Vol.2016-HPC-155 No.26
2016/8/9

アクセラレータクラスタ用プログラミング言語 XcalableACCによる格子QCDの並列化の検討

中尾 昌広[1,a)]　村井 均[1]　小田嶋 哲哉[1]　田渕 晶大[3]　朴 泰祐[2,3]　佐藤 三久[1]

概要：我々はアクセラレータを搭載したクラスタシステムを対象とするプログラミングモデル XcalableACC を提案している．XcalableACC の記述性を評価するため，格子 QCD コードの実装を XcalableACC を用いて行った．その結果，逐次プログラムの格子 QCD コードに対して指示文を追加することで，アクセラレータを搭載したクラスタシステムで動作する並列格子 QCD コードを実装できることを示した．

Parallelization of Lattice QCD by using XcalableACC

Masahiro Nakao[1,a)]　Hitoshi Murai[1]　Tetsuya Odajima[1]　Akihiro Tabuchi[3]　Taisuke Boku[2,3]
Mitsuhisa Sato[1]

1. はじめに

高い電力性能比を持つ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）は計算資源として広く利用されている [1, 2]．アクセラレータクラスタに対する重要な課題として，ハードウェア性能を十分に引き出し，かつ性能ポータビリティに優れたプログラミングモデルの提供が挙げられる．

我々はアクセラレータクラスタにおけるプログラミングモデル XcalableACC（XACC）[3] を提案している．XACC は Partitioned Global Address Space（PGAS）言語 XcalableMP（XMP）[4–6] の拡張であり，XMP 指示文と OpenACC 指示文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にする．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．

本稿では，XACC を用いた格子 QCD コードの実装を検討する．

2. XcalableACC

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 OpenACC 指示文を用いることにより，XMP 指示文で定義した分散配列に対する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やループ文の並列処理など）を行うことができる．さらに XACC 指示文を用いることで，アクセラレータ間の直接通信を記述することができる．

本章では格子 QCD コードの実装に必要な XACC の機能を説明する．XACC は C 言語および Fortran の拡張として定義されているが，格子 QCD コードは C 言語で記述するため，C 言語の XACC の構文を説明する．

[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
[a)] masahiro.nakao@riken.jp

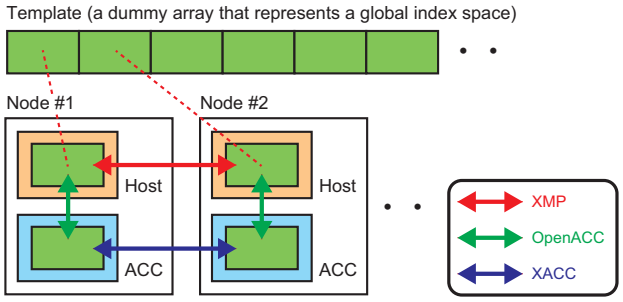

図 1 XcalableACC の概念図

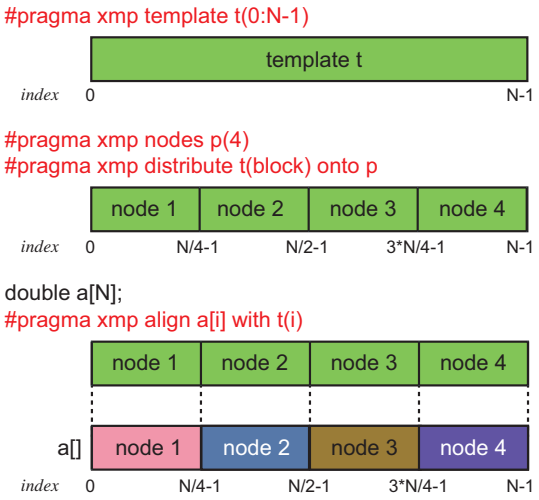

図 2 分散配列の定義 [5]

2.1 概要

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データ転送には XACC 指示文を用いる．

2.2 分散配列の定義

図 2 に分散配列の定義の例を示す．なお，XACC および XMP において指示文で分散を定義していない配列は，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．

```
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 }
```

図 3 XcalableACC のコード例

```
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  #pragma xmp shadow a[1:1]
7  ...
8  #pragma xmp reflect (a) acc
```

図 4 袖領域の同期を行うコード例

2.3 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．9〜12 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 loop** 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，この場合の XMP **loop** 指示文と OpenACC **parallel loop** 指示文の順序は問わない．

2.4 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 に袖領域の同期を行うコード例を示す．1〜5 行目は図 2 と同様である．6 行目の **shadow** 指示文は分散配列

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

図 5 集約演算を行うコード例

の上限と下限に 1 要素ずつの袖を定義している．8 行目の **reflect** 指示文は自ノードが持つアクセラレータ上の分散配列の端の要素を隣接ノードの袖に転送している．

図 5 に集約演算を行うコード例を示す．**reduction** 指示文はアクセラレータ上のローカル変数 f の総計を求めている．

2.5 XcalableACC コンパイラの実装

我々は XACC，XMP，OpenACC に対応した Omni コンパイラ [7–9] をオープンソースソフトウェアとして開発している．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 [3, 10]．Omni XACC コンパイラの処理の流れを**図 6** 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 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 [7]，accULL [11]，OpenUH [12] 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，2.4 節で述べたアクセラレータ間の直接通信には，(1)TCA [13, 14] を用いたもの，(2)GPUDirect RDMA [15] 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 (2) と比較してストライド通信の性能に優れているが，TCA の実装を搭載したハードウェアが必要になる．(2) は (3) と比較して全体的な性能に優れているが，MVAPICH2 [16] 等の GPUDirect RDMA に対応したソフトウェアおよびハードウェアが必要になる．(1) および (2) は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 を用いて他ホストに転送する実装方法である．ホストメモリに一旦コピーするため，(3) は他の 2 つと比べて性能は低いが，最も汎用的な実装である．

3. 格子 QCD の XcalableACC を用いた実装

3.1 格子 QCD とは

QCD（Quantum Chromo-Dynamics：量子色力学）は物質の最小単位であるクォークと，クォーク間における相互作用を結ぶグルーオン（糊粒子）を表す基本方程式である．格子 QCD は 4 次元（時間+XYZ 軸）の格子上で QCD のシミュレーションを行うものである．格子 QCD の並列化に関する研究は盛んに行われており，GPU を用いた高速化も行われている [17]．

本稿では，[18] および Bridge++ [19] を組合せて作成した逐次プログラムの格子 QCD コードを用いて XACC 化を検討する．全体的には [18] がベースとなっているが，Bridge++のコードも一部利用している．実装の手順は，まず逐次プログラムの格子 QCD コードを XMP を用いてクラスタ用に実装する．次に OpenACC 指示文および XACC 指示文を用いてアクセラレータクラスタ用に実装する．

3.2 格子 QCD の XcalableMP を用いた実装

格子 QCD の基本的な自由度はクォークとグルーオンであり，それぞれの物理量は複素数で表現される．クォークは 3 つの色を持つ “カラー” と 4 つのカラーを持つ “スピノル” を持つ．すなわちクォークは 4x3 の複素行列として表される．グルーオンは SU(3) 群の元であり，3x3 の複素行列として表される [20]．クォークは 4 次元格子の格子点上に定義されるのに対し，グルーオンは 4 次元格子の格子点を結ぶ格子線上に定義される．そこでクォークおよびグルーオンを表す構造体を**図 7** のように定義した．図中の NT・NZ・NY・NX は，時間・Z 軸・Y 軸・X 軸の要素数である．

次に XMP 指示文を用いて分散配列を定義する．本実装では時間次元と Z 軸次元のみ分割を行うことにする．また格子 QCD はステンシル計算であり，1 つ隣の要素からのみ影響を受ける．そのため，分散させる各次元は幅 1 の袖を持つように定義する．分散配列を定義するコードを**図 8** に示す．5 行目の **node** 指示文は 2 次元のノード集合を定義している．**node** 指示文中のアスタリスクは実行時にその要素数が決定することを示す．XMP の仕様書 1.2.1 [4] では最終次元のみアスタリスクが利用できると定義してあるが，omni コンパイラはこの仕様を拡張し，最終次元以外においてもアスタリスクを利用可能にしている．この場合，プログラム実行時に環境変数 **XMP_NODE_SIZEn**（n は 0 から始まる整数）により各次元のノード数を設定する．7 行目と 8 行目の **align** 指示文中のアスタリスクは，その次元の要素は各ノードで重複して持つことを意味する．9 行目と 10 行目の **shadow** 指示文中の 0 は，その次元は袖を持たないことを意味する．

格子 QCD は差分計算を各次元に対して行う．**図 9** に

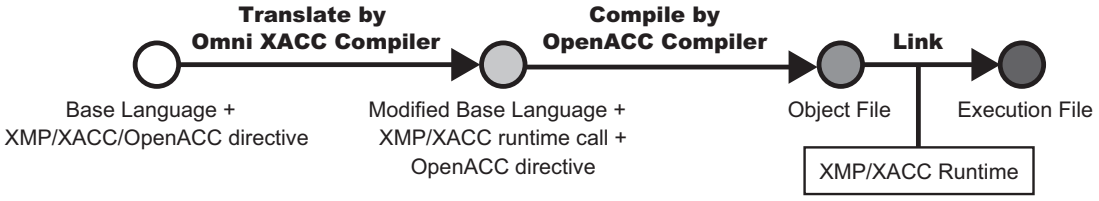

図 6 Omni XcalableACC コンパイラのコンパイルの流れ

```
1  typedef struct Complex {
2    double r;
3    double i;
4  } Complex_t;
5
6  typedef struct QCDSpinor {
7    Complex_t v[4][3];
8  } QCDSpinor_t;
9
10 typedef struct QCDMatrix {
11   Complex_t v[3][3];
12 } QCDMatrix_t;
13
14 QCDSpinor_t v[NT][NZ][NY][NX]; // quark
15 QCDMatrix_t u[4][NT][NZ][NY][NX]; // gluon
```

図 7 クォークおよびグルーオンを表す構造体

```
1  QCDSpinor_t v[NT][NZ][NY][NX]; // quark
2  QCDMatrix_t u[4][NT][NZ][NY][NX]; // gluon
3
4  #pragma xmp template t(0:NZ−1,0:NT−1)
5  #pragma xmp nodes p(∗,∗)
6  #pragma xmp distribute t(block, block) onto p
7  #pragma xmp align v[i][j][∗][∗] with t(j,i)
8  #pragma xmp align u[∗][i][j][∗][∗] with t(j,i)
9  #pragma xmp shadow v[1:1][1:1][0][0]
10 #pragma xmp shadow u[0][1:1][1:1][0][0]
```

図 8 格子 QCD コードにおける分散配列

```
1  #pragma xmp reflect(u,v) width(/periodic/1:1,1:1,0,0)
2  #pragma xmp loop (z,t) on t(z,t)
3    for(int t = 0; t < NT; t++){
4      for(int z = 0; z < NZ; z++){
5        for(int y = 0; y < NY; y++){
6          for(int x = 0; x < NX; x++){
7            ... = v[t−1][z][y][x].[0][0].r + ... ;
8            ... = u[3][t−1][z][y][x].[0][0].r + ... ;
```

図 9 格子 QCD コードにおけるループ文の並列処理（XcalableMP）

XMP 指示文を用いたループ文の並列処理を示す．このコードでは変数 t と z についてのみループ文の分割が行われる．ループ文の前に分散配列の袖の更新を **reflect** 指示文を用いて行う必要がある．格子 QCD は周期境界を持つので，**reflect** 指示文に width 節と periodic 修飾子を用いて．周期的な袖の更新を行う．

3.3 格子 QCD の XcalableACC を用いた実装

OpenACC 指示文と XACC 指示文を用いることで，3.2 節で作成したコードをアクセラレータに対応させる．図 9 を XACC 化すると**図 10** のようになる．図 9 の 1 行目では分散配列 u と v を各ノードのアクセラレータに転送している．3 行目の **reflect** 指示文には acc 節が追加されているため，アクセラレータ上に存在する分散配列 u と v の袖領域の更新が行われる．5 行目では OpenACC の **parallel loop** 指示文を用いることで，アクセラレータ上でループ文の分割が行われる．一般に NT の値はアクセラレータが持つスレッド数に比べて少ないため，collpase 節を用いてループ文をまとめて並列化する．

```
1  #pragma acc data copyin(u,v) ...
2  {
3  #pragma xmp reflect(u,v) width(/periodic/1:1,1:1,0,0) acc
4  #pragma xmp loop (z,t) on t(z,t)
5  #pragma acc parallel loop collapse(4)
6    for(int t = 0; t < NT; t++){
7      for(int z = 0; z < NZ; z++){
8        for(int y = 0; y < NY; y++){
9          for(int x = 0; x < NX; x++){
10           ... = v[t−1][z][y][x].[0][0].r + ... ;
11           ... = u[3][t−1][z][y][x].[0][0].r + ... ;
12 }
```

図 10 格子 QCD コードにおけるループ文の並列処理（XcalableACC）

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

4. まとめと今後の課題

本稿では XACC の記述性を評価するため，格子 QCD コードの実装を XACC を用いて行った．その結果，逐次プログラムの格子 QCD コードに指示文を追加することにより，アクセラレータクラスタで動作する格子 QCD コードを作成できることを示した．

今後の課題としては性能評価が挙げられる．本稿では記述していないが，OpenMP を用いて並列化した格子 QCD コードと OpenACC のみを用いて並列化した格子 QCD コードの性能を HA-PACS/TCA システム（スペックは **表 1** を参照．CPU および GPU は各 1 つのみを利用）を用いて比較した所，OpenMP の方がわずかに性能が高いと

いう結果になった．OpenACC の性能が低い原因はデータ形式にあると考えられるため [21]，そのためのコード変換を行う予定である．

謝辞 本稿で用いた格子 QCD コードは高エネルギー加速器研究機構の松古栄夫氏が作成したものを元に行っている．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] TOP500 Supercomputer Sites. http://www.top500.org.
[2] The Green500 List. http://www.green500.org.
[3] Masahiro Nakao and Hitoshi Murai and Takenori Shimosaka and Akihiro Tabuchi and Toshihiro Hanawa and Yuetsu Kodama and Taisuke Boku and Mitsuhisa Sato. 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, 2014.
[4] XcalableMP Specification. http://xcalablemp.org/specification.
[5] Masahiro Nakao and Jinpil Lee and 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 '12, pp. 402–409, 2012.
[6] Masahiro Nakao and Hitoshi Murai and 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.
[7] Akihiro Tabuchi and Masahiro Nakao and Mitsuhisa Sato. A Source-to-Source OpenACC Compiler for CUDA. In *Euro-Par Workhops*, pp. 178–187, 2013.
[8] Jinpil Lee. *A Study on Productive and Reliable Programming Environment for Distributed Memory System*. PhD thesis, University of Tsukuba, 2012.
[9] Omni Compiler. http://omni-compiler.org.
[10] 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.
[11] Reyes Ruymn and Lpez-Rodrguez Ivn and Fumero Juan J. and de Sande Francisco. accULL: An OpenACC Implementation with CUDA and OpenCL Support. In Kaklamanis, Christos and Papatheodorou, Theodore S. and Spirakis, Paul G., editor, *Euro-Par*, Vol. 7484 of *Lecture Notes in Computer Science*, pp. 871–882. Springer, 2012.
[12] Tian, Xiaonan and Xu, Rengan and Yan, Yonghong and Yun, Zhifeng and Chandrasekaran, Sunita and Chapman, Barbara. 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.
[13] Toshihiro Hanawa and Yuetsu Kodama and 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.
[14] Yuetsu Kodama and Toshihiro Hanawa and 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.
[15] NVIDIA GPUDirect. https://developer.nvidia.com/gpudirect.
[16] The Ohio State University. MVAPICH. http://mvapich.cse.ohio-state.edu/.
[17] Ken ichi Ishikawa and Ken ichi Ishikawa. Recent algorithm and machine developments for lattice qcd, 2008.
[18] http://research.kek.jp/people/matufuru/Research/Programs/Tuning_Cpp/Solv_Wilson_Cpp/.
[19] http://bridge.kek.jp/Lattice-code/.
[20] 土井淳. XcalableMP による格子 QCD の並列化と Blue Gene/Q における性能評価. Technical Report 28, 研究報告ハイパフォーマンスコンピューティング（HPC）, Dec. 2014.
[21] 土井淳. 格子 QCD における CPU と GPU の協調動作についての考察. Technical Report 11, 研究報告ハイパフォーマンスコンピューティング（HPC）, Jun. 2015.