Vol.2015-HPC-151 No.16
2015/10/1

PGAS言語XcalableACCを用いたHPC Challengeベンチマークの実装と評価

中尾 昌広[1,2,a)]　村井 均[1]　岩下 英俊[1]　田渕 晶大[3]　塙 敏博[4]　朴 泰祐[2,3]　佐藤 三久[1]

概要：我々はPGAS言語XcalableMP（XMP）のアクセラレータクラスタのための拡張であるXcalableACC（XACC）を提案している．本稿では，XACCの性能評価を行うため，HPC Challenge（HPCC）ベンチマークの中のSTREAM，High Performance Linpack，Fast Fourier TransformについてXACCによる実装方法について検討する．STREAMについては実際に実装を行い，筑波大学のHA-PACS/TCAクラスタを用いて性能評価を行った．性能評価の結果，STREAMはピーク性能の26%を達成し，XMPで記述したSTREAMと比較して3.3倍の性能を達成した．また，XACCで作成したSTREAMの計算カーネルはXMPで記述した計算カーネルに9行のコード変更で記述できたことから，XACCは簡易にプログラミングを行えると言える．

1. はじめに

メモリバンド幅および消費電力に対する演算性能に優れたアクセラレータを搭載したクラスタシステム（アクセラレータクラスタ）が普及している．アクセラレータクラスタで動作するアプリケーションを作成するためには，ユーザは2種類のプログラミングモデルを組合せて用いる必要がある．1つはアクセラレータに対してデータと計算のオフロードを行うためのプログラミングモデルであり，もう1つはノード間の通信を行うプログラミングモデルである．前者はOpenACC [1]，OpenMP4.0 [2], Open Computing Language（OpenCL）[3]，Compute Unified Device Architecture（CUDA）[4]などがよく用いられる．特に，OpenACCとOpenMP4.0は指示文を用いたプログラミングモデルであり，その生産性は優れていることが知られている．後者はMessage Passing Interface（MPI）[5]がよく用いられる．しかし，MPIを用いたプログラミングは各プロセスに対するデータのマッピングや転送，各プロセスの処理を個々に記述する必要があるため，その生産性は低いことが問題となっている．

[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

我々は，アクセラレータクラスタのためのプログラミングモデルXcalableACC（XACC）[6–8]を提案している．XACCはPartitioned Global Address Space（PGAS）言語XcalableMP（XMP）[9–11]のアクセラレータ拡張であり，XMPとOpenACCとの相互運用を可能にしたプログラミングモデルである．XACCでは，データのマッピングなどの処理をMPIの代わりにXMP指示文を用い，アクセラレータに対する処理はOpenACC指示文を用いる．さらに，XMP指示文を拡張し，XMP指示文とOpenACC指示文の組合せだけでは不可能な，アクセラレータ間の直接通信の記述も可能である．これらの機能により，XACCを用いることでアクセラレータクラスタで動作するアプリケーションを簡易に開発できることが期待できる．

本稿の目的はXACCの生産性と性能を評価することである．その評価のために，計算システムの性能を多角的に評価するベンチマーク集であるHPC Challenge（HPCC）ベンチマーク [12,13]を用いる．HPCCベンチマークを用いた並列言語のコンテストにHPCC Awards Competition Class 2 [14]がある．HPCC Awards Competition Class 2では，HPCCベンチマークの中でも主要な4つのベンチマークSTREAM, High Performance Linpack（HPL），Fast Fourier Transform（FFT），RandomAccessの中から3つ以上を用いて，並列言語による実装と性能評価が行われる．本稿では，HPCCベンチマークのSTREAM，HPL，FFTについてXACCによる実装を試みる．特にSTREAMについては性能評価も行った．なお，RandomAccessは細粒度の通信が頻発し，アクセラレータクラスタには不向きと

考えられるため，本稿では検討しない．

本稿の構成は下記の通りである．2 章では過去に我々が行ってきた XACC における性能評価についてまとめ，3 章では XACC のプログラミングモデルおよびコンパイラの実装について説明する．4 章では HPCC ベンチマークの実装と STREAM の性能評価について述べ，5 章ではまとめと今後の課題について述べる．

2. これまでの XcalableACC における性能評価

本章では，HPCC ベンチマーク以外を用いた XACC の評価についてまとめる．

我々は文献 [6,7] において，姫野ベンチマーク [15] による評価を行った．姫野ベンチマークはステンシルアプリケーションの 1 種であり，非圧縮流体解析コードの性能評価を行うベンチマークである．文献 [6, 7] で作成された Omni XACC コンパイラでは，アクセラレータ間の通信を小さいレイテンシで行うことができる Tightly Coupled Accelerators（TCA）[16, 17] を用いた．そのため，OpenACC と MPI で実装した姫野ベンチマークと比較して，XACC で実装した姫野ベンチマークは性能が高く，かつ半分以下の行数で実装できることが示した．

さらに，我々は文献 [8] において，Omni XACC コンパイラの実装についての詳細を説明し，アクセラレータ間の通信の実装の 1 つとして GPUDirect RDMA [18] を追加した．文献 [8] では，姫野ベンチマークおよび NAS Parallel Benchmarks（NPB）[19] CG の評価を行った．NPB CG は大規模疎行列の最小固有値を共役勾配法を用いて解くベンチマークである．XACC で記述した NPB CG の行数は，OpenACC と MPI で実装した NPB CG の 79%の行数であり，GPUDirect RDMA を利用している Omni XACC コンパイラを用いた姫野ベンチマークおよび NPB CG の性能は，それぞれの OpenACC と MPI で実装された性能とほぼ同等であることを示した．

3. XcalableACC

3.1 概要

XACC は，OpenACC 指示文，XMP 指示文，拡張した XMP 指示文（XACC 指示文）を用いたアクセラレータクラスタのためのプログラミングモデルである．XMP 指示文は分散配列の定義，ループ文の分散，ホスト間のデータ転送などを行う．そして，XMP 指示文が定義した分散配列を OpenACC 指示文に指定することにより，その分散配列に対してアクセラレータを用いた処理（例えば，ホストとアクセラレータ間の通信など）を行うことができる．また，XACC 指示文を用いることで，アクセラレータ間の直接通信の記述を行うことができる．XACC は，C 言語および Fortran の拡張として定義されている．本稿では C 言語の構文について説明する．

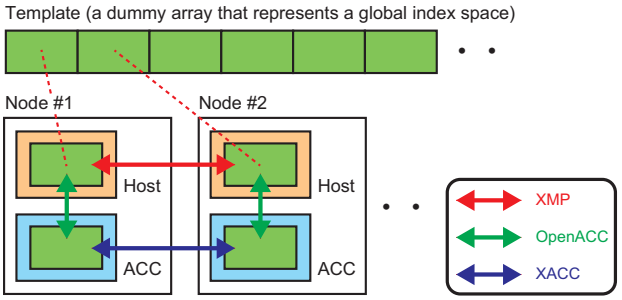

図 1 XcalableACC の概念図

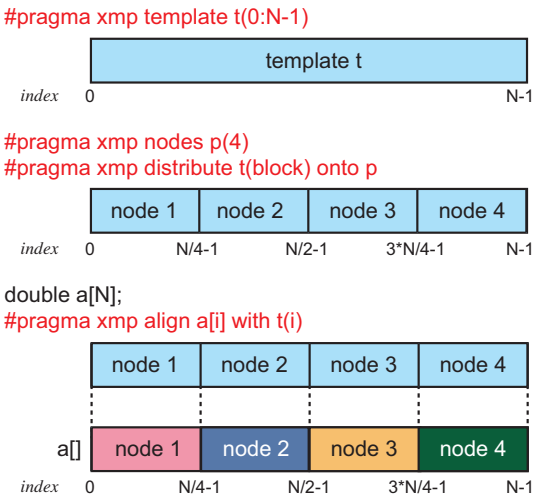

図 2 分散配列の定義 [10]

XACC の概念図を**図 1** に示す．XACC および XMP では，実行単位を “ノード” と呼び，全ノードで同じプログラムが実行される．そして，仮想インデックス集合である “テンプレート” を用いて分散配列を定義する．図 1 において各ノードに存在する緑色の矩形は，各ノードに割り当てられた分散配列を示している．XACC では，XMP 指示文を用いてホスト上のメモリに分散配列を定義し，OpenACC 指示文を用いてその分散配列をアクセラレータに転送する．また，ホスト間のデータ転送には XMP 指示文を用いるのに対し，アクセラレータ間の転送には XACC 指示文を用いる．次節より，各処理の詳しい説明を例を交えて行う．

3.2 分散配列の定義

図 2 に分散配列の定義の例を示す．XACC における分散配列の定義は XMP と同じである．なお XACC および XMP では，指示文で定義されていない配列は，全ノードで重複して持つ．

(1) **template** 指示文はテンプレート t を定義する．t のインデックスは 0〜N-1 である．
(2) **node** 指示文はノード集合 p を定義する．p は 4 ノードから構成されている．
(3) **distribute** 指示文はテンプレート t をノード集合 p に

```
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++){
12     a[i] = ... ;
13   }
14 }
```

図 3 XcalableACC のコード例

指定した分散方式（図 2 の場合は block）で割り当てる．block 分散の意味は，同じ幅のインデックスを各ノードに割り当てることを意味している．他の分散方式として，cyclic 分散，block-cyclic 分散，不均等ブロック分散がある．

(4) **align** 指示文は分散配列 *a[]* を，各ノードに割り当てたテンプレート *t* に整列させる．図 2 の場合に N=16 であるならば，各ノードは分散配列 *a[]* の 4 要素ずつを持つ．

3.3 ホストからアクセラレータへのデータ転送とループ文の並列化

図 3 に XACC のコード例を示す．1〜5 行目では，図 2 と同様に分散配列 *a[]* を定義している．7 行目では，OpenACC **data** 指示文によって分散配列 *a[]* をアクセラレータに転送している．9〜13 行目では，まず XMP **loop** 指示文がループ文を各ノードに分割し，さらに OpenACC **parallel** 指示文が，XMP によって分割されたループ文をさらにアクセラレータで実行するようにスレッド分割を行う．なお，この場合，XMP **loop** 指示文と OpenACC **parallel** 指示文の順序は，どちらが先でもよい．

3.4 アクセラレータ間の直接通信

XACC では，XMP が提供する通信指示文に対して **acc** という節を追加することにより，アクセラレータ間の直接通信を表現することができる．

図 4 に XACC **gmove** 指示文の例を示す．**gmove** 指示文は，グローバルイメージを保ったまま，分散配列に対するデータ転送を行う指示文である．図 4 では，アクセラレータ上に存在する分散配列 *b[2]*〜*b[6]* までの 5 要素を，分散配列 *a[0]*〜*a[4]* にコピーしている．この例において，**acc** 節が無い場合は，ホスト上に存在する分散配列に対して，同様の処理が行われる．

他の通信指示文の例を下記に示す．

- **bcast** 指示文は **from** 節で指定されたノードのアクセラレータ上に存在するローカル変数に対してブロードキャスト通信を発生させる．下記の例では，ノード *p(2)* が持つローカル変数 *e* をブロードキャストしている．

```
1  #pragma xmp bcast (e) from p(2) acc
```

- **reduction** 指示文は集約演算を行う．下記の例では，**reduction** 指示文は全アクセラレータが持つローカル変数 *f* の総計を求めている．

```
1  #pragma xmp reduction (+:f) acc
```

- **reflect** 指示文は袖領域の同期を行う．まず **shadow** 指示文により，分散配列に対して袖を定義する．**shadow** 指示文はホストで実行する必要があるため，**acc** 節は必要ない．下記の例では，配列の上限と下限に 1 要素ずつの袖を定義している．**reflect** 指示文は隣接ノードの端の要素を **shadow** 指示文で定義した袖に転送する．

```
1  #pragma xmp shadow a[1:1]
2  ...
3  #pragma xmp reflect (a) acc
```

```
#pragma xmp template t(0:8)
#pragma xmp nodes p(3)
#pragma xmp distribute t(block) onto p
#pragma xmp align a[i] with t(i)
#pragma xmp align b[i] with t(i)
...
#pragma acc data copy(a, b)
{
#pragma xmp gmove acc
  a[0:5] = b[2:5];
}
```

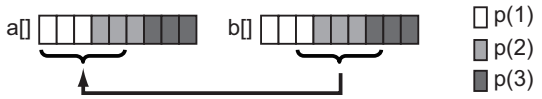

図 4 XcalableACC **gmove** 指示文の例 [10]

3.5 XcalableACC コンパイラの実装

我々は XACC，XMP，OpenACC および OpenMP に対応した Omni コンパイラ [20–22] を開発している．Omni コンパイラはベース言語（C 言語もしくは Fortran）と各指示文をランタイムの呼び出しに変換する source-to-source コンパイラである．

Omni XACC コンパイラは Omni XMP コンパイラを拡張することで作成している [6–8]．Omni XACC コンパイラの処理の流れを**図 5** に示す．最初に，ベース言語と XMP，XACC，OpenACC 指示文で記述されたコードは，OpenACC 以外の指示文がランタイムの呼び出しに置き換わる．次に変換されたコードが OpenACC コンパイラによってコンパイルされ，最終的に実行ファイルが生成される．汎用性を高めるため，コード変換では通常の OpenACC

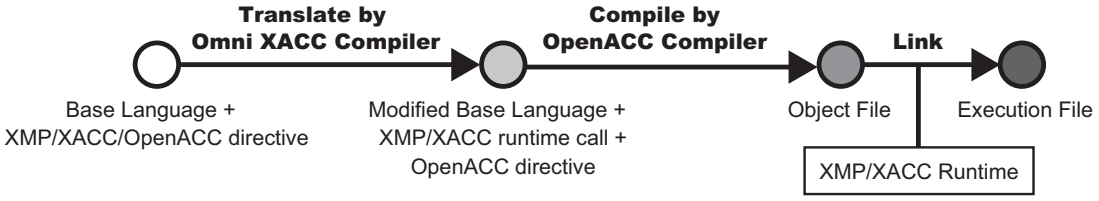

図 5 Omni XcalableACC コンパイラのコンパイルの流れ

表 1 評価環境（HA-PACS/TCA システム）

CPU	Intel Xeon-E5 2680v2 2.8 GHz x 2 Sockets
Memory	DDR3 SDRAM, 128GB, 59.7GB/s x 2
GPU	NVIDIA Tesla K20X x 4 GPUs, GDDR5 6GB x 4 and 250GB/s x 4[*1]
Interconnect	InfiniBand Mellanox Connect-X3 4 x QDR x 2rails 8GB/s
Compiler	Intel Compiler 15.0.2, Intel MPI 5.1.0, CUDA 6.5.14

のコードに変換される．すなわち，Omni XACC コンパイラは，どのような OpenACC コンパイラ（例えば，商用コンパイラの PGI，Cray，HMPP，オープンソースソフトウェアの Omni [20]，accULL [23]，OpenUH [24] と今後リリースされる予定の GNU コンパイラなど）であってもバックエンドコンパイラとして利用可能である．

Omni XACC コンパイラにおいて，3.4 節で述べたアクセラレータ間の直接通信には，(1)TCA [16,17] を用いたもの，(2)GPUDirect RDMA [18] を用いたもの，(3)MPI + CUDA を用いたもの，の 3 種類が実装されている．(1) は他の 2 つと比べて小さいレイテンシで通信を行うことが可能であるが，計算環境に TCA のシステムが必要になる．(2) は (3) と比較して性能に優れているが，MVAPICH2 [25] 等の GPUDirect RDMA に対応したソフトウェアおよびハードウェアのサポートが必要になる．(1) および (2) は，利用するソフトウェアがアクセラレータ間の直接通信を実現できるのに対し，(3) はアクセラレータ上のデータを CUDA を用いてホストメモリにコピーした後，MPI を用いて他ノードに転送する実装方法である．そのため，(3) は (1) と (2) に比べて性能は低いが，最も汎用的な実装である．

4. HPC Challenge ベンチマークの実装と性能評価

4.1 概要

文献 [26] において，XMP を用いた HPC Challenge ベンチマークの実装が行われている．そこで，XMP で記述された HPC Challenge ベンチマークをベースに，XACC 化を試みる．

性能評価には，筑波大学計算科学研究センターの HA-PACS/TCA システムを用いる．そのノード構成を**表 1** に示す．1 ノードあたり 4 枚の GPU が搭載されているため，1 ノードあたり 4 プロセスを割り当て，各プロセスに 1 枚の GPU を操作させる．

また，性能評価のための問題サイズの設定は，HPCC Awards Competition Class 2 の規約に従うものとした．その規約を下記に示す．なお，下記にある “システムメモリ” とは，ホスト上に存在するメモリのことであり，CPU のキャッシュメモリやアクセラレータのメモリなど，一時的に利用するメモリは含まれない．

- STREAM：計算に用いる 3 つの配列の合計サイズがシステムメモリの 1/4 以上
- HPL：係数行列のサイズがシステムメモリの 1/2 以上
- FFT：計算に用いる 2 つの配列の合計サイズがシステムメモリの 1/4 以上

4.2 実装

4.2.1 STREAM

STREAM は実効メモリバンド幅を測定するベンチマークである．具体的には，下記のように，あるスカラ値を掛けた配列と，別の配列を加算し，その結果をさらに別の配列に保存する速度を測定する．

```
1  double a[N], b[N], c[N];
2  ...
3  for(int j=0;j<N;j++)
4    a[j] = b[j] + scalar*c[j];
```

1 ノードに 4 プロセスを割り当てるため，配列 *a[]*，*b[]*，*c[]* のサイズの合計値は 8GB（$= 128GB/4/4$）以上である必要がある．しかしながら，Omni OpenACC コンパイラを用いた場合の各 GPU で確保可能な領域の最大値は約 5.4GB であった．そこで，ホストメモリ（システムメモリ）と GPU メモリの両方を用いることを考える．表 1 より，GPU メモリのバンド幅の方がホストメモリのバンド幅よりも広いため，GPU メモリに全体の 68%である約 5.4GB を，残りの 32%である約 2.6GB をホストが処理できるように STREAM を作成する．

図 6 に XACC で記述した STREAM のカーネル部分を示す．1 行目の XMP **nodes** 指示文により，このコードは並列に動作する．3 行目はアクセラレータに転送するデータサイズを計算し，5 行目の OpenACC **data** 指示文はデータをアクセラレータに転送している．8 行目と 21 行

[*1] メモリの一部は ECC ビットに利用されるため，ユーザ利用可能領域は 10%減少する [27]．また，メモリバンド幅も 10%低下すると仮定すると，1GPU あたりの性能は 225 GB/s となる．

```
1  #pragma xmp nodes p(*)
2  ...
3  int GPU_SIZE = array_elements * ratio; // ratio = 0.68
4  ...
5  #pragma acc data copy(a[0:GPU_SIZE], b[0:GPU_SIZE], c
       [0:GPU_SIZE])
6  {
7    for(int k=0;k<NTIMES;k++) {
8  #pragma xmp barrier
9      times[k] = -xmp_wtime();
10
11 #pragma acc parallel loop async
12     for(int j=0;j<GPU_SIZE;j++)
13       a[j] = b[j] + scalar*c[j];
14
15 #pragma omp parallel for
16     for(int j=GPU_SIZE;j<array_elements;j++)
17       a[j] = b[j] + scalar*c[j];
18
19 #pragma acc wait
20
21 #pragma xmp barrier
22     times[k] += xmp_wtime();
23   }
24 }
```

図 6 XcalableACC STREAM のコード

```
1  #pragma xmp nodes p(*)
2  ...
3  for(int k=0;k<NTIMES;k++) {
4  #pragma xmp barrier
5    times[k] = -xmp_wtime();
6
7  #pragma omp parallel for
8    for(int j=0;j<array_elements;j++)
9      a[j] = b[j] + scalar*c[j];
10
11 #pragma xmp barrier
12   times[k] += xmp_wtime();
13 }
```

図 7 XcalableMP STREAM のコード

目の XMP **barrier** 指示文は時間測定のためにバリア同期を行っている．11〜13 行目はアクセラレータが STREAM の計算をホストと非同期で行い，15〜17 行目はホストが STREAM の計算を行っている．19 行目は，11〜13 行目のアクセラレータの計算が終了するのを待つ．

XACC 版の STREAM は XMP 版の STREAM をベースに作成した．XMP 版の STREAM のカーネル部分を**図 7** に示す．図 6 と図 7 から，カーネル部分については，XMP 版の STREAM に対して 8 行のコードの追加（図 6 の 3, 5-6, 11-13, 19, 24 行目）と 1 行の既存コードの変更（図 6 の 16 行目）を行うのみで，XACC 版の STREAM を実装できたことがわかる．

4.2.2 High Performance Linpack

HPL は密行列の連立一次方程式を解く速度を計測するベンチマークであり，計算システムの演算性能が性能測定に大きく影響する．

XMP 版の HPL [26] では，高い性能とポータビリティを両立させるため，BLAS [28] を用いて実装している．そこで XACC 版の HPL においても，GPU のための BLAS である cuBLAS [29] や MAGMA [30] などを用いることを考えている．XACC 版の HPL の実装の参考には文献 [31] を用いる．文献 [31] では，演算密度の大きい BLAS 3 の関数である DGEMM と DTRSM を GPU 側で処理することで，全体的な高速化を行っている．さらに，計算対象となる行列を分割し，パイプライン的に GPU 側で BLAS 計算を行うことで，ホストとアクセラレータ間のデータ通信時間を隠蔽している．HPL では，STREAM と同様に，すべての係数行列を GPU のメモリに保存することはできないため，そのパイプラインによる計算手法は有効である．文献 [31] では，複雑な転送パターンである係数行列の縦方向の通信に MPI を用いているが，XACC 版の HPL では XMP 指示文を用いることができるため，より簡潔な実装になると見込んでいる．

上記のように実装する場合，あるホストが分解したパネルを別のホストのアクセラレータに送る必要がある．しかしながら，図 1 で示した通り，XACC はあるホストと異なるホストが持つアクセラレータとの直接通信はサポートしていないため，XMP 指示文と OpenACC 指示文の 2 つを用いて，その処理を行う必要がある．具体的には，ホストメモリにあるデータを XMP 指示文が異なるホストに送った後，OpenACC 指示文がそのデータをアクセラレータに送るという手順になる．この操作を 1 つの XACC 指示文で行えると HPL の実装がより簡易に行えると考えられる．そのような記法の実現ついては今後の課題である．

4.2.3 Fast Fourier Transform

FFT は 1 次元離散複素フーリエ変換に対する速度を計測するベンチマークであり，計算システムの演算性能と通信性能（全体全通信）の両方が性能測定に影響する．

XMP 版の FFT [26] では，FFTE [32] を用いて six-step FFT アルゴリズム [33, 34] を実装している．FFTE には CUDA Fortran で実装された GPU 用のルーチンがあるため，XACC 版の FFT ではその GPU 用のルーチンとホスト用のルーチンを同時に利用することを考えている．なお，XMP 版の FFT は XMP/Fortran で記述されているため，XACC 版の FFT も XACC/Fortran を用いる予定である．その場合，Fortran の OpenACC をサポートしている PGI コンパイラを Omni XACC コンパイラのバックエンドコンパイラとして用いる予定である．

4.3 STREAM の性能評価

表 1 に示した HA-PACS/TCA を用いて STREAM の性能評価を行う．HA-PACS/TCA の 2 つの CPU のコアの合計数は 20 であり，1 ノードにつき 4 プロセスを割り当てるため，OpenMP に対するスレッド数は 5 とした．ま

表 2 STREAM の性能結果

Nodes	Performance (GB/s)	
	XACC (/peak)	XMP (/peak, /peak only CPU)
1	297.44 (26.57%)	95.33 (8.52%, 79.84%)
2	609.27 (27.21%)	190.51 (8.51%, 79.78%)
4	1,250.50 (27.93%)	380.59 (8.50%, 79.69%)
8	2,456.81 (27.43%)	760.89 (8.50%, 79.66%)
16	4,732.01 (26.42%)	1,519.48 (8.48%, 79.54%)
32	9,458.91 (26.41%)	3,036.48 (8.48%, 79.47%)
64	18,895.76 (26.38%)	6,066.92 (8.47%, 79.39%)

た，性能比較のために，XMP で実装した STREAM の性能評価も行う．XMP で実装した STREAM では，1 ノードに 2 プロセスを割り当て，各プロセスのスレッド数は 10 とした．コンパイラは表 1 に示した Intel Compiler を用い，Intel Compiler に対するオプションは “-O3 -xAVX -opt-streaming-stores always” とした．

それぞれの性能結果を**表 2** に示す．なお，表 2 におけるピーク性能は，ECC を無効化した場合の値として 1119.4 GB/s（$= 59.7 \times 2 + 250 \times 4$）を用いた．また，XMP のピーク性能には，CPU のみのピーク性能である 119.4GB/s（$= 59.7 \times 2$）も追加した．表 2 より，XACC の方が XMP よりも約 3.1〜3.3 倍性能が高いことがわかる．

性能結果について考察する．4.2.1 節で記述したように，GPU は約 5.4GB，CPU は約 2.6GB を処理する．しかしながら表 1 より，GPU のメモリバンド幅は CPU のメモリバンド幅の 4 倍以上あるため，今回の計測条件では，CPU が性能のボトルネックになることがわかる．表 2 における XACC と XMP の性能差は約 3.1〜3.3 倍であり，全体の処理するメモリ量と CPU が処理するメモリ量の比も 8.0GB : 2.6GB = 約 3.1 : 1.0 であるため，表 2 の計測結果は妥当であると言える．

5. まとめと今後の課題

本稿ではアクセラレータクラスタのための新しいプログラミングモデル XACC の性能評価を行うため，HPCC ベンチマークの STREAM，HPL，FFT における実装方法について検討した．STREAM については実際に実装を行い，XMP 版の STREAM に対して 9 行のコード変更を行うことで XACC 版の STREAM を作成できることを示した．そして XACC 版の STREAM の性能評価を行った結果，XMP 版の STREAM と比較して約 3.1〜3.3 倍の性能向上を達成した．

今後の課題としては下記の 3 つが挙げられる．(1) XACC を用いた HPL，FFT の実装および性能評価を行う，(2) 4.2.2 節で示したあるホストと異なるホストが持つアクセラレータ間の直接通信をサポートする，(3) OpenACC は C++に対応しているが，XMP は対応していない．より多様なアプリケーションに対応するために，XACC および XMP を C++言語にも対応させる．

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] OpenACC. http://www.openacc-standard.org.
[2] OpenMP 4.0 Complete Specifications, 2013. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.
[3] OpenCL - The open standard for parallel programming of heterogeneous systems. http://jp.khronos.org/opencl.
[4] Parallel Programming and Computing Platform. http://www.nvidia.com/object/cuda_home_new.html.
[5] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI-The Complete Reference, Volume 1: The MPI Core*. MIT Press, Cambridge, MA, USA, 2nd. (revised) edition, 1998.
[6] Nakao Masahiro and Murai Hitoshi and Shimosaka Takenori and Tabuchi Akihiro and Hanawa Toshihiro and Kodama Yuetsu and Bokut Taisuke and Sato Mitsuhisa. XcalableACC: Extension of XcalableMP PGAS Language Using OpenACC for Accelerator Clusters. In *Proceedings of the First Workshop on Accelerator Programming Using Directives*, WACCPD '14, pp. 27–36, Piscataway, NJ, USA, 2014. IEEE Press.
[7] 中尾昌広, 村井均, 下坂健則, 田渕晶大, 塙敏博, 児玉祐悦, 朴泰祐, 佐藤三久. XcalableACC:OpenACC を用いたアクセラレータクラスタのための PGAS 言語 XcalableMP の拡張. 情報処理学会研究報告, Vol. 2014, No. 7, pp. 1–11, sep 2014.
[8] 田渕晶大, 中尾昌広, 村井均, 朴泰祐, 佐藤三久. 演算加速機構を持つ並列クラスタ向け PGAS 言語 XcalableACC の性能評価. 情報処理学会研究報告, Vol. 2015, , oct 2015.
[9] XcalableMP Specification Version 1.2, 11 2012. http://www.xcalablemp.org/spec/xmp-spec-1.2.pdf.
[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] Masahiro Nakao, Hitoshi Murai, Takenori Shimosaka, and Mitsuhisa Sato. Productivity and performance of the hpc challenge benchmarks with the xcalablemp pgas language. In *7th International Conference on PGAS Programming Model*, pp. 157–171, 2013.
[12] Dongarra, J. and Luszczek, P. Introduction to the HPC-Challenge Benchmark Suite. Technical report, ICL Technical Report, ICL-UT-05-01, (Also appears as CS Dept. Tech Report UT-CS-05-544), 2005.
[13] HPC Challenge Benchmarks. http://icl.cs.utk.edu/hpcc/.
[14] HPC Challenge Awards Competition. http://www.hpcchallenge.org.
[15] The Riken Himeno CFD Benchmark. http://accc.

riken.jp/2444.htm.

[16] Toshihiro Hanawa, Yuetsu Kodama, Taisuke Boku, and Mitsuhisa Sato. Tightly coupled accelerators architecture for minimizing communication latency among accelerators. In *IPDPSW '13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, pp. 1030–1039, 2013.

[17] Yuetsu Kodama, Toshihiro Hanawa, Taisuke Boku, and Mitsuhisa Sato. Peach2: Fpga based pcie network device for tightly coupled accelerators. In *Fifth International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART 2014)*, 2013.

[18] NVIDIA GPUDirect. https://developer.nvidia.com/gpudirect.

[19] Bailey, D.H. et al. THE NAS PARALLEL BENCHMARKS. Technical Report NAS-94-007, Nasa Ames Research Center, 1994.

[20] Akihiro Tabuchi, Masahiro Nakao, and Mitsuhisa Sato. A source-to-source openacc compiler for cuda. In *Euro-Par Workhops*, pp. 178–187, 2013.

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

[22] Omni XcalableMP Compiler. http://omni-compiler.org.

[23] Ruymn Reyes, Ivn Lpez-Rodrguez, Juan J. Fumero, and Francisco de Sande. accull: An openacc implementation with cuda and opencl support. In Christos Kaklamanis, Theodore S. Papatheodorou, and Paul G. Spirakis, editors, *Euro-Par*, Vol. 7484 of *Lecture Notes in Computer Science*, pp. 871–882. Springer, 2012.

[24] Xiaonan Tian, Rengan Xu, Yonghong Yan, Zhifeng Yun, Sunita Chandrasekaran, and Barbara Chapman. Compiling a high-level directive-based programming model for gpgpus. In *The 26th International Workshop on Languages and Compilers for High Performance Computing (LCPC 2013)*, 2013.

[25] The Ohio State University. MVAPICH. http://mvapich.cse.ohio-state.edu/.

[26] 中尾昌広, 村井均, 岩下英俊, 下坂健則, 朴泰祐, 佐藤三久. PGAS 言語 XcalableMP を用いた HPC Challenge ベンチマークの実装と評価. 情報処理学会研究報告, Vol. 2014-HPC-148, No. 21, pp. 1–12, feb 2015.

[27] Tesla k20x gpu accelerator. http://www.nvidia.com/content/PDF/kepler/Tesla-K20X-BD-06397-001-v07.pdf.

[28] http://www.netlib.org/blas/.

[29] cublas. https://developer.nvidia.com/cuBLAS.

[30] Matrix algebra on gpu and multicore architectures. http://icl.cs.utk.edu/magma/index.html.

[31] 遠藤敏夫, 額田彰, 松岡聡. スーパコンピュータ tsubame 2.0 における linpack 性能 1 ペタフロップス超の達成. 先進的計算基盤システムシンポジウム論文集, 第 2011 巻, pp. 373–380, may 2011.

[32] FFTE: A Fast Fourier Transform Package. http://www.ffte.jp.

[33] David H. Bailey. FFTs in external or hierarchical memory. *Journal of Supercomputing*, Vol. 4, pp. 23–35, 1990.

[34] Van Loan, C. *Computational Frameworks for the Fast Fourier Transform*. Society for Industrial and Applied Mathematics, 1992.