Vol.2014-HPC-143 No.34
2014/3/4

# Tightly Coupled Acceleratorsアーキテクチャに向けたXcalableMP拡張

中尾 昌広[1,2,a)]　村井 均[1]　下坂 健則[1]　田渕 晶大[3]　塙 敏博[4]　児玉 祐悦[2,3]　朴 泰祐[2,3]
佐藤 三久[1,2,3]

**概要**：本稿では，並列言語 XcalableMP（XMP）を拡張し，Tightly Coupled Accelerators（TCA）を搭載した GPU クラスタにおける生産性の高いプログラミングモデルを提案する．提案モデルは，TCA を扱うための典型的な処理を自動化し，さらにアクセラレータのためのプログラミングモデルである OpenACC を扱えるように XMP を拡張することで，TCA を搭載した GPU クラスタ上で動作するアプリケーションを簡易に作成可能にする．本稿では，提案モデルの優位性を調べるための初期評価として，従来モデルである MPI と OpenACC および TCA を扱うための API を用いたプログラミングモデルとの生産性と性能の比較を行った．2 次元 Laplace 方程式を用いてそれらの比較を行った結果，提案モデルの生産性は従来モデルと比較して高く，また提案モデルの性能は従来モデルの 98.5%以上を発揮できることを示した．

## 1. はじめに

高メモリバンド幅と優れた演算性能を持つ Graphics Processing Unit（GPU）を搭載した GPU クラスタが，HPC アプリケーションのための計算環境として普及しつつある．GPU クラスタは，2013 年 11 月に発表されたスーパーコンピュータの消費電力効率のランキングである Green500 [1] の 1〜11 位のすべてにランクインされていることから，その省電力性についても特長であると言える．

GPU クラスタにおいて，ある GPU メモリ上のデータを他ノードが持つ GPU メモリに転送する場合，ホストのメインメモリにデータをコピーする必要がある．さらに，ノード間の転送の際に Infiniband などを用いるため，データはプロトコル変換される．これらの処理から生じる通信遅延は，特にデータサイズが小さい場合，アプリケーションの性能のボトルネックになることが知られている．今後の HPC アプリケーションは，Weak Scaling より Strong Scaling で計算する機会が増えると考えられているため [2]，GPU 間の通信性能はより重要になると考えられる．

1 理化学研究所 計算科学研究機構
RIKEN Advanced Institute for Computational Science
2 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba
3 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
4 東京大学情報基盤センター
Information Technology Center, The University of Tokyo
a) masahiro.nakao@riken.jp

このような背景から，筑波大学計算科学研究センターでは，GPU 間の直接通信を可能にする密結合並列計算加速機構 Tightly Coupled Accelerators（TCA）[3–6] の開発が行われている．TCA を用いることで GPU 間の通信を高速に行えるため，TCA は次世代の計算環境のための要素技術として注目されている．

GPU クラスタ上で動作するアプリケーションを作成するには，プロセス間のデータ転送には MPI ライブラリを用い，GPU を用いた計算には CUDA [7] もしくは OpenCL [8] を用いることが主流である．しかしながら，プログラミングコストが大きいという問題点が存在するため，OpenACC [9] によるプログラミングモデルが注目されている．OpenACC では，C，C++，Fortran で記述されたコードに指示文を挿入することで，アクセラレータ上で動作するアプリケーションを作成することが可能になる．ただし，OpenACC は，単体ノードにおけるアクセラレータプログラミングを対象としている．また，TCA を用いてアプリケーションを作成するには，TCA を扱うための専用の API を用いて，GPU メモリ上のデータを操作する必要がある．具体的には，DMA のためのハンドルの生成，転送データの情報の登録，データ転送などの基本的な手続きが必要である．

本稿では，TCA を搭載した GPU クラスタを対象とした生産性の高いプログラミングモデルを提案する．そのプログラミングモデルは，Partitioned Global Address Space（PGAS）言語 XcalableMP（XMP）[10–12] を TCA および OpenACC [9] に対応するように拡張することで実現す

© 2014 Information Processing Society of Japan

る．XMP は分散メモリ環境においてデータマッピングや通信を簡易に行える機能を持つため，MPI と比較して生産性が高いという特徴を持つ．XMP を拡張することで TCA の利用に必要な処理の自動化を行い，さらに XMP が提供するデータマッピングなどの機能と OpenACC が提供するアクセラレータに処理をオフロードする機能を組合せることにより，GPU クラスタ上で動作するアプリケーションを簡易に作成できることを示す．

本稿の構成は下記の通りである．2 章では関連研究について述べる．3 章では XMP について，4 章では TCA について説明し，5 章では XMP に対する拡張について説明する．6 章では提案モデルの生産性と性能について評価し，7 章で本稿をまとめる．

## 2. 関連研究

### 2.1 単体ノードにおける GPU のためのプログラミングモデル

GPU を対象としたプログラミングには CUDA [7] と OpenCL [8] がよく用いられる．CUDA は NVIDIA 社が提供している GPU 開発環境であり，その対象は同社の GPU のみとなっている．それに対し，OpenCL はアクセラレータの種類に依存しない開発環境を提供しており，同じ記法で様々なメーカの GPU，CPU，また Cell Broadband Engine などで動作するアプリケーションを作成することが可能である．しかし，CUDA や OpenCL はアーキテクチャを強く意識したプログラミングを行う必要であるため，ハードウェアの性能を引き出すことができる反面プログラミングコストは大きく，またポータビリティも低い．

そのため，プログラマの負担を軽減するための研究がいくつか行われている．OpenACC [9] は，CUDA や OpenCL のような独自の記法を用いずに，逐次コードに指示文を挿入することで様々なアクセラレータ上で動作するアプリケーションを作成できる仕様である．指示文を用いるため学習コストおよびプログラミングコストは低く，さらに段階的な並列化が可能である．また，OpenACC コンパイラがある程度の最適化を行える仕様であるため，ポータビリティも高い．他の指示文を用いたプログラミングモデルとしては，OpenMP の記法を用いた OMPCUDA [13] や OpenMPC [14] が提案されており，逐次コードから CUDA コードを生成することが可能である．さらに，2013 年 7 月にリリースした OpenMP 4.0 で target 構文によるアクセラレータ対応が行われたため，今後の OpenMP を用いた GPU プログラミングについても注目していく必要がある．

### 2.2 GPU クラスタにおける PGAS 言語によるプログラミングモデル

XMP-dev [12,15,16] はアクセラレータのための XMP の言語拡張であり，ホスト・アクセラレータ間のデータ転送やループ文のスレッド並列化など，アクセラレータに対する典型的な処理を指示文で表現することができる．また，異なるプロセスにある GPU 間の通信を 1 つの指示文で表現することができる．なお，XMP-dev の実装には，アクセラレータ用のコードを CUDA で出力する実装 [12,15] と OpenCL で出力する実装 [16] の 2 種類が存在する．

他の PGAS 言語における取り組みには，Unified Parallel C（UPC）の拡張があり [17]，UPC が持つ shared ポインタや shared 配列を GPU メモリ上で扱うことを可能にしている．さらに，UPC の実装のネットワークレイヤで用いられている GASNet [18] も拡張し，GPU メモリに対する片側通信もサポートしている．OpenSHMEM では，既存の関数を用いてデバイスメモリの確保やデバイスメモリ間のデータ転送を行う仕様が提案されている [19]．なお，文献 [19] では，OpenSHMEM と CUDA もしくは OpenCL とを組合せたプログラミング環境を想定している．X10 および Chapel においても GPU 対応が行われており，それぞれの言語のシンタックスを用いた GPU プログラミングを行うことができる [20, 21]．文献 [20, 21] における X10 および Chapel の実装では，GPU 用に記述されたコードは CUDA コードに変換される．

## 3. XcalableMP

本章では，本稿で用いる XMP の機能についてのみ説明する．その他の機能については，文献 [11] を参考されたい．

### 3.1 特徴

XMP は分散メモリ型システムを対象とした並列プログラミングモデルであり，その仕様は PC クラスタコンソーシアム [22] の並列プログラミング言語 XcalableMP 規格部会によってオープンな検討により決定されている．XMP の仕様は HPC アプリケーションでよく用いられる C 言語と Fortran 言語の両方に対応しており，その基本概念は同一になるように工夫されている．本稿では，C 言語をベースに説明する．

XMP は High Performance Fortran（HPF）[23] のように，指示文を用いてプログラムの並列化を行う．XMP と HPF の大きく異なる点は，HPF ではコンパイラが通信を自動的に生成するのに対し，XMP では性能チューニングを行い易くするために，通信が発生する箇所はプログラマが明示的に指定する点である．

### 3.2 実行モデル

XMP では，その実行単位をノードと呼ぶ．XMP の実行モデルは MPI と同様 Single Program Multiple Data であり，基本的には各ノードで同じ処理が重複実行される．コード上で宣言された変数や配列は，次節で述べる XMP の機能によって分散を定義しない限り，各ノードで重複し

© 2014 Information Processing Society of Japan

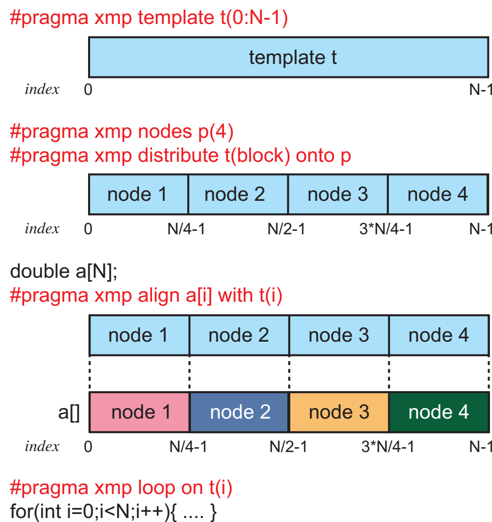


**図 1** データマッピングの流れとループ指示文を用いた並列処理の例

て宣言される．分散された配列に対して統一的な並列処理を行う場合や，他ノードが持つデータにアクセスする場合は，XMP の指示文を用いてそれらを行う．

### 3.3 データマッピングとループ文の並列処理

XMP では，テンプレートという仮想的なインデックス集合を用いてデータマッピングを定義する．**図 1** と下記にデータマッピングの流れとループ指示文を用いた並列処理の例を示す．

(1) template 指示文はテンプレートを定義する．図 1 では，1 次元のテンプレート *t* を定義しており，その下限値として 0 を，上限値として (N-1) を設定している．

(2) node 指示文は実行するノードの集合を定義する．図 1 では，4 つのノードで構成される 1 次元のノード集合 *p* を定義している．なお，テンプレートおよびノード集合は，多次元の定義を行うことも可能である．

(3) distribute 指示文はテンプレートをノード集合に分散する．分散の種類として，ブロック分散，サイクリック分散，ブロックサイクリック分散，user-defined 分散をサポートしている．図 1 ではブロック分散を指定している．なお，テンプレートの次元毎に分散を設定することが可能である．

(4) align 指示文は分散されたテンプレートを用いて実配列の分散を定義する．図 1 では，*t(i)* に設定された分散に従い，各ノードは配列要素 *a[i]* を確保することを示している．例えば，図 1 で N=16 の場合，node 1 は *a[0]* から *a[3]* までの要素を確保する．

(5) loop 指示文は直後のループ文を分散する．その分散方法は，loop 指示文の on 節で指定するテンプレートにより決定される．例えば，図 1 で N=16 の場合，node 1 はインデックス i=0 から 3 のイテレーションを担当する．

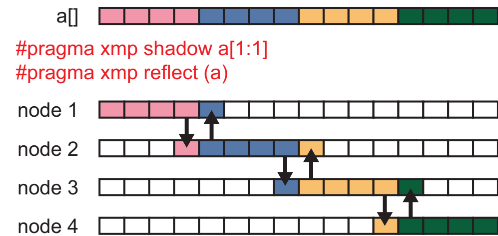


**図 2** 袖領域の宣言と同期の記述例と概念図（配列 *a[]* の要素数は 16 であり，配列 *a[]* は図 1 のように 4 ノードにブロック分散されているとする）

他にも XMP は典型的な集合通信や同期を行うための指示文を提供しているため，プログラマは逐次のイメージを保ったまま分散されたデータを操作することができる．

### 3.4 袖領域の定義と同期

XMP ではステンシル計算のアプリケーションを簡易に作成するために，分散配列に対して袖領域を設定する機能を提供している．**図 2** に，袖領域の宣言と同期の記述例と概念図を示す．shadow 指示文は袖領域の幅を指定する．図 2 では，分散配列の両端に 1 要素が袖領域として確保されている．袖領域の各要素は隣接ノードが持つ分散配列の端の要素と対応している．袖領域の同期には，reflect 指示文を用いる．reflect 指示文により，対応関係のある要素のコピーが図 2 の矢印の向きで行われるため，各ノードの袖領域は隣接ノードが持つ分散配列の端の要素と同じ値を持つことになる．

## 4. Tightly Coupled Accelerators

### 4.1 PEACH2

TCA 用のインタフェースボードとして，PEACH2 ボードがある [6]．TCA の基本的なハードウェア技術は PCI Express (PCIe) を応用したものであるため，PEACH2 ボード同士は PCIe 外部接続ケーブルによって接続される．1 章で述べた通り，ノードをまたぐ GPU 間の通信ではプロトコル変換が行われるが，TCA では PCIe のプロトコルのまま他ノードの GPU にデータを転送するため，その通信遅延は小さいという利点がある．

### 4.2 PEACH2 のネットワーク構成

TCA のみで数百ノードのクラスタを構成することはケーブル長の限界から物理的に困難であり，また性能上の利点も失われるため，8 から 16 ノード程度を TCA で結合する．このノードグループをサブクラスタと呼ぶ，各サブクラスタ同士は InfiniBand などで結合され，TCA と InfiniBand の階層ネットワークを構成する．PEACH2 ボードには，PCIe Gen2 x8 レーンのハード IP を 4 ポート内蔵した FPGA である Altera 社 Stratix IV GX [24] を用いている．4 ポートの内の 1 ポートはホスト側との接続に用い，残りの 3 ポー

© 2014 Information Processing Society of Japan

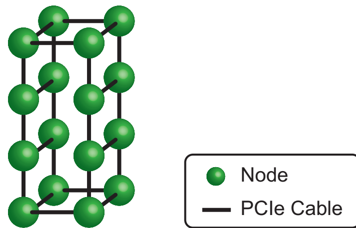


**図 3**　16 ノード場合の TCA システムのネットワーク構成例

トは他ノードとの接続に用いる．PEACH2 ボードを用いた TCA システムのネットワークはリングトポロジを構成する．**図 3** に，16 ノード場合の TCA システムのネットワーク構成例を示す．8 ノードのリングトポロジを 2 つ接続した構成をしている．

現在，TCA が対象にする GPU は NVIDIA 社の Kepler アーキテクチャである．Kepler では，GPUDirect Support for RDMA により PCIe デバイスとの間で直接 GPU メモリの読み書きが可能であり，その機能を TCA では利用している．

### 4.3　TCA における通信プログラミング

**表 1** に TCA が提供する API の一部を示す．PEACH2 チップは chaining DMA 機能を持ち，DMA 前に読み込み元と書き込み先 PCIe アドレス，転送サイズを指定したディスクリプタを登録することにより，連続した DMA を発行することが可能になる．一方，ディスクリプタを転送するオーバヘッドが存在するため，制御レジスタを設定することで数個の DMA を軽量に発行可能なレジスタモード転送も設けている．

## 5.　XcalableMP の拡張

### 5.1　概要

TCA は隣接通信において大きな効果を発揮するため，3.4 節で述べた XMP の shadow/reflect 指示文を TCA 用に拡張する．TCA は GPU メモリ上のデータを扱うため，TCA を用いるには GPU クラスタに対するプログラミングを行う必要がある．そこで，XMP が提供する分散配列を OpenACC 指示文が扱えるように XMP を拡張する．この 2 つの拡張を行うことにより，GPU クラスタ上で動作するアプリケーションを簡易に作成できると考えられる．

拡張の方針として，OpenACC 自体の拡張は行わないこととする．この方針により，既存のすべての OpenACC コンパイラとの連携が可能になる．また，拡張を行う XMP のコンパイラとして，Omni XMP Compiler [25] を用いる．

新しい Omni XMP Compiler のコンパイルプロセスの概略図を**図 4** に示す．まず，XMP コンパイラは XMP 指示文と OpenACC 指示文で記述されたベース言語（C 言語もしくは Fortran 言語）のコードを XMP のランタイム関数の呼び出しに変換する．その際，TCA を用いた通信を行うための XMP 指示文である **reflect_tca** 指示文の変換プロセスを追加する．また，OpenACC 指示文の引数の一部を変更する必要があるため，そのプロセスも追加する．それぞれの詳細は次節以降で説明する．次に，XMP コンパイラによって変換されたコードを，OpenACC コンパイラを用いてコンパイルする．その際，XMP のランタイムライブラリと TCA を操作するためのライブラリをリンクすることにより，実行ファイルを生成する．

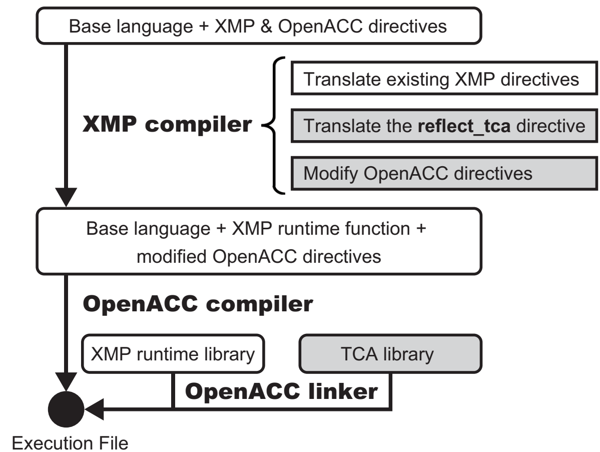


**図 4**　新しい Omni XMP Compiler のコンパイルプロセスの概略図（灰色の箇所が新規部分）

```
1  double a[N];
2  #pragma xmp template t(0:N−1)
3  #pragma xmp nodes p(4)
4  #pragma xmp distribute t(block) onto p
5  #pragma xmp align a[i] with t(i)
6  ...
7  #pragma acc data copy(a)
8  {
9  #pragma xmp loop on t(i)
10 #pragma acc parallel loop
11   for(int i=0;i<N;i++){ a[i] = ... }
12 }
```

**図 5**　XcalableMP と OpenACC の両方を用いたプログラム例

### 5.2　OpenACC のための拡張

#### 5.2.1　文法

**図 5** に XMP と OpenACC の両方を用いたプログラムの例を示す．まず，2〜5 行目の XMP 指示文は分散配列を各ノードに定義する．次に，7 行目の OpenACC の data 指示文はローカルノードが持つ分散配列をデバイスメモリに転送する．そして，9 行目の XMP の loop 指示文は 11 行目のループ文を各ノードで並列に実行する．その際，10 行目の OpenACC の parallel 指示文は，XMP 指示文によって分割されたループ文をさらにスレッド分割させることを示している．なお，9 行目と 10 行目は逆でも同じ意味になるようにする．また，OpenACC の parallel 指示文だけで

© 2014 Information Processing Society of Japan

**表 1** TCA が提供している API の一部

| 関数名（引数） | 機能 |
|---|---|
| tcaCreateHandle(tcaHandle *handle, void *ptr, size_t size, tcaMemoryType type) | ハンドルの作成 |
| tcaCreateDMADesc(int *tag, int desc_num) | ディスクリプタの作成 |
| tcaSetDMADesc_Memcpy(int desc_tag, int slot, int *next_slot, tcaHandle *dst_handle, off_t dst_offset, tcaHandle *src_handle, off_t src_offset, size_t size, tcaDMAFlag flag, int wait, int wait_tag) | 連続データを DMA に登録 |
| tcaSetDMADesc_Memcpy2D(int tag, int slot, int *next_slot, tcaHandle *dst_handle, off_t dst_offset, size_t dpitch, tcaHandle *src_handle, off_t src_offset, size_t spitch, size_t width, size_t height, tcaDMAFlag flag, int wait, int wait_tag) | ストライドデータを DMA に登録 |
| tcaStartDMADesc(int dma_ch) | DMA の実行 |
| tcaWaitDMARecvDesc(tcaHandle *src_handle, int wait, int wait_tag) | 転送の完了を待つ |
| tcaSetDMAReg_Memcpy(int dma_ch, int slot, int *next_slot, tcaHandle *dst_handle, off_t dst_offset, tcaHandle *src_handle, off_t src_offset, size_t size, tcaDMAFlag flag, int wait) | レジスタモードで 連続データを DMA に登録 |
| tcaSetDMAReg_Memcpy2D(int dma_ch, int slot, int *next_slot, tcaHandle *dst_handle, off_t dst_offset, tcaHandle *src_handle, off_t src_offset, size_t pitch, size_t width, size_t height, tcaDMAFlag flag, int wait) | レジスタモードで ストライドデータを DMA に登録 |
| tcaStartDMAReg(int dma_ch, int num_slots) | レジスタモードで DMA の実行 |
| tcaWaitDMARecvReg(tcaHandle *src_handle, int wait) | レジスタモード転送の完了を待つ |

はなく，kernels 指示文も利用可能にする．さらに，7 行目の OpenACC の data 指示文のようなデバイスメモリとの通信に関係する文についても，XMP で定義した分散配列を指定可能にする．

#### 5.2.2 制限

XMP の loop 指示文と 5.3 節で述べる TCA を用いるための **reflect_tca** 指示文以外の XMP 構文は，OpenACC の parallel 指示文と kernels 指示文の構造化ブロック内に記述できないこととする．

#### 5.2.3 実装

**loop 指示文などの処理** 図 5 の 9 行目の XMP の loop 指示文と 11 行目のループ文のみを Omni XMP Compiler が変換した場合，**図 6** のようになる．図 6 の 4 行目では，ループ文における各ノードが担当するインデックスを計算している．そのため，例えば図 5 の場合，図 6 の 4 行目と 5 行目の間に **#pragma acc parallel loop** を挿入するようにコード変換を行う．

**data 指示文などの処理** Omni XMP Compiler では，分散配列の領域は関数 malloc() を用いることで動的に確保される．OpenACC では動的に確保された領域がデバイスメモリとの通信対象になる場合，その開始要素と転送数を記述するという制約がある．そのため，data 指示文などに分散配列が指定された場合，その指定箇所を書き換える必要がある．図 5 の場合，7 行目は **#pragma acc data copy(a[start:size])** のように変換する．変数 **start** と **size** はローカルノードにおける分散配列の開始要素と転送数であり，それぞれの値は XMP の関数を用いて取得可能である．

### 5.3 TCA のための拡張

#### 5.3.1 文法

TCA を用いた隣接通信用の **reflect_tca** 指示文を新しく作成する．**reflect_tca** 指示文の書式は**図 7** の通りであり，reflect 指示文の書式と同一である．丸括弧内の *array-name* は配列名であり，複数指定可能である．角括弧以下はオプションであり，shadow 指示文で定義した袖幅と異なる幅に対して同期を行う場合に用いる．*reflect-width* には，*[/periodic/] int-expr* もしくは *[/periodic/] int-expr : int-expr* を指定する．*periodic* は文字列であり，分散配列における上限と下限の袖を更新したい場合に用いる．例えば図 2 の場合，node 1 は a[0] の要素を node 4 の a[16] の位置にコピーし，node 4 は a[15] の要素を node 1 の a[-1] の位置にコピーすることになる．*int-expr* は int 型の整数であり，コロンの前の値は各分散配列の左の袖幅を示し，コロンの後の値はその右の袖幅を示す．コロンがない場合は，左右とも同じ幅を示す．async 節がある場合は，非同期で実行する．async 節の引数である *id* は int 型の整数であり，完了の確認を行うための指示文で用いる．

#### 5.3.2 制限

XMP のプログラミングモデルに従うため，全実行ノードで同じ命令を実行する必要がある．具体的には，ノードによって袖幅が異なったり，特定のノード間のみの同期を行ったりすることは，**reflect_tca** 指示文では行えない．もし，それらの操作を行いたい場合は，OpenACC の host_data 指示文を用いて分散配列のポインタからデバイスポインタを取得し，プログラマが DMA のための登録や同期の処理を手動で行う必要がある．

© 2014 Information Processing Society of Japan

```
1 int _XMP_init_i;
2 int _XMP_cond_i;
3 int _XMP_step_i;
4 _XMP_sched_loop_template_BLOCK(0, N, 1, &(_XMP_loop_init_i), &(_XMP_loop_cond_i), &(_XMP_loop_step_i), ...);
5 for(int i = _XMP_loop_init_i; i < _XMP_loop_cond_i; i += _XMP_loop_step_i) { ... }
```

**図 6** XcalableMP によるループ文の変換例

```
#pragma xmp reflect_tca (array-name [,array-name]..) [width (reflect-width [,reflect-width]..)] [async (id)]
     where reflect-width must be one of:
            [/periodic/] int-expr
            [/periodic/] int-expr : int-expr
```

**図 7** **reflect_tca** 指示文の書式

```
1  double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
2  #pragma xmp nodes p(x, y)
3  #pragma xmp template t(0:YSIZE−1, 0:XSIZE−1)
4  #pragma xmp distribute t(block, block) onto p
5  #pragma xmp align u[j][i] with t(i, j)
6  #pragma xmp align uu[j][i] with t(i, j)
7  #pragma xmp shadow uu[1:1][1:1]
8  ...
9  #pragma acc data copy (uu,u)
10 {
11 for(k = 0; k < NITER; k++){
12 #pragma xmp loop (y, x) on t(y, x)
13 #pragma acc parallel loop collapse(2)
14   for(x = 1; x < XSIZE−1; x++)
15     for(y = 1; y < YSIZE−1; y++)
16       uu[x][y] = u[x][y];
17
18 #pragma xmp reflect_tca (uu)
19
20 #pragma xmp loop (y, x) on t(y, x)
21 #pragma acc parallel loop collapse(2)
22   for(x = 1; x < XSIZE−1; x++)
23     for(y = 1; y < YSIZE−1; y++)
24       u[x][y] = (uu[x−1][y] + uu[x+1][y] + uu[x][y−1] +
             uu[x][y+1])/4.0;
25
26 } // end of for−loop k
27 } // end of acc data
```

**図 8** XMP と OpenACC による 2 次元 Laplace 方程式の一部

### 5.3.3 実装

**図 8** に，拡張した XMP と OpenACC を用いて作成した 2 次元 Laplace 方程式のプログラムの一部を示す．この例を用いて，XMP のランタイムが行う処理の流れを説明する．18 行目の **reflect_tca** 指示文は，デバイスメモリにある配列 *uu[]* に対する袖領域の同期を実行する．図 8 のように **reflect_tca** 指示文に *width* 節がない場合は，7 行目の shadow 指示文で定義された袖領域の情報を登録し，同期を行う．もし *width* 節がある場合は，*width* 節から得られた袖領域の情報を登録し，同期を行う．なお，同期通信には PEACH2 のレジスタモードを用いている．

```
1  #pragma acc parallel
2  {
3  #pragma xmp loop (y, x) on t(y, x)
4  #pragma acc loop collapse(2)
5    for(x = 1; x < XSIZE−1; x++)
6      for(y = 1; y < YSIZE−1; y++)
7        uu[x][y] = u[x][y];
8
9  #pragma xmp reflect_tca (uu)
10
11 #pragma xmp loop (y, x) on t(y, x)
12 #pragma acc loop collapse(2)
13   for(x = 1; x < XSIZE−1; x++)
14     for(y = 1; y < YSIZE−1; y++)
15       u[x][y] = (uu[x−1][y] + uu[x+1][y] + uu[x][y−1] +
             uu[x][y+1])/4.0;
16 }
```

**図 9** GPU から同期命令を発行する 2 次元 Laplace 方程式

図 8 のように，ステンシル計算では一定のステップごとに同じ袖領域の同期が実行されることが多い．**reflect_tca** 指示文を実行する度に，情報の登録を行うのは非効率であるため，登録情報はキャッシュするように実装している．そのため，2 回目以降に同じ袖幅の **reflect_tca** 指示文を実行する場合は，同期のみが行われる．

図 8 では，OpenACC の parallel 指示文の構造化ブロックの外に **reflect_tca** 指示文があるため，ホストから TCA に対して処理を発行していることになる．しかし，GPU から TCA に対して処理を発行することも可能である．図 8 の 12 行目から 24 行目を，GPU から TCA に処理を発行するように変更した例を**図 9** に示す．図 8 と図 9 の異なる点は，図 8 では 2 つの parallel 指示文を用いているのに対し，図 9 では 1 つの parallel 指示文のみを用いており，さらに parallel 指示文の構造化ブロック内で **reflect_tca** 指示文を実行している点である．図 9 の記述法の方がカーネル関数の呼び出し回数は少なく，また記述量も少なくなる場合があるが，同期処理は GPU がホストを介してその処理を発行することになるため，同期処理は図 8 の方が高速に実行できる．

© 2014 Information Processing Society of Japan

## 6. 評価

### 6.1 概要

拡張した XMP と OpenACC を用いたプログラミングモデル（以下，提案モデル）の初期評価を行うため，MPI と OpenACC および TCA を扱うための API を用いたプログラミングモデル（以下，従来モデル）における生産性と性能を比較する．各評価には，ステンシル計算の典型例である 2 次元 Laplace 方程式を用いて行う．提案モデルを用いて作成した 2 次元 Laplace 方程式は図 8 の通りである．

提案モデルに対応したコンパイラは Omni XMP Compiler をベースに作成中であるが，一部の機能は未完成である．そのため，本稿の性能評価では手動でソースコードの変換を行った．

### 6.2 生産性評価

逐次プログラムからのコードの変更量と全行数を用いて，提案モデルと従来モデルの生産性を比較する．**表 2** に，逐次プログラムに対して新しく追加した行数，既存の行の一部を変更した行数，全行数を示す．表 2 に示す値は，空行およびコメント行は除いている．

提案モデルの XMP における 18 行の追加の内訳は，指示文の追加が 14 行，残りの 4 行はインクルードヘッダの追加と結果出力のためのノード番号の取得などである．OpenACC における追加の内訳は，図 8 の通り，指示文の追加が 3 行，残りの 2 行は構造化ブロックのための波括弧の追加である．

従来モデルにおける 81 行の追加の内訳は，隣接ノードにおけるデータ情報の生成と登録などが 41 行，袖の同期の実行が 9 行，OpenACC 指示文は 8 行，その他が 23 行である．OpenACC 指示文の追加行が提案モデルよりも多い理由は，TCA に対するハンドルの生成のために同期を行う配列の GPU 上のメモリアドレスが必要であり，そのアドレスの取得のために OpenACC の host_data 指示文を用いているからである．従来モデルにおける変更行の 13 行は，ループ文におけるグローバルインデックスからローカルインデックスの変換に要した行数である．

プログラミングモデルの生産性は行数のみでは判断することはできないが，提案モデルは行の追加のみで並列化を行うことができ，さらにグローバルインデックスをローカルインデックスに変換する必要もないため，提案モデルの方が簡易にプログラミングを行えると考えられる．

### 6.3 性能評価

筑波大学計算科学研究センターの HA-PACS/TCA システムの 2〜16 ノードを用いて，提案モデルと従来モデルの性能を評価する．HA-PACS/TCA のスペックを**表 3** に示す．問題サイズ（図 8 の定数 XSIZE と YSIZE）は 16,384，最外ループのイテレーション数（図 8 の定数 NITER）は 100 とした．1 ノードにつき 1 プロセス 1GPU を計算に用いる．評価値は 10 試行中の最良値を用いる．

**表 2** 生産性評価

| | 逐次プログラム | XMP+OpenACC | | MPI+TCA+OpenACC |
|---|---|---|---|---|
| | | XMP | OpenACC | |
| 追加行 | - | 18 | 5 | 81 |
| 変更行 | - | 0 | 0 | 13 |
| 全行数 | 46 | 69 | | 140 |

**表 3** HA-PACS/TCA システム

| | |
|---|---|
| CPU | Intel Xeon-E5 2680v2 2.8 GHz x 2 Socket |
| ピーク性能 | 224 GFlops/CPU |
| メモリ | DDR3 1866 MHz x 4 チャネル, 128GB |
| GPU | NVIDIA Tesla K20X x 4 GPU |
| ピーク性能 | 1.31 TFlops/GPU |
| メモリ | GDDR5 6GB/GPU |
| InfiniBand | Mellanox Connect-X3 Dual-port QDR |
| ノード数 | 64 |
| コンパイラ | PGI Compiler version 13.9 |
| 通信ライブラリ | mvapich2 1.8.1 |

**表 4** 実行に要した時間（単位は秒）

| 並列数 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|
| XMP+OpenACC | 8.98 | 5.99 | 4.62 | 3.95 |
| MPI+TCA+OpenACC | 8.85 | 5.99 | 4.56 | 3.89 |
| 性能比（%） | 98.53 | 100.00 | 98.64 | 98.56 |

**表 4** に，実行に要した時間を示す．すべての並列数において，提案モデルは従来モデルの 98.5%以上の性能を示すことがわかる．速度差の原因は，各ノードが担当する GPU にオフロードするループ文のインデックス計算を，提案モデルでは最外ループ内で行っているのに対し，従来モデルでは最外ループ外で行っているからである．しかし，一般的なアプリケーションで出現する GPU にオフロードするループ文の内容は，評価で用いた 2 次元 Laplace 方程式の内容よりも多い．そのため，一般的なアプリケーションに対して今回のような並列化を行った場合は，速度差はより小さくなると考えられる．

## 7. まとめと今後の課題

本稿では，TCA を搭載した GPU クラスタにおける生産性の高いプログラミングモデルを提案した．提案するモデルは，XMP を拡張することにより TCA を扱うための典型的な処理を自動化する．さらに，OpenACC が持つ生産性の高い GPU プログラミングを XMP を拡張することにより，GPU クラスタ上で行えるようにした．

提案モデルの優位性を調べるための初期評価として，従来モデルである MPI と OpenACC および TCA を扱うための API を用いたプログラミングモデルとの生産性と性能の比較を行った．2 次元 Laplace 方程式を用いて比較を

行った結果，提案モデルの生産性は従来モデルと比較して高く，また提案モデルの性能は従来モデルの 98.5%以上を発揮できることを示した．

今後の課題として，本稿で行った TCA における同期通信の実装には PEACH2 が提供するレジスタモードを利用したが，転送サイズの大きな通信にはハードウェアの制約からレジスタモードを利用できない場合がある．そのため，ある条件をもとにレジスタモードから chaining DMA 機能に通信を切り替えるといったことを考えている．また，本稿で述べた XMP と OpenACC の混合利用は XMP の一部の指示文のみを対象にしているため，それらの混合利用における詳細を検討する必要がある．さらに，提案モデルを実アプリケーションに適用し，より大規模な環境において提案モデルの有効性を調べる予定である．

**謝辞** 本研究は JST-CREST 研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」，研究課題「ポストペタスケール時代に向けた演算加速機構・通信機構統合環境の研究開発」による．また，HA-PACS/TCA システムの利用は筑波大学計算科学研究センターの学際共同利用プログラムによる．

## 参考文献

[1] The Green500 List, Nov, 2013. http://www.green500.org/lists/green201311.

[2] HPCI 技術ロードマップ白書, March 2012. http://open-supercomputer.org/wp-content/uploads/2012/03/hpci-roadmap.pdf.

[3] 塙敏博, 児玉祐悦, 朴泰祐, 佐藤三久. Tightly coupled accelerators アーキテクチャに基づく gpu クラスタの構築と性能予備評価. 情報処理学会論文誌（コンピューティングシステム）, Vol. 6, No. 4, pp. 14–25, 2013.

[4] Toshihiro Hanawa, Yuetsu Kodama, Taisuke Boku, and Mitsuhisa Sato. Interconnect for tightly coupled accelerators architecture. In *IEEE 21st Annual Sympsium on High-Performance Interconnects (HOT Interconnects 21)*, pp. 79–82, 2013.

[5] Toshihiro Hanawa, Yuetsu Kodama, Taisuke Boku, and Mitsuhisa Sato. Tightly coupled accelerators architecture for minimizing communication latency among accelerators. In *The Third International Workshop on Accelerators and Hybrid Exascale Systems (AsHES2013) in conjunction with IEEE International Parallel and Distributed Processing Symposium (IPDPS2013)*, pp. 1030–1039, 2013.

[6] 朴泰祐, 佐藤三久, 塙敏博, 児玉祐悦, 高橋大介, 建部修見, 多田野寛人, 藏増嘉伸, 吉川耕司, 庄司光男. 演算加速装置に基づく超並列クラスタ ha-pacs による大規模計算科学. 情報処理学会研究報告. [ハイパフォーマンスコンピューティング], Vol. 2011, No. 21, pp. 1–7, jul 2011.

[7] 並列プログラミングおよびコンピューティングプラットフォーム— cuda. http://www.nvidia.co.jp/object/cuda-jp.html.

[8] OpenCL - the open standard for parallel programming of heterogeneous systems. http://jp.khronos.org/opencl.

[9] OpenACC. http://www.openacc-standard.org.

[10] Masahiro Nakao, Jinpil Lee, Taisuke Boku, and Mitsuhisa Sato. Productivity and performance of global-view programming with xcalablemp pgas language. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, CCGRID '12, pp. 402–409, Washington, DC, USA, 2012. IEEE Computer Society.

[11] XcalableMP Specification Working Group. Xcalablemp specification version 1.2, 11 2012. http://www.xcalablemp.org/spec/xmp-spec-1.2.pdf.

[12] Jinpil Lee. *A Study on Productive and Reliable Programming Environment for Distributed Memory System.* PhD thesis, University of Tsukuba, 2012.

[13] Satoshi Ohshima, Shoichi Hirasawa, and Hiroki Honda. Ompcuda: Openmp execution framework for cuda based on omni openmp compiler. In *Proceedings of the 6th International Conference on Beyond Loop Level Parallelism in OpenMP: Accelerators, Tasking and More*, IWOMP'10, pp. 161–173, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] Seyong Lee and Rudolf Eigenmann. Openmpc: Extended openmp programming and tuning for gpus. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pp. 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[15] 李珍泌, Tran Minh Tuan, 小田嶋哲哉, 朴泰祐, 佐藤三久. Pgas 並列プログラミング言語 xcalablemp における演算加速装置を持つクラスタ向け拡張仕様の提案と実装. 情報処理学会論文誌. コンピューティングシステム, Vol. 51, No. 10, pp. 1234–1252, Octorber 2010.

[16] 野水拓馬, 高橋大介, 李珍泌, 朴泰祐, 佐藤三久. 並列言語 xcalablemp のアクセラレータ向け言語拡張の opencl 実装. Technical Report 9, 研究報告ハイパフォーマンスコンピューティング（HPC）, March 2012.

[17] Yili Zheng, Costin Iancu, Paul H. Hargrove, Seung-Jai Min, and Katherine Yelick. Extending unified parallel c for gpu computing. In *SIAM Conference on Parallel Processing for Scientific Computing*, pp. 24–26, February 2010.

[18] GASNet communication system. http://gasnet.lbl.gov.

[19] S. Potluri, D. Bureddy, H. Wang, H. Subramoni, and D.K. Panda. Extending openshmem for gpu computing. *Parallel and Distributed Processing Symposium, International*, Vol. 0, pp. 1001–1012, 2013.

[20] Dave Cunningham, Rajesh Bordawekar, and Vijay Saraswat. Gpu programming in a high level language: Compiling x10 to cuda. In *Proceedings of the 2011 ACM SIGPLAN X10 Workshop*, X10 '11, pp. 8:1–8:10, New York, NY, USA, 2011. ACM.

[21] Albert Sidelnik, Bradford L. Chamberlain, Maria J. Garzaran, and David Padua. Using the high productivity language chapel to target gpgpu architectures. Technical report, Cray, 2011.

[22] PC Cluster Consortium. http://www.pccluster.org/en/.

[23] Charles H. Koelbel, David B. Loverman, Robert S. Shreiber, Guy L. Steele Jr., and Mary E. Zosel. *The High Performance Fortran Handbook.* MIT Press, 1994.

[24] Altera Corp. Stratix iv device handbook. http://www.altera.co.jp/literature/lit-stratix-iv.jsp.

[25] Jinpil Lee and Mitsuhisa Sato. Implementation and performance evaluation of xcalablemp: A parallel programming language for distributed memory systems. *2012 41st International Conference on Parallel Processing Workshops*, Vol. 0, pp. 413–420, 2010.

© 2014 Information Processing Society of Japan