Vol.2015-HPC-151 No.5
2015/9/30

PGAS 言語 XcalableMP における動的タスク機能の提案

津金 佳祐[1,a)]　中尾 昌広[2]　李 珍泌[2]　村井 均[2]　佐藤 三久[1,2]

概要：本稿では，動的なタスク並列機能を分散メモリ環境上で実行可能かつ簡易に記述可能とするために，Partitioned Global Address Space（PGAS）言語 XcalableMP（XMP）において tasklet 指示文の提案を行う．ノード内では OpenMP task 指示文の depend 節を用いて依存関係を記述したタスクの生成，実行を行い，ノードを跨ぐタスク間の依存関係は MPI によりタスク内で通信を発生させ，通信の完了により依存関係を表す．以上の処理を 1 行で記述可能とした指示文が tasklet 指示文である．XMP のリファレンス実装である Omni XMP Compiler は，XMP 指示文により記述されたコードを MPI コードへと変換を行う．そのため，提案手法の予備評価としてブロックコレスキー分解のコードを対象に，XMP コンパイラによるコード変換を想定した MPI+OpenMP コードを手動で作成し，性能・生産性の評価を行った．結果として，1 ノード実行と比較して最大 2.5 倍まで性能が向上した．また，指示文によるプログラミングモデルを導入することによって MPI+OpenMP のコードと比較してプログラムの行数を 71%に抑えたことから，tasklet 指示文によるプログラムの生産性の高さを示すことができた．

1. はじめに

近年，高性能計算分野においてチップ内に多くのコアを搭載するメニーコアプロセッサを用いた大規模並列システムが登場している．そのようなシステムにおけるプログラミングには，ノード内の多くのコアを効率よく使用することが出来る並列化方法の選択や，ノード間における最適な通信の記述に加え，異なるプログラミングモデルを組み合わせて用いる必要があり，プログラムが複雑になりやすい等，多くの問題がある．これらの問題を解決するために，ノード内の並列化方法としてタスク並列，ノード間のプログラミングモデルとして（Partitioned Global Address Space）PGAS モデルが注目されている．

タスク並列を記述可能なプログラミングモデルとして，OpenMP[1] がある，OpenMP におけるタスク並列は OpenMP 3.0 から登場した task 指示文で記述可能であり，再帰的構造や while ループ等の各スレッドでの演算が動的に決定する場合に用いられる．また，OpenMP 4.0 からは task 指示文の節として depend 節が登場し，タスク間での依存関係を記述可能となった．depend 節により記述された依存関係を基に動的にタスクが生成，実行されるため，従来のタスク全体での同期ではなくタスク間の細かい単位での同期とすることが可能である．そのため，多くのコアを持つメニーコアプロセッサにおいてタスク並列は，動的にコアへと演算を割り当てる点やタスク間のみの少ない同期コストにより，従来の parallel 指示文によるデータ並列と比較して性能向上が見込まれる．しかし，OpenMP は単一ノードでの実行を対象としているため，分散メモリ環境においては他のプログラミングモデルと組み合わせたハイブリッドな記述が必要とされる．

一方で，分散メモリ環境上でのプログラミングモデルとしては MPI が広く普及している．しかし，MPI はプロセス毎のデータの分散配置や複雑な通信の記述等，並列化を行う上での様々な処理手順を明示的に示す必要があるために，プログラミングの学習コストが高くソースコードが煩雑になりやすいといった生産性の低下が問題となっている．そこで，分散メモリ環境上での並列プログラミングをより容易にするために開発されたのが，PGAS 言語 XcalableMP（XMP）[2][3][4] である．XMP は，既存言語（C，Fortran）の拡張であり，OpenMP に似た指示文を用いてループの並列化やプロセス間通信を行うことが可能である．そのため，既存の逐次プログラムを大きく変更することなく容易に並列化を行うことができる．また，XMP は OpenMP と同時に記述可能であるため，プログラミングモデルを組み合わせることで，動的なタスク並列を記述することは可能である．しかし，分散メモリ環境において動的なタスク並列を行うためには，データの分散を考慮し

[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] 国立研究開発法人理化学研究所計算科学研究機構
RIKEN Advanced Institute for Computational Science
[a)] tsugane@hpcs.cs.tsukuba.ac.jp

© 2015 Information Processing Society of Japan

通信とタスクの依存関係を合わせて記述する必要がある．したがって，記述は複雑になりやすい．

以上の背景を踏まえ本稿では，タスク間の依存関係とノード間における通信を同時に記述可能とするため，XMP において新たな指示文として tasklet 指示文の提案を行う．実装対象の XMP のリファレンスコンパイラである Omni XMP Compiler[5] は，XMP 指示文により記述されたコードを MPI コードへと変換を行う．よって，tasklet 指示文の初期評価として Omni XMP Compiler によるコード変換を想定した MPI+OpenMP コードの作成を行い，性能・生産性の評価を行うことを目的とする．XMP における動的タスク機能は，XMP の次期仕様として PC クラスタコンソーシアムの並列プログラミング言語 XcalableMP 規格部会にて検討中であり，本提案はその一つの記法である．

本稿の構成を以下に示す．2 章は関連研究の紹介を行う．3 章では OpenMP のタスク並列，4 章では XMP の概要を紹介する．5 章は提案する指示文である tasklet 指示文の説明を行い，6 章では tasklet 指示文を用いたプログラムの評価を示す．7 章でまとめと今後の課題を述べる．

2. 関連研究

タスク並列を記述可能なプログラミングモデルとして Threading Building Blocks（TBB)[6] や Cilk Plus[7] がある．TBB は C++，Cilk Plus は C，C++言語を対象としている．本稿で対象とする XMP は C，Fortran を対象としたプログラミングモデルであるために今後の開発を考慮し OpenMP を用いた．

分散メモリ環境においてタスク並列を記述可能なプログラミングモデルとして StarPU[8] や QUARK-D[9] がある．StarPU は，INRIA により開発されている CPU や GPU 等を対象にタスク並列が可能なランタイムシステムである．記述方法としてタスク制御のために codelet と呼ばれる構造体を使用し，実行関数，実行リソースや依存関係等を記述し，タスク生成時に用いる方法をとる．QUARK-D は，テネシー大学により開発が進められている分散メモリ環境を対象としたタスク並列が可能なランタイムシステムである．QUARK-D が提供する API である QUARKD_Insert_Task を用いて StarPU 同様に実行関数，実行リソースや依存関係等を登録しタスク並列を行う．

タスク並列を記述可能なプログラミングモデルを用いて様々なプログラムの開発が進められている．例えば，INRIA が提供する KASTORS benchmark suite[10] では，OpenMP の task 指示文や depend 節を用いてポアッソン方程式をヤコビの反復法で解くソルバー，シュトラッセンのアルゴリズムを用いた行列積及び LU 分解を行うプログラム等を提供している．また Barcelona Supercomputing Center（BSC）では，OpenMP に似たプログラミングモデルである OmpSs[11] が開発されており，パッケージ内にてコレスキー分解のコードや stream 等を公開している．

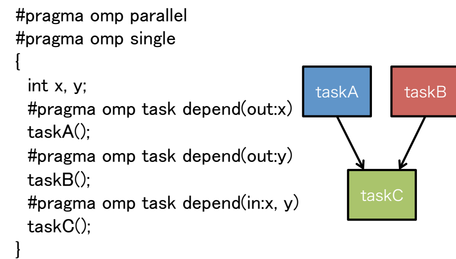

図 1 OpenMP task 指示文によるプログラミング例

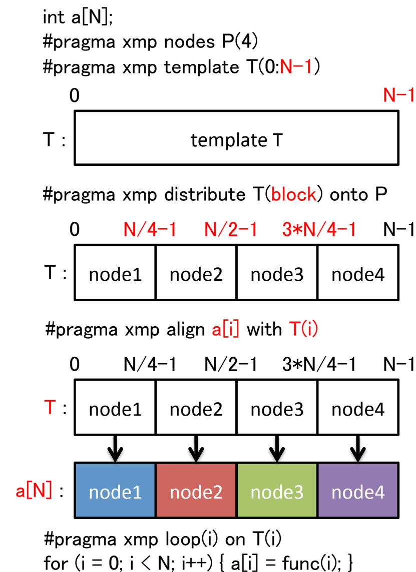

図 2 グローバルビューモデルによるプログラミングの例

3. OpenMP

OpenMP には多くの機能が指示文により提供されているが，本稿では task 指示文による記述方法のみの説明を行う．task 指示文は OpenMP 3.0 から登場したタスク並列を記述可能とする指示文であり，再帰的構造や while ループ等の各スレッドでの演算が動的に決定する場合に用いられる，また，OpenMP 4.0 から登場した depend 節により，タスク間の依存関係が記述可能となっている．depend 節には，in，out 及び inout の 3 種類が指定可能であり，合わせて変数または配列を記述する．また，依存関係を示す変数または配列は必ずしも task ブロック内で用いる必要はない．**図 1** に task 指示文と depend 節を用いたコード例とタスクフローを示す．例では 3 種類のタスクが生成される．依存関係 out が指定されている taskA，taskB は並列に実行され，taskA，taskB の depend 節で指定されている変数 x，y が in にて指定されている taskC は taskA，taskB の実行終了後に実行される．

4. PGAS 言語 XcalableMP

XMP は，次世代並列プログラミング言語検討委員会及

© 2015 Information Processing Society of Japan

```
#pragma xmp tasklet tasklet-format[, tasklet-format, ...] [on {node-ref | template-ref}]
  (structured-block)

  where tasklet-format is :
    dependence-type (array_name[, {node-ref | template-ref}[, tag]])

  and dependence-type is :
    in, out, or inout
```

図 4　XMP tasklet 指示文

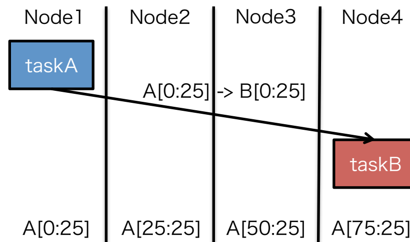

図 3　分散メモリ環境におけるタスクフローの例

```
1  int A[100], B[25];
2  #pragma xmp nodes P(*)
3  #pragma xmp template T(0:99)
4  #pragma xmp distribute T(block) onto P
5  #pragma xmp align A[i] with T(i)
6  /* ··· */
7  #pragma xmp tasklet out(A[0:25], T(75:99))
8  taskA();
9  #pragma xmp tasklet in(B, T(0:24)) out(A[75:25])
10 taskB();
11 #pragma xmp taskletwait
```

図 5　XMP tasklet 指示文によるプログラミング例

び PC クラスタコンソーシアム並列プログラミング言語 XcalableMP 規格部会により，仕様検討及び策定されている分散メモリ型 SPMD（Single Program Multiple Data）を実行モデルとする並列言語である．リファレンス実装である Omni XcalableMP Compiler[5] は，筑波大学と理化学研究所により開発が進められている．XMP の実行単位はノードである．XMP はプログラミングモデルとしてグローバルビュー・ローカルビューの二種類を提供している．グローバルビューモデルは典型的なデータ分散，通信を指示文で提供しており，ローカルビューモデルは Fortran 2008 から正式採用された Co-array Fortran[12] をベースとした coarray 記法を提供している．本稿では，グローバルビューモデルを用いるためその説明のみを行う．

グローバルビューモデルは，問題で扱うグローバルな配列を各ノードに分散する指示文を記述することで，並列実行を行うプログラミングモデルである．そのため，基本的には逐次実装に指示文を挿入するのみでの並列実装を行うことが可能である．また XMP と OpenMP は，XMP の仕様上同時に記述することが可能である．**図 2** にグローバルビューモデルによるプログラミング例を示す．ノード毎にデータの分散を行うには，テンプレートと呼ばれる仮想的なインデックス空間を用いる．まず，node，template 指示文により実行ノード（プロセス数）とテンプレートサイズを指定する．次に，テンプレートに対して distribute 指示文により分割方法（ブロック，サイクリック，ブロック・サイクリック及び，不均等ブロック）の指定を行い，align 指示文で対象の配列と分割されたテンプレートを対応付けることで，各ノードへとデータの分散を行う．これらの動作は全て指示文によるものであり，ユーザは各ノードへと分散されたデータの配置を意識することなく，分割したいループに対して記述された loop 指示文により，並列実行を行うことが可能となる．また，グローバルビューモデルによる並列プログラムは，基本的に XMP 指示文を無視することで逐次実装の C，Fortran 言語によるプログラムとして解釈することが可能である．

5. XcalableMP における動的なタスク並列機能を実現する指示文

本章では，提案する記述方法である tasklet 指示文と，コンパイラによるコード変換によって XMP コードからどのような MPI+OpenMP のコードへと変換されるかの説明を行う．また現在，tasklet 指示文は XMP 規格部会にて仕様検討が行われている段階であり，本稿で紹介する記述方法はその一つである．

5.1 tasklet 指示文

分散メモリ環境における動的なタスク並列機能を実現するためには，ノードを跨ぐタスク間の依存関係の記述が必要である．しかし，ノード内におけるタスクの依存関係の記述は OpenMP task 指示文の depend 節で表すことが可能だが，ノードを跨ぐタスク間の依存関係を表す機能は MPI，OpenMP 共に存在しない．そのため本稿では，ノード内で実行されるタスク内で MPI による通信を発生させ，その通信の完了を以って依存関係を表す方法をとる．**図 3** にノードを跨ぐタスク間の依存関係の例を示す．taskA と taskB に依存関係があり，taskB にてノード 1 が持つ値である配列 A[0:25][*1] を用いる場合，taskA の実行終了後に

[*1] C 言語における部分配列記法．配列 A のインデックス 0 から 25 要素を表す．

© 2015 Information Processing Society of Japan

```
1  #pragma omp parallel
2  #pragma omp single
3  {
4      if (T(0:24) を持つノードならば) {
5          #pragma omp task depend(inout:A[0:25])
6          {
7              MPI_Wait(···);
8          }
9          #pragma omp task depend(out:A[0:25])
10         {
11             taskA();
12         }
13         #pragma omp task depend(inout:A[0:25])
14         {
15             MPI_Isend(A, 25, T(75:99), tag, ···);
16         }
17     }
18     if (T(75:99) 持つノードならば) {
19         #pragma omp task depend(inout:B)
20         {
21             MPI_Irecv(B, 25, T(0:24), tag, ···);
22             MPI_Wait(···);
23         }
24         #pragma omp task depend(in:B) \
25                          depend(out:A[75:25])
26         {
27             taskB();
28         }
29     }
30     #pragma omp taskwait
31     MPI_Waitall(···);
32 }
```

図 6 XMP によるコード変換例

配列 A[0:25] の送信を行い，taskB の開始時に配列 B の受信を行う．そして受信完了後に taskB の処理を再開する．したがって，taskA と taskB の間に依存関係を指定することが可能である．

XMP は OpenMP と合わせて記述することが可能である．そのため既存の機能を組み合わせることで動的なタスク並列機能を実現可能である．XMP の task 指示文により実行ノードを決定，XMP の task ブロック内で OpenMP の task 指示文によりノード内でのタスクを生成し，OpenMP の task ブロック内にて XMP の通信機能を用いる．通信機能には分散配列に対する通信を表す指示文である gmove 指示文や各ノード内のローカルな配列に対する通信が記述できる coarray があげられる．しかし，通信の記述以外はほぼ MPI+OpenMP と同様の記述が必要なため生産性は高くはない．そのため，Omni XMP Compiler により MPI だけではなく MPI+OpenMP コードを生成させることで，上記の記述を指示文 1 行の tasklet 指示文とし，生産性の向上を目指す．

図 4 に tasklet 指示文の記述方法を示す．tasklet 指示文では，ユーザが OpenMP 同様の依存関係の記述に加え，ノードを跨いだ場合の通信相手の記述，受信データを格納するバッファや実行ノードを指定する．dependence-type は OpenMP 同様に依存関係 in，out 及び inout を記述し，合わせて XMP の分散配列，もしくは受信バッファに加え，通信相手を指定する XMP のノード集合またはテンプレート，タスク間の通信制御のためのタグが指定可能である．また，on 節により実行ノードの決定が行われ，on 節が省略された場合は依存関係 out，inout で指定された分散配列を持つノードでタスクが実行される．通信相手，タグの指定が省略された場合は OpenMP の depend 節同様に処理される．通信相手の記述をする場合には必ず送信，受信のタスクが対になる必要がある．

表 1 実験環境（COMA）

CPU	Intel Xeon E5-2670v2 × 2
	CPU（10 cores/CPU）× 2 = 20 cores
Memory	64GB
Interconnect	Infiniband FDR Connect-X3（56 Gbit/s）
OS	Red Hat Enterprise Linux Server 6.4
Compiler	Intel 15.0.2
MPI	Intel MPI 5.0.3
Intel MKL	11.2.2

```
1  double A[nt][nt][ts*ts];
2  /* ··· */
3  #pragma omp parallel
4  #pragma omp single
5  {
6      for (int k = 0; k < nt; k++) {
7          #pragma omp task depend(inout:A[k][k])
8          omp_potrf (A[k][k], ts, ts);
9
10         for (int i = k + 1; i < nt; i++) {
11             #pragma omp task depend(in:A[k][k]) \
12                              depend(inout:A[k][i])
13             omp_trsm (A[k][k], A[k][i], ts, ts);
14         }
15         for (int i = k + 1; i < nt; i++) {
16             for (int j = k + 1; j < i; j++) {
17                 #pragma omp task depend(in:A[k][i], A[k][j]) \
18                                  depend(inout:A[j][i])
19                 omp_gemm (A[k][i], A[k][j], A[j][i], ts, ts);
20             }
21             #pragma omp task depend(in:A[k][i]) \
22                              depend(inout:A[i][i])
23             omp_syrk (A[k][i], A[i][i], ts, ts);
24         }
25     }
26     #pragma omp taskwait
27 }
```

図 7 OpenMP task 指示文によるブロックコレスキー分解のコード

5.2 コード変換

tasklet 指示文の実装を行う XMP コンパイラは Omni XMP Compiler を想定する．Omni XMP Compiler は，XMP 指示文により記述されたコードを MPI コードへと source-to-source 変換を行うトランスレータである．その

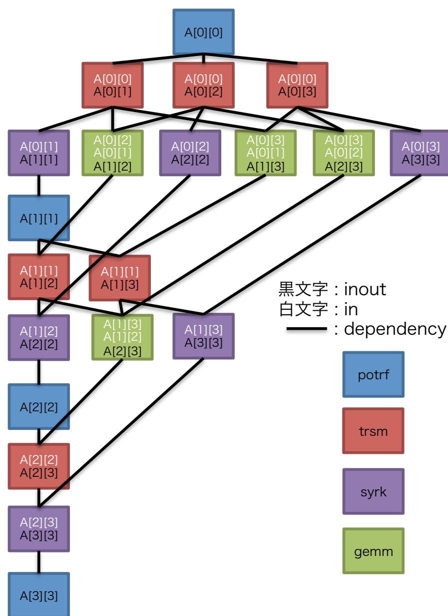

図 8 1 ノード実行におけるブロックコレスキー分解のタスクフロー

ため，XMP tasklet 指示文が Omni XMP Compiler によるコード変換でどのような MPI+OpenMP コードへと変換されるかを説明する．

図 5 に図 3 のタスクフローを tasklet 指示文で実装した場合の例を示し，**図 6** に XMP によるコード変換例を示す．taskA，taskB 共に on 節が省略されているため依存関係 out で指定された分散配列を持つノードのみが実行される．例の場合は taskA は A[0:25] を持つノード 1，taskB は A[75:25] を持つノード 4 が実行する．taskA は依存関係 out のため，MPI_Isend により分散配列 A[0:25] を T(75:99) を持つノード 4 へと送信する．また，タスク間の通信には MPI のノンブロッキング通信を用いるため，既に A[0:25] が送信途中である場合も想定される．したがって，MPI_Wait を用い通信完了を保証してからタスクを実行する．taskB は依存関係 in により指定されたテンプレート T(0:24) を持つノード 1 から，MPI_Irecv により受信し受信バッファである配列 B へと格納する．通信完了保証後に taskB が実行される．また，taskletwait 指示文は生成されたタスクの実行終了を保証する指示文であり，加えて全ての通信の完了を保証する．

6. 評価

XMP の tasklet 指示文を用いたプログラムの評価には筑波大学計算科学研究センターの COMA[13] を利用した．そのノード構成を**表 1** に示す．1 ノードあたり 1MPI プロセスを割り当て最大 16 ノード 16MPI プロセスを用いる．また，ノード内ではスレッド数を 1 から 16 へと変動させ評価を行う．

```
1  double A[nt][nt][ts*ts], B[ts*ts], C[nt][ts*ts];
2  #pragma xmp node P(*)
3  #pragma xmp template T(0:nt-1)
4  #pragma xmp distribute T(cyclic) onto P
5  #pragma xmp align A[*][i][*] with T(i)
6  /* ... */
7  for (int k = 0; k < nt; k++) {
8      #pragma xmp tasklet inout(A[k][k], T(k+1:nt-1))
9      omp_potrf (A[k][k], ts, ts);
10
11     for (int i = k + 1; i < nt; i++) {
12         #pragma xmp tasklet in(B, T(k)) \
13                             inout(A[k][i], T(i+1:nt-1))
14         omp_trsm (B, A[k][i], ts, ts);
15     }
16     for (int i = k + 1; i < nt; i++) {
17         for (int j = k + 1; j < i; j++) {
18             #pragma xmp tasklet in(A[k][i]) in(C[j], T(j)) \
19                                 inout(A[j][i])
20             omp_gemm (A[k][i], C[j], A[j][i], ts, ts);
21         }
22         #pragma xmp tasklet in(A[k][i]) inout(A[i][i])
23         omp_syrk (A[k][i], A[i][i], ts, ts);
24     }
25 }
26 #pragma xmp taskletwait
```

図 9 XMP tasklet 指示文によるブロックコレスキー分解のコード

6.1 ブロックコレスキー分解

ブロックコレスキー分解を解くコードは BSC OmpSs[11] が提供するコードを用いる．ブロックコレスキー分解は以下の 4 つの処理に分けられる．

(1) コレスキー分解（potrf）
(2) 三角行列を係数行列とする行列方程式を解く（trsm）
(3) 対称行列のランクを更新（syrk）
(4) 行列積 (gemm)

上記の処理は全てブロック単位で行われる．**図 7** に OpenMP で実装したブロックコレスキー分解のコードと**図 8** にブロック数が 4×4 の場合の実行時のタスクフローを示す．それぞれの処理には依存関係があり，例えば k = 0 の場合には，potrf(A[0][0]) が終了するまで trsm は実行できない．また，trsm は trsm(A[0][0], A[0][1])，trsm(A[0][0], A[0][2])，trsm(A[0][0], A[0][3]) を並列に実行可能である等，空いているスレッドに依存関係を満たしたタスクが割り当てられ順次実行される．

XMP tasklet 指示文により実装されたブロックコレスキー分解のコードを想定した MPI+OpenMP コードの評価を行った．行列サイズは 4096×4096 と 8192×8192 の 2 種類を用いる．分割方法は 1 次元のサイクリック分割とする．また，ブロックコレスキー分解の 4 つの処理には全て Intel Math Kernel Library（MKL）を用いる．まず，1 ノードで実行を行いブロックコレスキー分解を行う場合における最適なブロックサイズと生成されるタスク数の関係

© 2015 Information Processing Society of Japan

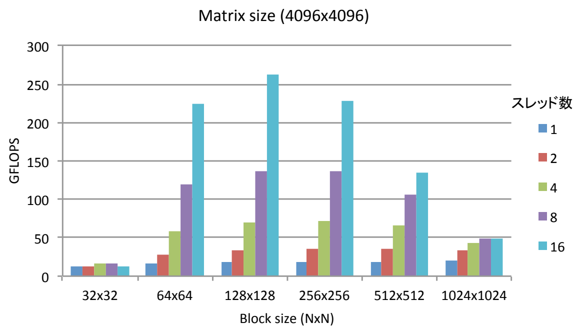

図 10 1 ノード実行におけるブロックコレスキー分解の評価（行列サイズ 4096×4096）

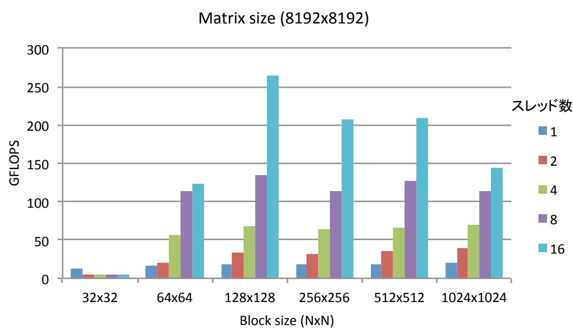

図 11 1 ノード実行におけるブロックコレスキー分解の評価（行列サイズ 8192×8192）

表 2 ブロックコレスキー分解のコード行数

	OpenMP	MPI+OpenMP	XMP
演算部分（指示文を除く）	16	97	14
OpenMP 指示文	7	11	-
XMP 指示文	-	-	10
その他	388	477	394

の調査を行う．ブロックサイズを 32×32 から 1024×1024，スレッド数を 1 から 16 へと変動させ評価を行った．**図 10** に行列サイズ 4096×4096，**図 11** に行列サイズ 8192×8192 の評価を示す．結果としてどちらの行列サイズの場合もブロックサイズ 128×128 が最適であることがわかった．次にブロックサイズを 128×128 とし，最大で 16 ノードを用いて各ノードでのスレッド数を 1 から 16 へと変動させた場合の評価を行った．**図 12**，**図 13** にその評価を示す．行列サイズ 4096×4096 の場合，ノード数の増加と共に性能向上が見られず，特にプロセス数が 16 のスレッド数が 16 の場合に大きな性能低下が見られた．行列サイズが小さく十分なタスク数が生成されないために性能が低下していると思われるが，詳しい原因は現在調査中である．行列サイズ 8192×8192 の場合は，ノード数の増加と共に性能が向上し，1 ノード実行と比較して 16 ノードで最大 2.5 倍の性能向上が見られた．

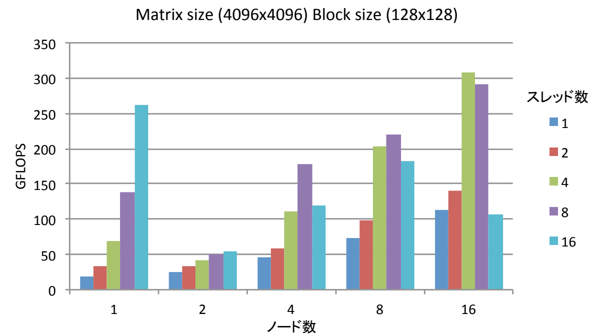

図 12 1-16 ノードを用いたブロックコレスキー分解の評価（行列サイズ 4096×4096）

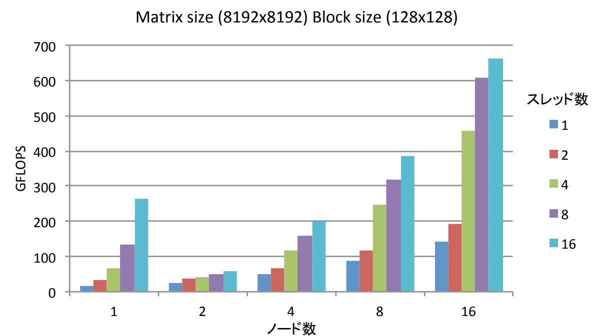

図 13 1-16 ノードを用いたブロックコレスキー分解の評価（行列サイズ 8192×8192）

6.2 生産性

XMP tasklet 指示文は，ノード内でのタスク並列に加えノード間での依存関係を満たすための通信を 1 行で記述できる．そのため，従来の OpenMP のみの実装とほぼ同等なコードとなっている．**表 2** に OpenMP のみ，MPI+OpenMP 及び XMP による実装のコード行数を示す．ブロックコレスキー分解の演算部分は，図 7 と**図 9** より XMP による実装は OpenMP のみの実装と同等である．また，OpenMP よりも XMP 実装がコード行数の増加が見られるのは，実行ノード数や領域分割等，XMP の初期化に用いる XMP 指示文とブロックコレスキー分解の演算で用いる受信用のローカルバッファの確保があるためである．MPI+OpenMP と XMP 実装のコード行数を比較すると 71%に抑えることが出来た．以上より，XMP tasklet 指示文によるプログラムの生産性は高いと言える．

7. おわりに

本稿では，動的なタスク並列機能を分散メモリ環境上でも実行可能かつ簡易に記述可能とするために，PGAS 言語 XcalableMP において tasklet 指示文の提案を行った．OpenMP によるノード内の依存性の記述に加え，XMP のノード集合またはテンプレートを用いることでノードを跨ぐタスク間の依存関係を通信の送受信の完了により記述可能とした．tasklet 指示文の初期評価として，ブロックコレ

© 2015 Information Processing Society of Japan

スキー分解のコードを対象とし，Omni XMP Compiler によるコード変換を想定した MPI+OpenMP コードを手動で作成し，性能・生産性の評価を行った．結果として，1 ノードの性能と比較すると最大で 2.5 倍の性能向上が見られた．また，MPI+OpenMP と XMP 実装のコード行数を比較すると 71%に抑えることが出来たことから，生産性の高さも示すことが出来た．

今後の課題として，他の分割方法による評価，性能低下の原因の調査や最適化があげられる．また，実際に Omni XMP Compiler に対して tasklet 指示文の実装を行い，性能・生産性の評価を行うことや，Intel Xeon Phi を用いた評価を行うこともあげられる．

謝辞 本研究の一部は，理化学研究所計算科学研究機構と筑波大学計算科学研究センターの共同研究「ポスト京の並列プログラミング環境およびネットワークに関する研究」による．また本研究の評価は，筑波大学計算科学研究センターの平成 27 年度学際共同利用プロジェクト「アクセラレータクラスタにおける高生産言語 XcalableACC の開発と評価」（代表者：中尾昌広）を利用して得られたものである．

参考文献

[1] OpenMP Application Interface version 4.0, 2013, http://openmp.org/wp/
[2] XcalableMP specification version 1.2.1, 2014, http://www.xcalablemp.org/download/spec/xmp-spec-1.2.1.pdf
[3] Jinpil L and Mitsuhisa S, Implementation and Performance Evaluation of XcalableMP : A Parallel Programming Language for Distributed Memory Systems, 39th Annual International Conference on Parallel Processing (2010).
[4] Masahiro N, Jinpil L, Taisuke B, and Mitsuhisa S, Productivity and Performance of Global-view Programming with XcalableMP PGAS Language, CCGrid 2012 - The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Ottawa, Canada, May, 2012.
[5] Omni Compiler Project, http://omni-compiler.org/
[6] Intel Threading Building Blocks, https://www.threadingbuildingblocks.org/
[7] Intel Cilk Plus, https://www.cilkplus.org/
[8] C Augonnet and R Namyst, A unified runtime system for heterogeneous multicore architectures. In Proceedings of the International Euro-Par Workshops 2008, HPPC'08, volume 5415 of LNCS, August 2008.
[9] YarKhan A, Dynamic Task Execution on Shared and Distributed Memory Architectures, PhD disseration, Major Advisor: Jack Dongarra, University of Tennessee, Dec. 2012.
[10] INRIA, KASTORS benchmark suite version-1.1, 2015, https://gforge.inria.fr/projects/kastors/
[11] Duran A, Ayguade E, Badia RM, Labarta J, Martinell L, Martorell X, Planas J, Ompss: a proposal for programming heterogeneous multi-core architectures. Parallel Process Lett 21(2):173-193
[12] ISO/IEC 1539-1:2010, Information technology – Programming languages – Fortran –, 2010.
[13] 筑波大学計算科学研究センタースーパーコンピュータ COMA(PACS-IX) について , http://www.ccs.tsukuba.ac.jp/files/coma-general/coma_outline.pdf

© 2015 Information Processing Society of Japan