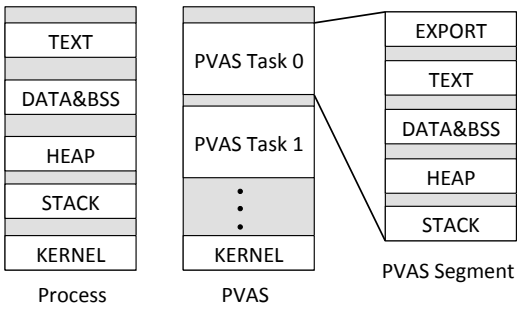

図 5 PVAS アドレス空間の構造

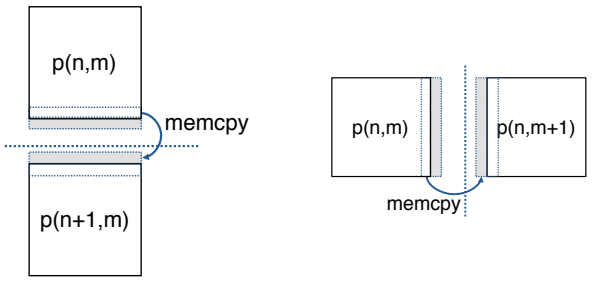

図 6 reflect 通信の処理

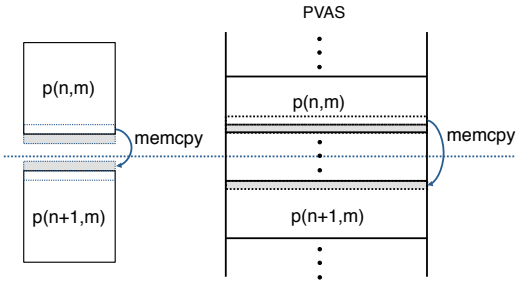

図 7 PVAS を用いた袖通信

表 1 実験環境

MIC	Intel Xeon Phi Coprocessor 7120
MPSS	2.6.38.8-pvas+mpss3.2.3
Compiler	Intel Compiler 13.1.3
MPI	OpenMPI 1.8
	Intel MPI 4.1

5. PVAS を用いた XMP ランタイムライブラリの設計

本稿では，Omni XcalableMP Compiler を使用した．このコンパイラは，C 言語に XMP 指示文を挿入したプログラムを，実行時ルーチンを呼び出す並列プログラムに変換する．本稿における設計では，MIC 内のプロセスモデルに PVAS を用いる．便宜上，XMP プロセスの作成などに MPI を使用する必要があり，OpenMPI[8] を使用した．

5.1 PVAS を用いた reflect 通信

現在の XMP は，すべてのプロセス間通信において MPI を使用している．XMP の reflect 機能では，あらかじめ shadow 指示文で指定された袖領域における送受信の設定を MPI_Send_init，MPI_Recv_init による persistent 通信によって行う．そして，reflect 指示文が呼び出されるたびに，**図 6** のように袖領域の pack，MPI 通信，unpack が実行される．

実際に設計と実装を行った袖通信を**図 7** に示す．図 7 のように，送信側プロセスにおける袖領域のアドレスから，受信側プロセスにおける袖領域のアドレスに memcpy を行うように設計した．PVAS を用いることで，送信側プロセス，受信側プロセスはアドレス空間を共有するため，直接データをコピーすることが可能である．memcpy には，参照先の仮想メモリアドレスが必要となる．このため，XMP で行われていた MPI_Send_init，MPI_Recv_init による袖通信の初期設定の代わりに，参照先の仮想メモリアドレスを取得し，内部的に袖領域のアドレスとして保持させるように設計した．袖通信の開始時，終了時に隣接プロセス間で同期を取ることで，データレースを防ぐ．

5.2 PVAS を用いた gmove 通信

現在の gmove 通信は，図 6 と同様に，送信側は送信データを pack バッファにコピーし，MPI によって通信する．受信側は，MPI によって unpack バッファに受信し，バッファから unpack する．通信は MPI_Isend/Irecv による 1 対 1 通信によって行われる．参照データが複数プロセスに渡るとき，1 対 1 通信は複数回行われ，通信のたびに pack/unpack が行われる．送受信に必要な pack バッファ，unpack バッファは pack/unpack が発生するたびにメモリに確保される．

設計した gmove 通信は，reflect 通信と同様に，送信側の仮想メモリアドレスから受信側の仮想メモリアドレスへ memcpy によって通信する．MPI_Isend/Irecv を memcpy とし，pack/unpack を省く．通信パターンを限定したため，同じ gmove のパターンが使用される際，初回に通信相手のメモリアドレスやストライド距離を記憶し，2 回目以降は記憶された情報を使用する．現在の PVAS による実装では，これらの情報交換に MPI を使用している．

6. 性能評価

MIC 内における，姫野ベンチマークによるステンシル計算と，NPB-CG による性能を示す．

6.1 実験環境

本稿で使用した環境を**表 1** に示す．MPSS は PVAS のパッチを当てた 2.6.38.8-pvas+mpss3.2.3 である．XMP ではコードを変換する上で，グローバル配列の参照がポインタとして置き換えられることで，MIC において重要であるプリフェッチがコンパイラによって効率的に挿入されない．そこで，手動プリフェッチを加えることで最適化した．

6.2 姫野ベンチマーク

実験には，姫野ベンチマーク [9] を使用し，512×256×256 要素のサイズ L を使用した．3 次元ステンシル計算を，i, j 方向に同数で 2 次元分割し，reflect 指示文によって袖通信を行った結果を**図 8** に示す．図 8 における，MPI は MPI によって実装されたもの [9] であり，OMP は i,j 反復を OpenMP によって並列化したものである．XMP-MPI, XMP-PVAS は，XMP によって実装されたものであり，通信に MPI を使用した結果と PVAS を使用した結果を示し

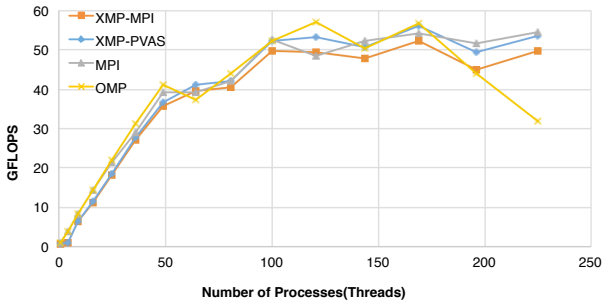

図 8 姫野ベンチマークの性能

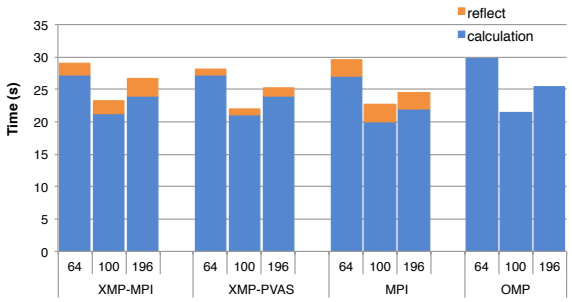

図 9 姫野ベンチマークにおける計算時間と通信時間

ている．

本稿で設計した XMP-PVAS は，XMP-MPI に比べ性能が向上していることがわかる．性能のピークである 169 プロセスのとき，1.076 倍の性能を達成した．64，100，196 プロセスのときの，XMP-MPI，XMP-PVAS，MPI における計算時間と reflect 通信の時間を**図 9** に示す．100 プロセスのとき，XMP-MPI に比べ 45%に抑えることができた．これは，pack/unpack が不要になったこと，MPI 通信を memcpy1 回に置き換えたためである．MPI による実装と比較しても，通信時間を抑えることができた．121 プロセスのとき，MPI よりも高い性能を得ることができた．

6.3 NAS Parallel Benchmarks

PVAS を用いた XMP による NPB-CG[10] の性能評価を示す．CG を 2 次元分割するとき，列に対して重複して持たせたベクトル同士の内積計算が発生する．この内積計算には，片方のベクトルを行に対して重複して持たせたベクトルに転置する必要がある．そのため，**図 10** のような通信パターンが発生するが，XMP では gmove 指示文により記述が可能である．

150000 × 150000 要素，各行の非ゼロ要素が 15 である CLASS C を使用したときの結果を**図 11** に示す．NPB-CG の制約のため，2^N プロセスで実行した．最大実行プロセス数は，MIC が持つ 244 スレッド以内で最大の 128 プロセスである．MPI は NPB3.3 の NPB-MPI に含まれる CG[10] であり，MPI コンパイラに Intel MPI を使用する．OMP は，ソウル大学校の Center for Manycore Programming により配布されている NPB-OMP の CG を用いる [11]．XMP-MPI，XMP-PVAS は，XMP によって実装された [12] ものであり，それぞれ，バックエンドに MPI，PVAS を使用したときの結果である．XMP-MPI の gmove 通信では，pack/unpack の度に，pack/unpack バッファを確保することによって，性能が大きく低下していることがわかった．そこで，pack/unpack バッファを初回に確保し，2 回目以降は確保したバッファを使い回すように修正した．この結果を XMP-MPI-NOALLOC とする．

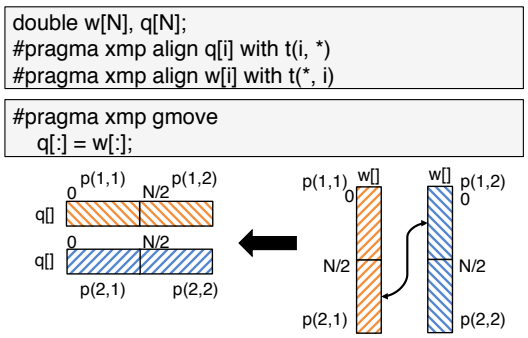

図 10 XMP における CG の通信パターン

図 11 から，NPB-CG では，MPI の性能が最も高く，XMP，OpenMP よりも高いという結果となった．32 プロセス，64 プロセス，128 プロセスで実行したときの通信時間を**図 12** に示す．図 12 から，XMP の gmove 通信，reduction 通信の時間が MPI に比べて大きいことがわかる．この通信時間によって，図 11 のような MPI と XMP の性能差になったと考えられる．通信を行わず，計算だけを行ったときの性能を**図 13** に示す．MPI と XMP で多少の差はあるが，プリフェッチ等の最適化が不足していることによると考えられる．これについては現在調査中である．

XMP-MPI-NOALLOC は，pack/unpack で行うバッファ確保によるオーバヘッドが無くなったため，XMP-MPI に比べて gmove 通信の時間が減少している．PVAS によって実装した XMP-PVAS の性能が，XMP-MPI，XMP-MPI-NOALLOC に比べ，性能を向上させることができた．128 プロセスのとき，XMP-MPI-NOALLOC に比べ，1.071 倍の性能を達成することができた．図 12 から，XMP-PVAS の gmove 通信を XMP-MPI，XMP-MPI-NOALLOC に比べ減少させることができた．中でも，128 プロセスのとき，XMP-PVAS の gmove 通信時間を XMP-MPI-NOALLOC の 34%に抑えることができた．

7. 結論と課題

本稿では，MIC 内の通信方法として PVAS を使用し，MIC 内並列化の性能評価を行った．XMP の MIC 内袖通信と分散配列の通信において，PVAS を使用することで，従来よりも高速な通信を目的とした．MIC 内の MPI 通信

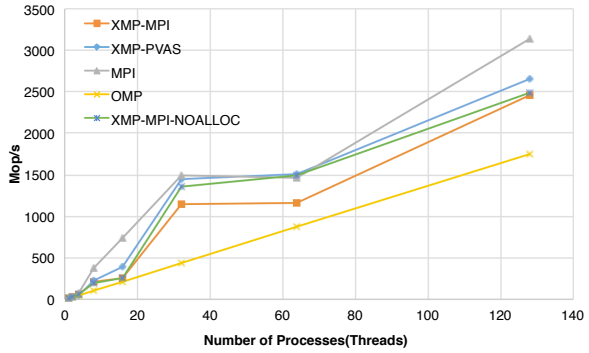

図 11 NPB-CG の性能

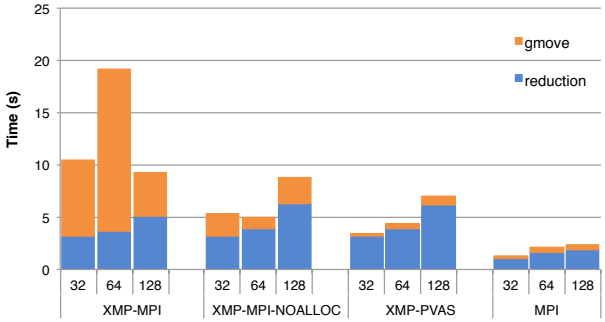

図 12 NPB-CG の反復回数 32 回による通信時間

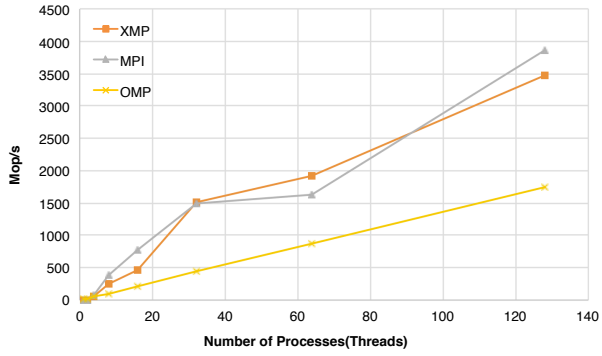

図 13 NPB-CG の通信を省いた計算時間

では，通信のための pack/unpack と MPI 通信が発生した．PVAS を使用することで，pack/unpack を省き，MPI 通信を memcpy1 回に置き換えた．その結果，3 次元ステンシル計算の 2 次元分割では，袖通信の通信時間を 45%に減少させることができた．CG の 2 次元分割では，重複して持たせたベクトルの転置の通信時間を 34%程度に減少させることができた．

姫野ベンチマークでは，MPI，OpenMP，XMP の性能が同程度であり，プロセス数によっては，XMP-PVAS を使用するこで，MPI よりも性能を引き出すことができた．NPB-CG では，MPI と XMP が OpenMP よりも高い性能を得ており，MPI や XMP を使用することが望まれる．

XMP は gmove 通信と reduction 通信によって，性能が低下した．PVAS を使用することで，gmove 通信の時間を抑えることができ，MPI との性能差を縮めることができた．PVAS を用いることで，reduction の通信時間を抑えることができれば，さらに MPI との性能差を縮めることができる．

現在の設計，実装では，MIC 内並列化に対応しているが，MIC 間，ノード間通信に対応していない．そこで，今後は，大規模計算に対応できるように，複数 MIC，複数ノードを対象とした設計，実装を行い，性能測定を行う．また，他の通信として，reduction 通信の設計を行う予定である．

参考文献

[1] PGAS: Partitioned Global Address Space, http://www.pgas.org/.
[2] XcalableMP: XcalableMP 仕様書，http://www.xcalablemp.org/download/spec/xmp-spec-1.2.pdf.
[3] XcalableMP: Omni Compiler Project, http://omni-compiler.org/index.html.
[4] A.Shimada, B.Gerofi, A.Hori and Y.Ishikawa: Proposing a New Task Model Towards Many-core Architecture, *Proceedings of the First International Workshop on Many-core Embedded Systems*, MES '13 (2013).
[5] 島田明男， 堀敦史， 石川裕：新しいタスクモデルによる MPI ノード内通信の高性能化，情報処理学会研究報告，Vol. 2014-OS-130, No. 18, pp. 1–8 (2014).
[6] Intel: Intel Xeon Phi Coprocessor 7100 Series, http://ark.intel.com/ja/products/series/75809.
[7] M.Ikei and M.Sato: A PGAS Execution Model for Efficient Stencil Computation on Many-Core Processors., *CCGRID*, pp. 305–314 (2014).
[8] OpenMPI: Open Source High Performance Computing, http://www.open-mpi.org/.
[9] 姫野ベンチマーク：姫野ベンチマーク, http://www.open-mpi.org/.
[10] NPB: NAS Parallel Benchmarks, http://www.nas.nasa.gov/publications/npb.html.
[11] SNU-NPB: SNU NPB Suite, http://aces.snu.ac.kr/software/snu-npb/.
[12] M.Nakao, J.Lee, T.Boku and M.Sato: XcalableMP Implementation and Performance of NAS Parallel Benchmarks, *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*, PGAS '10, pp. 11:1–11:10 (2010).