Vol.2017-HPC-158 No.17
2017/3/9

KNLメニーコア・プロセッサにおけるPGAS言語XcalableMPアプリケーションの性能評価

津金 佳祐[1,a)]　田渕 晶大[1]　李 珍泌[2]　村井 均[2]　朴 泰祐[3,1]　佐藤 三久[2,1]

概要：近年，高性能計算分野において消費電力性能比が良いことからチップ内に多くのコアを搭載したメニーコアプロセッサが注目を集めている．筑波大学と東京大学による最先端共同HPC基盤施設（JCAHPC）が運用するOakforest-PACS（OFP）システムでは，Intelのメニーコアプロセッサである Knights Landing（KNL）が採用されている．そのようなメニーコア環境における高性能・高生産性言語として Partitioned Global Address Space（PGAS）言語 XcalableMP（XMP）が提案されている．本稿では，OFPシステムにおいてKNLプロセッサ上でのXMPアプリケーションの評価を行い，KNLプロセッサにおける性能特性に関する知見を示す．XMPの性能評価では，対象のアプリケーションを Himeno Benchmark と HPC Challenge Benchmark の Fast Fourier Transform（FFT）とした．OFPの64ノードを用いて評価を行ったところ，XMP実装は Himeno Benchmark で MPI実装の97%の性能，FFTでは98%の性能が得られ，どちらも MPI実装とほぼ同等の性能を得ることができた．現在，XMPはメニーコア対応に向けたバージョン2.0の仕様検討が進められている．XMP2.0では，提案の一つとして比較的性能の低い個々のコアに対して，Omni-Path Architecture など高い通信バンド幅を持つインターコネクトの性能を引き出すために，従来のプロセス単位ではなくスレッド単位による通信が検討されている．XMP2.0に向けた評価として，単一スレッドと複数スレッドが通信可能な Cholesky Factorization を実装し予備性能評価を行った．複数スレッド通信による Cholesky Factorization の性能は，単一スレッド通信の場合と比較して77%の性能に留まり，今後性能低下の原因を調査する予定である．

1. はじめに

近年，高性能計算分野において消費電力性能比が良いことからチップ内に多くのコアを搭載したメニーコアプロセッサが注目を集めている．代表的なメニーコアプロセッサとしては Intel Xeon Phi が挙げられる．Xeon Phi は，x86 に互換性のある命令セットを持つため，基本的に Xeon プロセッサ上で動作していたアプリケーションは，コード修正なしに実行可能という特徴を持つ．2016年6月に Xeon Phi の第2世代プロセッサである Knights Landing（KNL）プロセッサが発表され，同年11月に公開されたスーパーコンピュータの性能ランキングである Top500 [1] において，5位の National Energy Research Scientific Computing Center（NERSC）の Cori システムや，6位の Joint Center for Advanced HPC：最先端共同HPC基盤施設（JCAHPC）[2] の Oakforest-PACS（OFP）システムに採用されるなど，今後 KNL を搭載した大規模な並列システムはより増加すると考えられる．

上記のメニーコアシステムにおけるプログラミングは Message Passing Interface（MPI）と OpenMP を組み合せたハイブリッドな記述がデファクトスタンダードとなりつつある．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記述など，並列化のための様々な処理手順を明示的に示す必要があり，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されている．そこで，分散メモリ環境上の並列プログラミングをより容易にするために開発が進められているのが Partitioned Global Address Space（PGAS）言語 XcalableMP（XMP）[3–5] 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ノード間通信を記述できる．また，OpenMP とのハイブリッドな記述も可能であることから，メニーコアシステムにおけるプログラミング言語としても十分な性能・生産性が期待できる．

[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
[3] 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba
[a)] tsugane@hpcs.cs.tsukuba.ac.jp

そこで本稿では，KNL プロセッサ上で XMP アプリケーションの評価を行い，KNL プロセッサにおける性能特性に関する知見を示す．対象のアプリケーションである Himeno Benchmark [6] と HPC Challenge Benchmark の Fast Fourier Transform（FFT）[7,8] をメニーコアシステム向けに実装し，MPI 実装と比較をすることで性能評価を行う．評価環境には，筑波大学計算科学研究センターと東京大学情報基盤センター共同の JCAHPC が運用する OFP システムを用いる．OFP システムは，Omni-Path Architecture（OPA）をインターコネクトとした，2017 年 2 月現在，世界最大規模のメニーコアシステムである．また，XMP は OFP システムにデフォルトでインストールされる予定のプログラミング言語である．

現在，XMP はメニーコアに向けた新たな仕様であるバージョン 2.0 [9,10] の検討が進められている．提案の一つとして比較的性能の低い個々のコアに対して，OPA や InfiniBand EDR など高い通信バンド幅を持つインターコネクトの性能を十分に引き出すために，従来のプロセス単位ではなくさらに細粒度なスレッド単位での通信による実装が検討されている．そこで，XMP2.0 に向けた評価として，単一スレッドと複数スレッドが通信可能な Cholesky Factorization を MPI+OpenMP で実装し予備性能評価を行う．

本稿の構成を以下に示す．2 章は KNL の詳細，3 章は XMP の概要，及びグローバルビューモデルを説明する．4 章では XMP による各アプリケーションの実装を示し，5 章で XMP アプリケーションの性能評価を行う．6 章で XMP2.0 の概要と XMP2.0 に向けた予備性能評価を行い，最後に 7 章でまとめと今後の課題を述べる．

2. Knights Landing（KNL）メニーコアプロセッサ

KNL [11] は Intel Xeon Phi の第 2 世代プロセッサであり，Knights Corner（KNC）の後継機である．KNC は GPU と同様に PCI Express 接続のアクセラレータ型で，Xeon などのホストプロセッサを必要とするが，KNL は自身がブート可能となったため KNL のみをホストプロセッサとするシステムを構築可能となった．また，高バンド幅な 3D 積層メモリである MCDRAM の搭載や AVX512 命令のサポート，さらにコア間はリングバス接続から 2 次元メッシュネットワークになるなど，KNC から多くの変更点がある．

図 1 に KNL のチップ内構成を示す．KNL は 2 コアを 1 タイルとして扱う．タイルは 1MB の L2 キャッシュとコア毎に 2 つの Vector Processing Unit（VPU）で構成され，タイル間は 2 次元メッシュネットワークで接続される．各コアはハイパースレッディングにより最大 4 スレッドまで動作することから，OFP，Cori システムなどで用いられている Intel Xeon Phi 7250 では 34 タイル，68 コアで最大 272 スレッドによる実行が可能である．MCDRAM は 2GB ずつ合計で 16GB がチップ内に搭載され，チップ両端には DDR4 メモリチャネルが 3 つずつ合計 6 チャネルある．

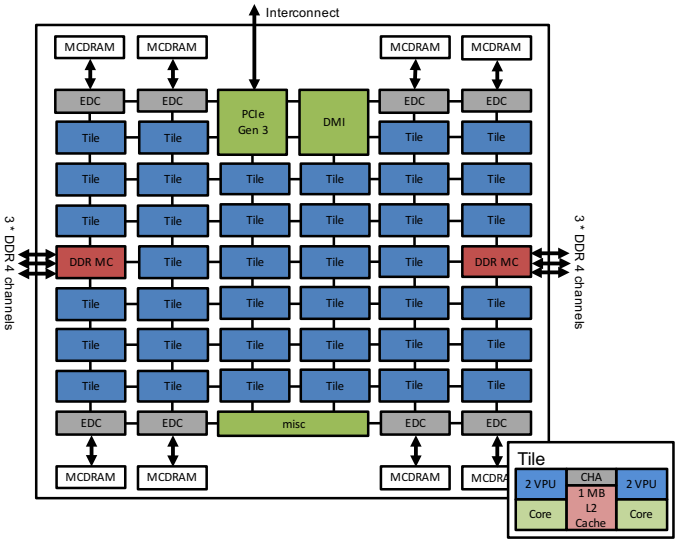

図 1 KNL（Knights Landing）プロセッサの構成．

KNL は 2 次元メッシュネットワークの扱いに All-to-All，Quadrant，Sub-NUMA Clustering（SNC4）の 3 種類のクラスタリングモードを提供している．All-to-All mode は，チップ内の全てのコアを単一プロセッサとして扱うため，メモリアクセスによってはコアとメモリ間のパスが長くなる．Quadrant mode は，All-to-All mode と同様に単一プロセッサとしてコアを扱うが，ネットワークは仮想的に 4 分割され各領域に最も近いメモリにデータが配置される．従って，ユーザはデータのメモリ配置を意識することなくプログラミングすることが可能である．SNC4 はチップを仮想的に 4 分割し，4 ソケットの Xeon プロセッサのように 4 つの NUMA ノードとして扱う．基本的に分割された各領域に閉じたメモリアクセスとなるため，`numactl` コマンドで NUMA ノードを扱う手間はあるが高い性能が期待される．さらに KNL は，MCDRAM と DDR4 メモリの扱いのために 3 種類のメモリモード（Flat，Cache，Hybrid）を提供している．Flat mode は，MCDRAM と DDR4 メモリが異なるメモリアドレス上に配置され，`numactl` コマンドなどにより明示的に割り当てる．Cache mode は，MCDRAM を DDR4 メモリの L3 キャッシュのように扱う．そのため，明示的に MCDRAM を利用することはできない．Hybrid mode は，MCDRAM を分割し Flat，Cache mode のそれぞれを利用可能とする．

3. PGAS 言語 XcalableMP

3.1 概要

XMP は，次世代並列プログラミング言語検討委員会及び PC クラスタコンソーシアム並列プログラミング言語

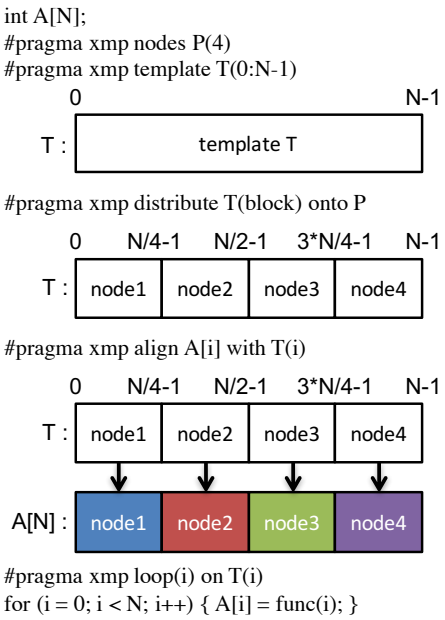

図 2 グローバルビューモデルのプログラム例.

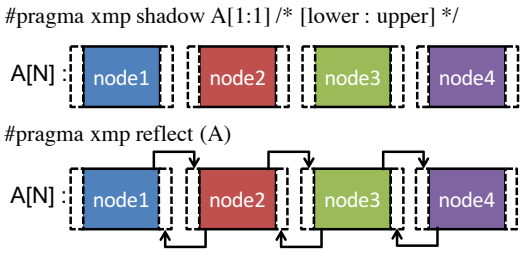

図 3 shadow，reflect 指示文による 1 次元配列の袖領域通信の例.

XcalableMP 規格部会により，仕様検討及び策定が行われている分散メモリ型 Single Program Multiple Data（SPMD）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 は，理化学研究所と筑波大学による Omni Compiler プロジェクト [12] により開発が進められており，XMP 指示文が挿入された C，Fortran コードを MPI で記述されたランタイム呼び出しへと変換する source-to-source なトランスレータである．MPI は 0 オリジンで “プロセス” を実行単位としているが，XMP は 1 オリジンで “ノード” としている．XMP はプログラミングモデルとしてグローバルビューとローカルビューの 2 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や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 より正式採用された coarray と互換性がある片側通信をサポートしている．他にも XMP は様々な指示文やランタイム関数を提供しているが，本稿では実装に用いたグローバルビューのみ説明する．

3.2 グローバルビューモデル

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挿入することで並列実行を可能とするプログラミングモデルである．従って，基本的に逐次プログラムに指示文を挿入するのみで並列プログラムを実装できる．**図 2** にグローバルビューモデルのプログラム例を示す．まず，`nodes` 指示文により実行ノード集合を定義する．数値の記述により静的に実行ノード数を指定できるほか，“*” とした場合は実行時に XMP ランタイムが自動的に判断する．`template` 指示文はテンプレートを定義する．XMP はテンプレートによりデータや処理の分散を記述する．図 2 では分散する配列 A のサイズに合わせてテンプレート長を指定している．次に，テンプレートに対して `distribute` 指示文で分割方法（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）を指定し，`align` 指示文により対象の配列と分散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分散を行う．分散されたデータを用いる for 文では `loop` 指示文を挿入することで，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，並列実行が可能である．基本的にグローバルビューモデルによる並列プログラムは，指示文追加による実装であるため XMP コンパイラが無い環境では逐次実装の C，Fortran コードとして実行できる．

XMP は典型的な通信のための指示文を提供している．`shadow`，`reflect` 指示文は，ステンシル演算などで広く用いられている袖領域通信を実行する指示文である．**図 3** は，XMP によりブロック分割された 1 次元配列の袖領域通信の例である．`shadow` 指示文により，各ノードに分散された配列の上端・下端に任意幅の袖領域を確保する．例の場合は上端・下端ともに一要素ずつ確保される．`shadow` 指示文で指定された配列を `reflect` 指示文で指定することで各ノードが持つ袖領域の値を更新する．

4. 実装

本稿では，KNL 上で評価を行うアプリケーションとして Himeno Benchmark と HPC Challenge Benchmark の Fast Fourier Transform（FFT）を対象とする．Himeno Benchmark，FFT の XMP 実装 [13] は既に行われているため詳細なコードの説明は省略する．本章では KNL プロセッサを持つクラスタに向けた XMP+OpenMP，MPI+OpenMP による実装のみを示す．

OpenMP [14] は仕様 4.0 よりプログラムの SIMD 化を促進するために `simd` 指示文や `simd` 節が追加された．使用例として `for` 指示文の節として指定することで，スレッド並列化されたループをスレッド単位で SIMD 化するようにコンパイラに知らせる．基本的に本実装では，`parallel for`，`for` 指示文が記述されたループに対して全て `simd` 節を追加した．また，KNL のようにコアあたりの性能が低いプロセッサにおいて for ループに `parallel for` 指示文を記述した場合，ループ単位で発生するスレッドの fork-join によるオーバヘッドが性能に与える影響が大きいと考えら

表 1　実験環境

	Oakforest-PACS
CPU	Intel Xeon Phi 7250 (68 cores)
Memory	16GB（MCDRAM）+ 96GB（DDR4）
Interconnect	Intel Omni-Path Architecture
Software	Intel Compiler ver. 17.0.1
	Intel MPI Library 2017 Update 1
	Intel MKL 2017 Update 1
	Omni XMP Compiler ver. 1.1.0

れる．そこで，出来る限り演算領域全てを `parallel` 領域としループ毎に `for` 指示文を追加した．

5. 性能評価

各アプリケーションの評価には OFP システムを用いる．1 ノードあたりの構成やソフトウェアのバージョンは**表 1** に示す通りである．まず，予備評価として OFP システムのインターコネクトである OPA の通信性能の評価を示す．その後，各アプリケーションの XMP，MPI 実装の性能を示す．以降の評価では MPI の実行単位に合わせて全て “プロセス” と表記し，“ノード” は 1KNL プロセッサを指す．以下に調査項目を示す．

- KNL のメモリモード（Flat，Cache mode）
- コアあたりのスレッド数（1–4 スレッド/コア）
- 64，68 コア，及びコア 0 を含むタイルを除いた 64 コア
- ノードあたりのプロセス数（1–32 プロセス/ノード）

2017 年 2 月現在，OFP システムは KNL のクラスタリングモードとして Quadrant mode，メモリモードは Flat，Cache mode が実行可能であるため，その違いによる各アプリケーションの性能評価を行う．Flat mode で簡易に MCDRAM を用いる方法として `numactl` コマンドによる方法がある．`numactl` コマンドにより DDR4 メモリが NUMA ノード 0，MCDRAM が NUMA ノード 1 として見える．そこで，MCDRAM 上に全てのデータを配置する場合は，`numactl -membind 1` と指定する．本稿で用いるアプリケーションの問題サイズは全て MCDRAM におさまる 16GB 以下であるため，以上の方法により MCDRAM を使用する．KNL はハイパースレッディングによりコアあたり 4 スレッドまで実行可能であるため，スレッド数を変動させた場合の評価も合わせて行う．

OPA の通信スタックとして Tag Matching Interface（TMI），OpenFabrics Alliance（OFA），及び OpenFabrics Interfaces（OFI）が使用可能であるが，本稿では TMI（I_MPI_FABRICS=shm:tmi）の評価のみを示す．文献 [15] では，TMI を用いた場合は高性能だが 1 スレッド以上が通信のために専有されると述べられており，文献 [16] では，OFP はタイマ割り込みを受けつけるコアが 0 と設定してあると述べられている．また，KNL は同一タイルにおいて 2 コアが L2 キャッシュを共有しているため，L2 キャッシュ汚染を考えると 2 コアを演算に用いない設定が良いともされている．本稿ではコア 0 が性能に与える影響を調査するために，スレッドアフィニティを設定しない 64，68 コアを用いた評価に加えて，コア 0 を含んだタイルを除いた 64 コアによる評価も行う．

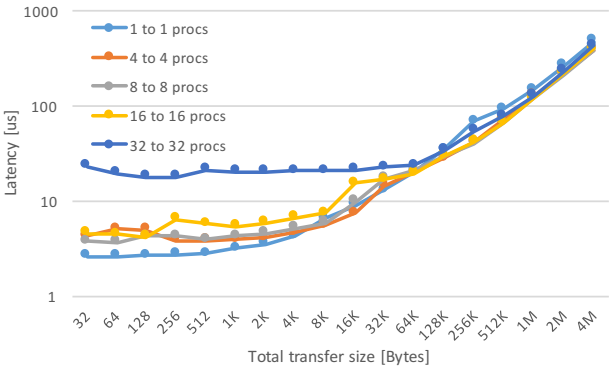

図 4　Send/Recv 通信のレイテンシ（Inter-node）[us].

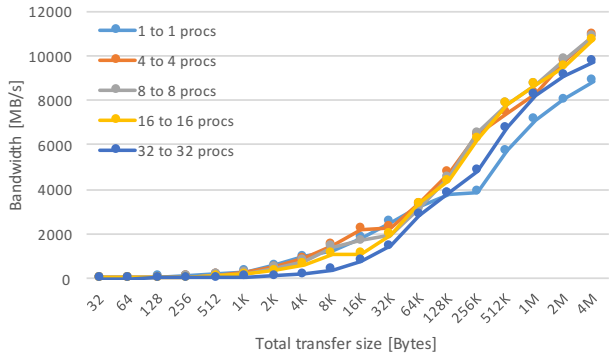

図 5　Send/Recv 通信のバンド幅（Inter-node）[MB/s].

KNL においてプロセスあたりのスレッド数の割り当てやコアへの配置は，KMP_HW_SUBSET を用いる．例えば，KMP_HW_SUBSET=”64c,2t” とした場合は，プロセスあたり 64 コアを使用し 1 コアあたり 2 スレッドを配置する．また，I_MPI_PIN_PROCESSOR_EXCLUDE_LIST によって，プロセスがバインドされないスレッドを指定することができる．従って，OFP システムの Intel Xeon Phi 7250（68 コア）において，上記で述べたコア 0 を含んだタイルを除いて実行したい場合は，I_MPI_PIN_PROCESSOR_EXCLUDE_LIST=”0,1,-68,69,136,137,204,205” と指定する．

5.1 予備評価

予備評価として OFP システムのインターコネクトとして用いられている OPA の通信性能の調査を行う．通信性能の評価にはオハイオ州立大学により開発が進められている OSU Micro-Benchmarks（version 5.3.2）[17] を用いる．対象アプリケーションの Himeno Benchmark は Send/Recv 通信，FFT は All-to-All 通信が主な通信であり，各通信によるレイテンシやバンド幅を示す．本稿では，1 ノードに

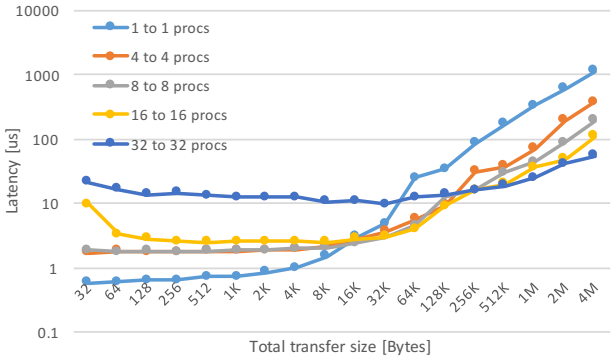

図 6 Send/Recv 通信のレイテンシ（Intra-node）[us].

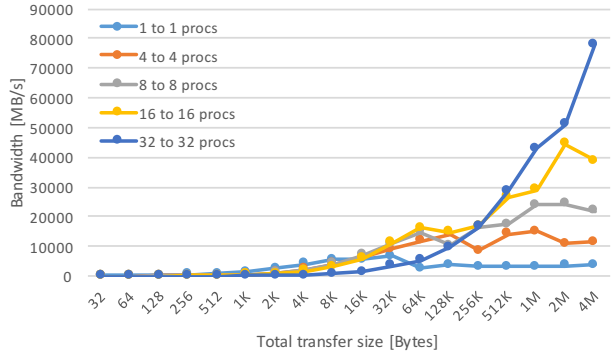

図 7 Send/Recv 通信のバンド幅（Intra-node）[MB/s].

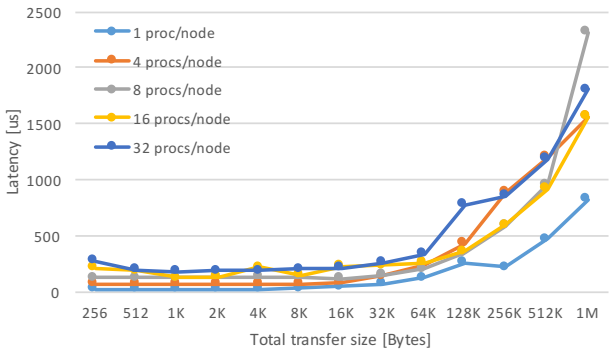

図 8 4 ノードを用いた All-toAll 通信のレイテンシ [us].

複数プロセスを配置した場合の性能評価も行うため，2 プロセスの Ping-pong ベンチマークによる評価に加えて，各ノードから複数の通信を同時に行う複数対複数プロセスでの評価も行う．

図 4 にノードを跨ぐ Send/Recv 通信のレイテンシ，**図 5** にバンド幅を示す．Ping-pong ベンチマークにより性能を評価するが 1 ノードあたりのプロセス数を 1，4，8，16，及び 32 と増加させる．通信はノードを跨いだプロセス同士でのみ行われ通信相手は固定である．横軸は 1 ノードが同時に通信するサイズの合計を表しており，例えば 1 ノード 4 プロセス（凡例では 4 to 4 procs）の場合は 4 対 4 プロセスでの通信を表し，横軸の 4KB は 4 プロセス合計で

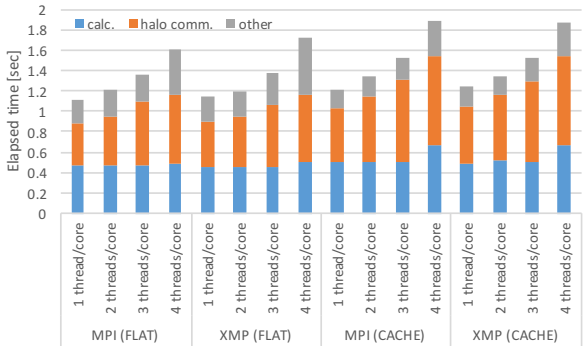

図 9 16 ノードによる Himeno Benchmark の実行時間の内訳 [sec]. Flat，Cache mode を対象にノードあたり 64 コアを使用し，コアあたりのスレッド数を 1–4 と変動した場合の評価.

4KB，つまり 1 プロセスが 1KB を送ることを示す．図 4，図 5 より，32KB を超えたあたりで 1 ノード 1 プロセスよりも 1 ノードに 4，8，及び 16 プロセスで通信した場合に通信性能が良いことがわかる．また，1 ノード 32 プロセスの場合でも 128KB 以降は 1 ノード 1 プロセスの場合よりも良い．

図 6 にノード内の Send/Recv 通信のレイテンシ，**図 7** にバンド幅を示す．ノードを跨いだ場合の性能評価と同様にプロセス数を増加させるが，全てのプロセスが 1 ノード上に配置される．従って，この評価は MPI の Send/Recv による MCDRAM 間のメモリコピーの性能調査となる．横軸は同時に通信が実行される合計サイズを表す．図 6，図 7 より，16KB 以上で 1 対 1 プロセスによる通信よりも，4，8，及び 16 プロセスと増加させ，同時に通信を実行した方が高速であることがわかる．また，32 プロセスの場合にも 64KB 以上で同様の傾向が見られる．以上より，KNL+OPA 環境において Send/Recv 通信の性能を引き出すためには，複数プロセスで同時に通信を発生させる必要があると考えられる．

図 8 に All-to-All 通信のレイテンシを示す．4 ノードを用い 1 ノードのあたりのプロセス数を 1，4，8，16，及び 32 プロセスと増加させ評価を行った．横軸は通信サイズを表し，1 ノード 1 プロセス時の通信サイズを基準にプロセスあたりの通信サイズを合わせている．Send/Recv 通信時はノードあたりのプロセス数を増加した場合に良い性能を得られていたが，All-to-All 通信の場合は 1 ノード 1 プロセスにおける通信性能が良くプロセス数を増加する毎に性能が低下していることがわかる．

通信性能の調査により，Send/Recv 通信を用いる Himeno Benchmark ではノードあたりのプロセス数を増加した場合に性能が向上すると考えられ，All-to-All 通信がある FFT では 1 ノード 1 プロセス時の性能が良くなると考えられる．

5.2 Himeno Benchmark

Himeno Benchmark は非圧縮流体解析コードの性能評価

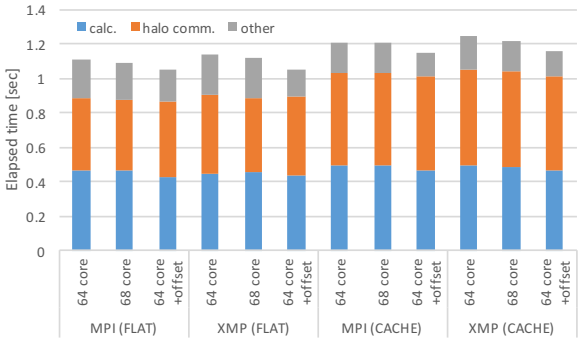

図 10 16 ノードによる Himeno Benchmark の実行時間の内訳 [sec]．Flat，Cache mode を対象にノードあたり 64，68 コア，及びコア 0 を含むタイルを除いた 64 コアによる評価．

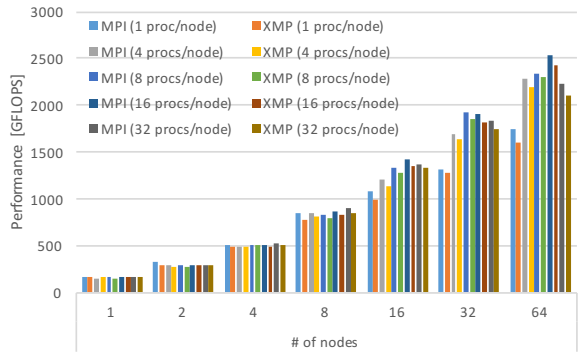

図 11 1 ノードあたりのプロセス数を変動した場合における Himeno Benchmark の性能評価 [GFLOPS]．ノードあたりのプロセス数を 1–32 と変動させ，1–64 ノードを用いて実行．

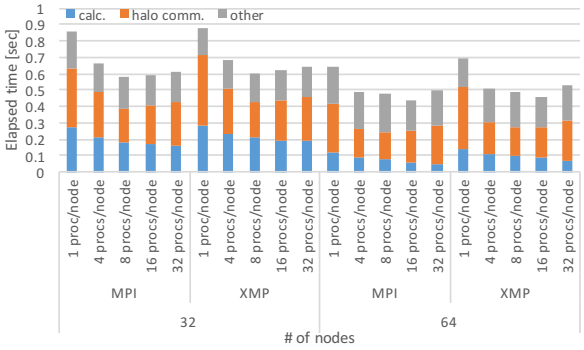

図 12 1 ノードあたりのプロセス数を変動した場合における Himeno Benchmark の実行時間の内訳 [sec]．最も性能差が大きいノードあたり 32，64 プロセスを 1–64 ノードを用いて実行．

のためのベンチマークである．主な演算はポアッソン方程式解法をヤコビの反復法で解く 3 次元の 19 点ステンシル演算でありメモリインテンシブなベンチマークである．主な通信は袖領域の交換による Send/Recv 通信である．本稿で用いる問題サイズは Large（$i \times j \times k$）=（256 × 256 × 512）とする．分割は i, j 次元の 2 次元分割とした．3 次元ステンシル演算を解く 3 重ループに対して，ループを展開しスレッド割付のためのオーバヘッドを減らすために，

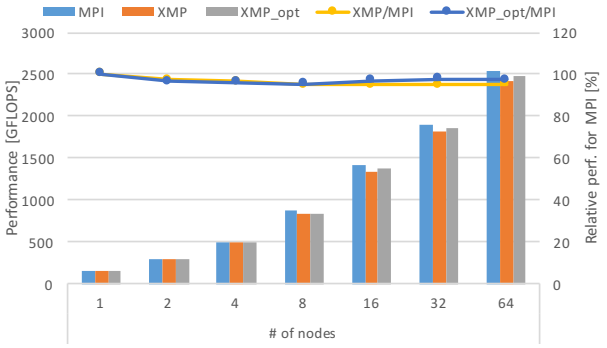

図 13 1 ノード 16 プロセスにおける Himeno Benchmark の性能評価 [sec]．1–64 ノードを用いて実行．MPI，XMP，及び配列アクセスの最適化を施した XMP の比較．

最外ループに `parallel for collapse(2)` と指定して i, j 次元に対してスレッド並列化し，最内ループに対してはベクトル化するよう `simd` 指示文を指定した．

図 9 に，16 ノードを用いた KNL のメモリモード（Flat, Cache mode）やコアあたりのスレッド数を変動した場合の実行時間の内訳を示す．メモリモードの違いにより Flat mode は MPI，XMP 実装ともに 8–10%程度 Cache mode より高速である．本稿の性能測定で用いた Himeno Benchmark の問題サイズ Large は MCDRAM の 16GB 内におさまるため，キャッシュ利用によるオーバヘッドにより性能が低下していると考えられる．また，ステンシル演算は一般的にメモリバンド幅律速であるため，コアあたりのスレッド数を増加させた場合に演算，通信ともに影響を受け性能が低下している．**図 10** に，メモリモードの違いに加え，64，68 コア，及びコア 0 を含んだタイルを除いた 64 コアによる実行時間の内訳を示す．この評価ではコアあたりのスレッド数を 1 とした．図 9 の評価同様に Cache mode より Flat mode が高速である．また，5 章で紹介した研究で述べられている通り，64，68 コアの実行と比較してコア 0 を除いた 64 コアによる実行において性能の向上が見られる．

以上の結果を踏まえ，メモリモードを Flat mode とし，コアあたりのスレッド数を 1，コア 0 を含んだタイルを除いた 64 コアという構成で，1 ノードあたりのプロセス数を変動させた場合の性能評価を行った．**図 11** にノードあたり 1，4，8，16，及び 32 プロセスと変動させ，最大で 64 ノードを用いて実行した場合の性能を示し，**図 12** に特に性能差が生じたノードあたり 32，64 プロセスを用いた実行時間の内訳を示す．図 11 より，1 ノード 16 プロセスで実行した場合の性能が良く，MPI 実装で 2536GFLOPS，XMP 実装で 2416GFLOPS の性能が得られた．図 12 を見ると，ノードあたりのプロセス数を増加した場合に演算時間が減少していることがわかる．これは，プロセス数増加によるプロセスあたりの演算量の減少により，メモリアクセスの局所性が増加しキャッシュヒット率を向上させてい

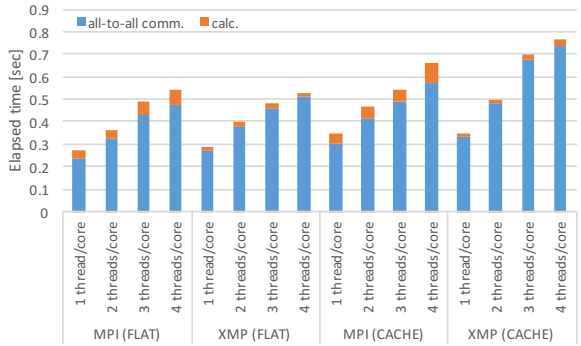

図 14 16 ノードによる FFT の実行時間の内訳 [sec]．Flat，Cache mode を対象にノードあたり 64 コアを使用し，コアあたりのスレッド数を 1–4 と変動した評価．

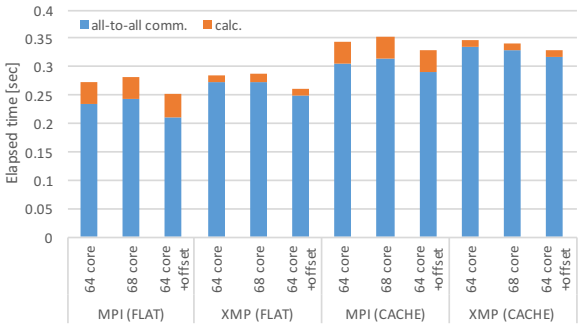

図 15 16 ノードによる FFT の実行時間の内訳 [sec]．Flat，Cache mode を対象に 64，68 コア，及びコア 0 を含むタイルを除いた 64 コアによる評価．

ると考えられる．今後，Intel VTune などの性能解析ツールを用い詳細な解析を行う予定である．

図 13 に，図 11 において最も性能が良い 1 ノード 16 プロセスの MPI，XMP，及び XMP 実装において最適化を施した “XMP_opt” の性能を示す．図 12 を見ると，XMP 実装は MPI 実装と比較して演算時間に差が生じており，94%の性能がピークである．MPI と XMP 実装の大きな違いとして，配列アクセス方法が異なる点が挙げられる．Omni XMP Compiler では，多次元分散配列を 1 次元ポインタに変換し，分散配列毎に各次元のサイズを変数とし保存する．従って，配列アクセスはそれらを用いたポインタ演算に置き換わる．そこで，本稿では Omni XMP Compiler に変更を加え，分散配列のローカルサイズを事前に計算し，1 次元ポインタではなく多次元配列の形状を保つようにした．図 13 の “XMP_opt” が，その実装の性能である．結果として，改良した XMP 実装は 64 ノード実行で 2469GFLOPS の性能が得られ，MPI 実装と比較して 97%まで性能が向上した．

5.3 Fast Fourier Transform（FFT）

FFT は 1 次元離散複素フーリエ変換に対する演算速度を評価するベンチマークであり，実行時間の大部分を全対

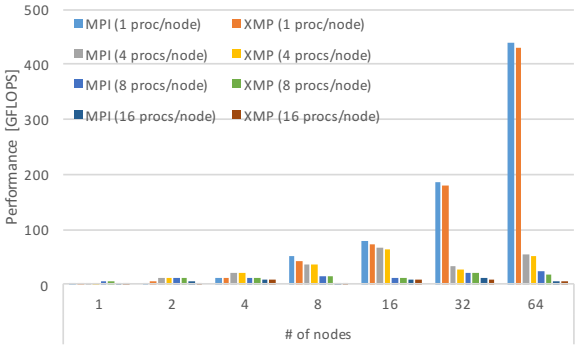

図 16 1 ノードあたりのプロセス数を変動した場合における FFT の性能評価 [GFLOPS]．1 ノードあたりのプロセス数を 1–16 と変動させ，1–64 ノードを用いて実行．

全通信が占める通信インテンシブなベンチマークである．XMP 実装は HPCC のリファレンス実装と同じく FFTE ライブラリ [18] による six-step FFT アルゴリズムで実装されている．本稿では問題サイズを 2^{27} とした．

まず，Himeno Benchmark による評価と同様に，16 ノードを用いて KNL のメモリモード（Flat，Cache mode）による性能への影響やコアあたりのスレッド数を変動した場合の性能を調査した．**図 14** に評価を示す．図 14 より，コアあたりのスレッド数が増加する毎に性能が低下しており，特に All-to-All 通信にかかる時間が増加している．また，メモリモードの違いによる性能差があり，Flat mode は MPI 実装では 20%，XMP 実装では 31%，Cache mode より高速である．本稿で用いた問題サイズは MCDRAM の 16GB 内におさまるため，Himeno Benchmark 同様にキャッシュ利用によるオーバヘッドにより性能が低下していると考えられる．**図 15** に 64，68 コア，及びコア 0 を含んだタイルを除いた 64 コアによる実行の性能を示す．コア 0 を含むタイルを除いた場合に最も良い性能が得られているのがわかる．コア 0 を含んだ 64 コアによる実行では MPI，XMP 実装ともに All-to-All 通信の性能が低下し，さらに 68 コア全てを用いた場合には若干の演算性能の向上は見られるが，それ以上に All-to-All 通信の性能が低下している．

以上の結果を踏まえ，メモリモードを Flat mode とし，コアあたりのスレッド数を 1，コア 0 を含むタイルを除いた 64 コアという構成で，1 ノードあたりのプロセス数を 1，4，8，及び 16 と変動させた評価を行った．**図 16** にその評価を示す．図 8 での考察の通り，All-to-All 通信のレイテンシが最も低かった 1 ノード 1 プロセスでの実行において最も良い性能が得られており，64 ノードにおいて MPI 実装で 438GFLOPS，XMP 実装で 431GFLOPS である．また，XMP による実装は MPI 実装の 98%の性能を得られており，MPI 実装とほぼ同等の性能であると言える．

6. XcalableMP2.0 の検討

6.1 XcalableMP2.0 の概要

現在，XMP ではメニーコアに向けた新たな仕様であるバージョン 2.0 の検討が進められている．提案の一つとして互いに依存関係を持つ “tasklet” によるタスク並列が検討されている [20]．ノード内/外におけるタスク間の依存関係を統一的に記述することで，コア全体による同期を排除したタスク間の細粒度な同期や，タスク化された通信と計算のオーバラップにより性能向上が期待される．また，その提案の中には比較的性能の低い個々のコアに対して，Omni-Path Architecture や InfiniBand EDR など高い通信バンド幅を持つインターコネクトの性能を十分に引き出すために，従来のプロセス単位ではなくスレッド単位での通信による実装も検討されている．本章では，XMP2.0 へ向けた予備性能評価として単一スレッドと複数スレッドが通信可能な Cholesky Factorization の MPI 実装の性能を示す．

6.2 Cholesky Factorization

Cholesky Factorization は，正定値対称行列を上三角行列と下三角行列に分解するコードであり，本稿ではブロック化した行列を対象とする．Barcelona Supercomputing Center（BSC）の OmpSs [19] のパッケージ内にあるコードを参考に MPI+OpenMP による実装を行った．演算は全て BLAS や LAPACK のルーチン（`potrf`，`trsm`，`syrk`，及び `gemm`）であり，本稿では Intel Math Kernel Library（MKL）を用いた．各演算を OpenMP の `task` 指示文と `depend` 節で記述し，タスク依存によるタスク並列実装とした．分割方法は 1 次元のブロックサイクリック分割とし，問題サイズは 32K × 32K，ブロックサイズを 512 × 512 とする．主な通信はブロック転送のための Send/Recv 通信である．さらに，単一スレッドのみが通信を行う実装と，通信をタスク内で実行させ全てのスレッドが通信を実行可能な実装をし，プロセス単位よりもさらに最粒度なスレッド単位での通信性能の評価を示す．

5.2，5.3 節の評価を踏まえ，メモリモードを Flat mode，コアあたりのスレッド数を 1，コア 0 を含んだタイルを除いた 64 コアという構成で，1 ノード 1 プロセスで最大 32 ノードを用いて評価を行った．スレッド単位での通信による性能評価を行うため，ノードあたり複数プロセスを配置した場合の評価は行っていない．性能評価の結果，少ないノード数の場合には単一スレッドによる通信と複数スレッド通信による性能差はない．しかし，ノード数が増えていく毎に性能差が増加していき，32 ノードの場合複数スレッド通信による性能は単一スレッドによる性能の 77%しか出ておらず，今後性能低下の原因の調査を行う予定である．

7. まとめ

本稿では，KNL メニーコアプロセッサにおいて PGAS 言語 XMP アプリケーションの性能評価や XMP2.0 に向けた初期評価を行った．加えて評価環境である OFP システムのインターコネクトである OPA の通信性能の評価や，実行に最適なコア数の調査などを行った．OPA の通信性能の評価では，Send/Recv 通信はメッセージ長が長い場合に，1 ノードあたりに複数プロセス配置した方が 1 ノード 1 プロセスよりも良い性能が得られ，All-to-All 通信では 1 ノード 1 プロセスが最も性能が良いことがわかった．さらに，通信やタイマ割り込みなどの影響を受けるコア 0 を除いた 64 コアでの実行の性能が良く，KNL のメモリモードは，MCDRAM に全てのデータが配置可能ならば Flat mode が高速であることがわかった．以上の評価により得られた構成で，1 ノードあたりのプロセス数を変化させてアプリケーションの評価を行ったところ，Himeno Benchmark は 1 ノードあたり 16 プロセス実行において，MPI 実装で 2536GFLOPS，XMP 実装で 2469GFLOPS の性能が得られた．また，FFT は 1 ノードあたり 1 プロセスの実行において，MPI 実装で 438GFLOPS，XMP 実装で 431GFLOPS の性能が得られた．64 ノードを用いた評価において，XMP 実装は Himeno Benchmark で MPI 実装の 97%の性能，FFT では 98%の性能が得られ，どちらも MPI 実装とほぼ同等の性能を得ることができた．

XMP2.0 の向けた予備性能評価では，単一スレッドと複数スレッドによる通信を用いた Cholesky Factorization を実装し性能評価を行った．複数スレッド通信による Cholesky Factorization の性能は，単一スレッドによる通信の場合と比較して 77%の性能に留まり，今後性能低下の原因を調査する予定である．

今後の課題として，メモリモードやパラメータ，通信方法による性能差の詳細な原因を調べるために Intel VTune などの性能解析ツールによる調査を行うことが挙げられる．また本稿では，OPA における Send/Recv，All-to-All 通信のみ性能を調査したが，他の collective 通信や片側通信についての性能調査や，MCDRAM の 16GB を超えるデータサイズでの評価などが挙げられる．

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，本研究で用いた Oakforest-PACS システムは最先端共同 HPC 基盤施設（JCAHPC）の「Oakforest-PACS 試験運用プログラム」による．

参考文献

[1] Top500 Supercomputer Sites, https://www.top500.org/
[2] JCAHPC（Joint Center for Advanced HPC：最先端共同

HPC 基盤施設）, http://jcahpc.jp/
[3] XcalableMP Language Specification Version 1.2.1, http://www.xcalablemp.org/, 2014
[4] J. Lee, M. Sato, Implementation and Performance Evaluation of XcalableMP: a Parallel Programming Language for Distributed Memory Systems, Proc. 39th Int. Conf. Parallel Process. Workshops (ICPPW), San Diego, (2010) pp. 413–420.
[5] M. Nakao, J. Lee, T. Boku, M. Sato, Productivity and Performance of Global-view Programming with XcalableMP PGAS Language, Proc. 12th IEEE/ACM Int. Symposium Cluster, Cloud Grid Comput (CCGrid), Ottawa, (2012) pp. 402–409.
[6] 理化学研究所情報基盤センター：姫野ベンチマーク, http://accc.riken.jp/supercom/himenobmt/
[7] HPC Challenge, http://www.hpcchallenge.org/
[8] J. Dongarra, P. Luszczek, HPC Challenge: Design, History, and Implementation Highlights, Contemporary High Performance Computing: From Petascale Toward Exascale, Jeffrey Vetter eds. Taylor and Francis, CRC Computational Science Series, Boca Raton, FL, (2013).
[9] 佐藤 三久, XcalableMP の現状と課題, 第 16 回 PC クラスタシンポジウム, 東京, 2016 年 12 月
[10] 村井 均, XcalableMP 2.0 の概要, 第 4 回 XcalableMP ワークショップ, 秋葉原, 東京, 2016 年 11 月
[11] A Sodani, 2nd Generation Intel Xeon Phi Processor, IEEE Hot Chips 27 Symposium, (2015) pp. 1–24
[12] Omni Compiler Project, http://omni-compiler.org/
[13] M. Nakao, H. Murai, T. Shimosaka, M. Sato, Productivity and Performance of the HPC Challenge Benchmarks with the XcalableMP PGAS language, 7th International Conference on PGAS Programming Models, Edinburgh, Scotland, UK, (2013).
[14] OpenMP, http://www.openmp.org/
[15] 廣川 祐太, 朴 泰祐, 佐藤 駿丞, 矢花 一浩, 電子動力学コード ARTED による Knights Landing プロセッサの性能評価, Vol.2016-HPC-157, No.8, pp.1–9, 沖縄, 2016 年 12 月
[16] 塙 敏博, 中島 研吾, 大島 聡史, 星野 哲也, 伊田 明弘, パイプライン型共役勾配法の性能評価, Vol.2016-HPC-157, No.6, pp.1–9, 沖縄, 2016 年 12 月
[17] OSU Micro-Benchmarks, http://mvapich.cse.ohio-state.edu/benchmarks/
[18] FFTE: A Fast Fourier Transform Package, http://www.ffte.jp/
[19] 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.
[20] 津金 佳祐, 中尾 昌広, 李 珍泌, 村井 均, 佐藤 三久, ”軽量スレッドライブラリ Argobots による PGAS 言語 XcalableMP の動的タスク並列機能の設計”, Vol.2016-HPC-155, No.29, pp.1–8, 長野, 2016 年 8 月