

Xeon Phi+OmniPath環境におけるOpenMP, MPI性能最適化

埴 敏博^{1,a)} 星野 哲也¹ 中島 研吾¹ 大島 聡史¹ 伊田 明弘¹

概要: メニーコアプロセッサからなるクラスタシステムでは、OpenMP のスレッド数と MPI プロセス数について様々な組み合わせが考えられる。Intel Xeon Phi および OmniPath で構成される Oakforest-PACS および ofp-mini (最先端共同 HPC 基盤施設) を用いてこれらの基本性能を評価し、システムの性能最適化およびオーバーヘッド削減について示す。さらに、これらの知見をアプリケーションに適応した結果得られた性能改善について報告する。

1. はじめに

高い演算性能を限られた電力制約の中で実現するため、様々なプロセッサの開発が行われている。その中で、汎用性を維持しつつ、並列性を高めることにより演算性能向上を実現するメニーコアプロセッサが注目を集めている。

Intel Xeon Phi は、当初 MIC (Many Integrated Core) アーキテクチャとして開発が始められたが、最初に市販された Knights Corner (KNC) はコプロセッサであり、通常の Xeon サーバに PCI Express (PCIe) を介して接続する形で実現されていた。従って、KNC では自ら通信する手段は持たず、例えば、ホストの Xeon に接続された InfiniBand (IB) などに対して、PCIe 経由で IB のエミュレーションデバイスや SCIF といった専用インタフェースを介して通信を行う必要があった。また、CPU が in order の Pentium 相当のコアで、コア間がリングバス接続であったり、搭載メモリも GDDR5 で容量も 6~16GB と限られていた。

第2世代の Knights Landing (KNL) では、セルフブート可能なプロセッサになり、通常の Linux がそのまま動作する。Xeon とバイナリ互換になるなど、利便性も高い。また、高バンド幅メモリを搭載し、通常の DDR4 メモリの数倍のバンド幅を持つ。

一方、HPC システム向けのインタコネクタとして、これまで InfiniBand が用いられることが多かったが、Intel は Omni-Path アーキテクチャ (OPA) というインタコネクタ製品を発表している。InfiniBand EDR と同じく 100 Gbps のリンク速度を持ち、スイッチ、PCIe のカード (Host Fabric Interface: HFI) だけではなく、KNL のチップと統

合された製品も市販されている。

著者らの所属している東京大学情報基盤センターは、筑波大学計算科学研究センターと協力して、最先端共同 HPC 基盤施設 (JCAHPC: Joint Center for Advanced High Performance Computing) を立ち上げ、メニーコア型大規模スーパーコンピュータシステムの調達を行ってきた [1]。その結果、8,208 ノードの KNL と OPA を搭載し、ピーク性能 25PFLOPS の「Oakforest-PACS システム」が導入され、2016 年 12 月より稼働を開始した。これに先立ち、8 ノードの実験用システム ofp-mini も用いて様々なテストを行ってきた。

このようなメニーコア型のクラスタシステムで並列処理を行う際に、OpenMP と MPI をどのように組み合わせるのが最適か、など、最適化の余地は多岐にわたる。

本稿では、Intel Xeon Phi KNL と Omni-Path アーキテクチャから構成されるクラスタにおいて、OpenMP や MPI に関するベンチマークを用いて、基本性能を明らかにするとともに、これらの知見をアプリケーションに適用した結果、得られた性能改善について報告する。

2. Knights Landing プロセッサと Omni-Path アーキテクチャ

本章では、Knights Landing 世代の Intel Xeon Phi プロセッサ (本章では Knights Landing プロセッサと呼ぶことにする) と、Omni-Path アーキテクチャの概略について述べる。詳しくは文献 [2], [3] などを参考にされたい。

¹ 東京大学 情報基盤センター

^{a)} hanawa@cc.u-tokyo.ac.jp

2.1 Knights Landing プロセッサ

2.1.1 内部構成

Knights Landing プロセッサは、Silvermont 世代の Atom プロセッサをベースにした、2 パイプラインからなる out-of-order コアを、最大 72 コアまで搭載したメニーコアプロセッサである。各コアは 4 つまでの HyperThreading に対応している。

各コアには、2 つの AVX-512 ベクタ命令を実行するベクタ演算ユニットが付いている。図 2 に示すように、2 コア分とそれらで共有される 1 MB の L2 キャッシュで 1 つのタイルを構成している。

図 1 に KNL のブロック図を示す。合計 38 タイルが 2 次元メッシュで接続されており、そのうちいくつかのタイルは無効化されている。メッシュ上のルーティングは YX ルーティングを採用し、静的に経路が決定する。つまり、まず垂直 (Y) 方向に転送されたのちに水平 (X) 方向に転送され目的のタイルに到着する。1 ホップ転送するのに、Y 方向には 1 クロック、X 方向には 2 クロックかかる。

L2 キャッシュでは分散タグディレクトリを採用しており、その情報は、各タイルにある CHA (Caching Home Agent) に保持されている。CHA が実質的にチップ内部のインターコネクトとタイルとを接続する役割をしている。

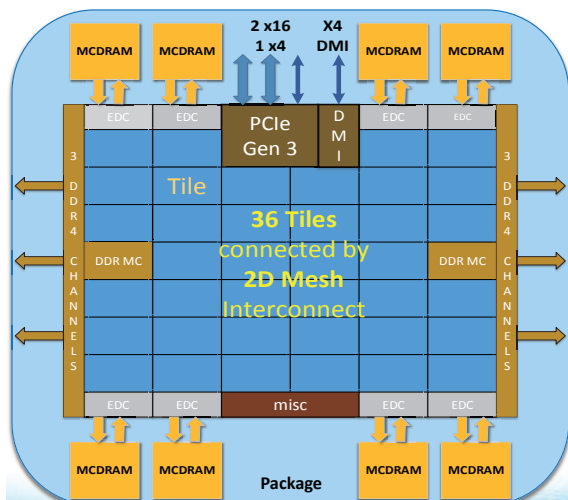


図 1 KNL の構成 ([2] 発表スライドから引用)

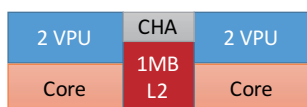


図 2 KNL におけるタイルの構成

2.1.2 メモリ

KNL では、通常の Xeon などでは用いられている DDR メモリに加えて、高いバンド幅を持つ MCDRAM (MultiChannel-DRAM) が 16 GB 搭載されている。

図 1 に示す通り、MCDRAM は 8 つの専用メモリコントローラに接続され、KNL チップと同一パッケージ上に搭載される。

MCDRAM を利用するためのモードとして、以下のメモリモード、クラスタリングモードを備えており、BIOS によって切り替えることができる。

● メモリモード

Flat: MCDRAM に対して DDR4 とは異なるメモリアドレスが与えられ、明示的にアクセスが可能になる。memkind ライブラリにより、プログラム中で C 言語では `hbw_malloc` 関数、Fortran では FastMem 指示子によって動的に確保するか、プログラム実行時に `numactl` などによって MCDRAM に対して bind することによって利用可能になる。

Cache: MCDRAM を DDR4 メモリのキャッシュのように扱う。プログラム中から明示的に MCDRAM をアクセスすることはできない。KNL では Direct Map による L3 キャッシュとして振る舞う。

Hybrid: MCDRAM の容量を分割して、Flat モードと Cache モードの両方が利用可能になる。

● クラスタリングモード

KNL ではチップ上のリソースを各象限に 4 分割して考えるのが自然な設計になっている。ここでは、それを踏まえて Quadrant モードと SNC-4 モードについて説明する。

Quadrant: 全体を 1 ソケットの CPU のように扱うが、内部的なメモリアクセスのトラフィックとしては各象限に閉じたローカルなアクセスになるようなアドレッシングになる。

SNC-4: 各象限が独立した NUMA ドメインとして扱われ、プログラムからは、あたかも 4 ソケットシステムのように見える。これをサブ NUMA クラスタリングと呼ぶ。適切なアフィニティ制御を行うことで Quadrant よりも高い性能が期待できる。

2.2 Omni-Path アーキテクチャ

Omni-Path アーキテクチャ (OPA) は、Intel TrueScale (旧 QLogic 社) の InfiniBand、および Cray 社の Aries インタコネクトをベースに、HPC 向けに新たに開発されたインタコネクトである。低レイテンシ、高バンド幅、高密度なシステム実装を実現している。

性能に関わるパラメータとしては、Max Transfer Unit (MTU) が InfiniBand EDR では 4KB なのに対して、OPA では 8KB と 2 倍であり、高いバンド幅性能が期待できる。

また、InfiniBandなどのスイッチはこれまで36ポートが基本単位となっていたが、OPAでは48ポートが基本単位となっており、スイッチ数削減に寄与している。

KNLには、Omni-Pathのインタフェースをパッケージに統合したバージョンも存在するが、今回使用するOakforest-PACSおよびofp-miniでは、いずれも独立したPCIeのHFIを用いてOPAに接続している。

OPAでは、Performance-Scaled Messaging 2(PSM2) APIの使用が標準になっているが、InfiniBandとの互換性のためにVerbsの実装も使用することができる。また、OpenFabrics Alliance (OFA)によって標準化されているOpenFabrics Interfaces (OFI)も、PSM2用のライブラリが提供されており、OFIに対応していればOPAの性能を十分に活かすことができる。

2.3 Oakforest-PACSシステムと ofp-mini システム

Oakforest-PACSシステム(OFP)は、最先端共同HPC基盤施設(JCAHPC)によって共同運営されている、8,208ノードからなるKNLシステムである。各計算ノードは、Intel Xeon Phi 7250 1ソケットで構成され、68コア1.40GHzで動作する。各計算ノードにPCIe接続のOPA HFIが装着され、エッジスイッチにメタル線で接続された後、Directorスイッチに光ファイバで接続されている。

一方、ofp-miniシステムも同じくJCAHPCによって使われている8ノードの実験用システムである。各計算ノードは、Intel Xeon Phi 7210 1ソケットであり、64コア1.30GHzで動作する。各計算ノードにはOFP同様、PCIe接続のOPA HFIが装着され、エッジスイッチにメタル線で接続されている。

表1にそれぞれの環境をまとめる。

今回の評価においては、上で述べたメモリモードは、全てFlatモードとQuadrantモードの組み合わせで実施している。

表1 実験環境

略称	OFP	ofp-mini
CPU	Xeon Phi 7250	Xeon Phi 7210
周波数	1.40	1.30
コア数 (最大有効スレッド数)	68 (272)	64 (256)
理論演算性能 (GFLOPS)	3,046.4	2,662.4
メモリ容量 (GB)	MCDRAM: 16 DDR4: 96	
メモリバンド幅 (GB/sec)	MCDRAM: 490 DDR4: 84.5	MCDRAM: 454 DDR4: 72.5
OS	CentOS 7.2	
カーネル	xppsl 1.4.1	xppsl 1.5.0
OPAドライバ	10.2	10.3

3. ベンチマークによる性能評価

本章では、OpenMP マイクロベンチマーク、OpenMPによるpingpongベンチマーク、MPIバンド幅性能とコア番号の依存性、最後にOpenMP/MPI マイクロベンチマークの結果について述べる。

3.1 OpenMP マイクロベンチマーク

エジンバラ大学 EPCC によって配布されている、OpenMP マイクロベンチマーク [4], [5] を用いて、KNLにおけるOpenMP各指示文のオーバヘッドを測定した。具体的には、syncbenchを用い、主に同期のオーバヘッドを測定した。

結果を図3、図4に示す。指示文によりオーバヘッドの時間に大きな差が見られたため、グラフを2つに分けた。なお、2コアでは同一タイル内、64コアまでは異なる物理コアへの割り当てを行い、128コア、256コアでは、それぞれHyperThreadingコアを2個ずつ、4個ずつと指定した。

図3から、parallelとparallel forとsingle, forとbarrierが似た傾向を示していることがわかる。これらの指示文は暗黙のうちに終了時にbarrierを実行するため、barrierのオーバヘッドを内包していると考えられる。またこれらのオーバヘッドは16コアまではそれほど大きな差がないが、32ノード以上になると顕著にオーバヘッドが増える。128, 256コアはHyperThreadingを使っているため、オーバヘッドがさらに増加しているが、特に256コアにおけるオーバヘッドは128コアの1.4倍程度に増加している。

このことから、比較的頻繁にOpenMP指示文を呼び出す場合には、32スレッドを超えるとスレッド間の同期等のコストで、性能が劣化するであろうと予測される。特に、HyperThreadingを用いた場合にはその影響が大きく、演算器資源を有効に使い切れるか、OpenMPリージョンの大きさととのトレードオフになると考えられる。

また、比較のために、Reedbush-U (RBU, Xeon E5-2695v4 2.1GHz 18コア, Broadwell-EP x 2ソケット)の結果を図5に示す。RBUでは、ソケットあたり16コアを使い、32コアの場合のみ2ソケットにまたがったスレッド配置になっている。

32コアの場合でOFPとRBUとを比較すると、parallelに類する指示文はおおよそ1.7倍程度OFPの方が遅い。コアのクロック周波数を考えると、RBUが2.1GHzに対しOFPが1.4GHzであり、周波数比で1.5倍であるが、Xeonコアが非常に高機能であり、Turboboostの効果も期待できることから、1.7倍の性能比は妥当であると考えられる。

一方図4によると、OFPの16コア以上ではAtomicに比べてCriticalの方が高速である。Xeonにおいては、Atomicは圧倒的に高速であり、32コアにおいても0.1μ秒以下であり、OFPとは8倍以上の性能差がある。これは、KNL

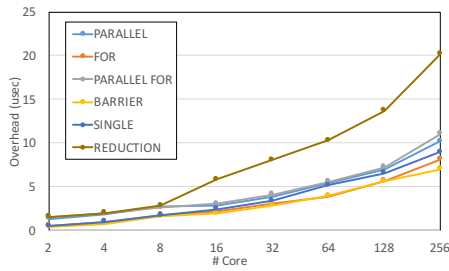


図 3 OFP における OpenMP 指示文のオーバーヘッド (parallel, for, parallel for, barrier, single, reduction)

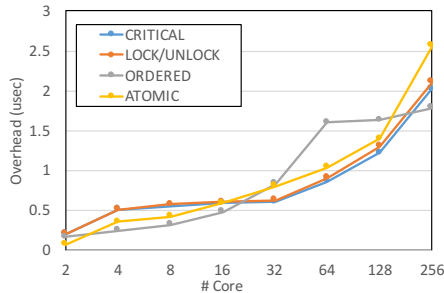


図 4 OFP における OpenMP 指示文のオーバーヘッド (critical, lock/unlock, ordered, atomic)

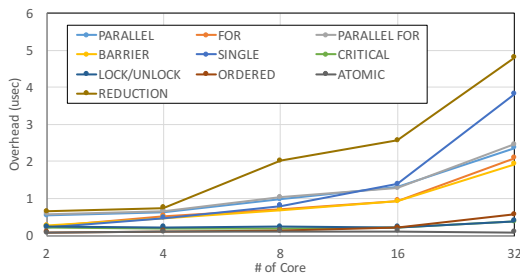


図 5 Reedbush-U における OpenMP 指示文のオーバーヘッド

のコアとメモリ間のアクセスレイテンシが Xeon に比べて非常に大きいため、Atomic が影響を受けやすかったと考えられる。実際にタイル内で L2 キャッシュを共有する 2 コアの場合には、Xeon の Atomic でのオーバーヘッドとほとんど変わらないため、チップ内ネットワークを通過するレイテンシが大きいと考えられる。

3.2 OpenMP による pingpong

KNL 1 ノードにおいて、2 コア間の距離を測定する目的で、OpenMP コードにより pingpong を行うプログラムを作成した。様々なバージョンを作成したが、最も安定して高速だったコードの例を図 6 に示す。実際には、PAUSE は空であり、100 回連続の pingpong を行い、1 回あたりの round trip time(RTT) の平均値を求めるプログラムである。

OFP において、タイル 0 31 のうち 2 つを総当たりで測定し、それぞれ 5 回実施したうちで最も小さい値を求めた結果、図 7 に示す結果が得られた。縦横それぞれは、タイ

```

1  #pragma omp parallel private(old) shared(go,ack)
2  {
3      if(myid==0) { /* master */
4          #pragma omp atomic write
5              go = 1;
6              do { /* wait for ack */
7                  PAUSE;
8          #pragma omp atomic capture
9              { old = ack; ack = 0; }
10             }while(old == 0);
11
12         }else{
13             /* slave */
14             do { /* wait for start */
15                 PAUSE;
16             #pragma omp atomic capture
17                 { old = go; go = 0; }
18             }while(old == 0);
19             delay();
20             #pragma omp atomic write
21                 ack = 1;
22             }
23         } /* end parallel */

```

図 6 OpenMP によるコア間 pingpong の例

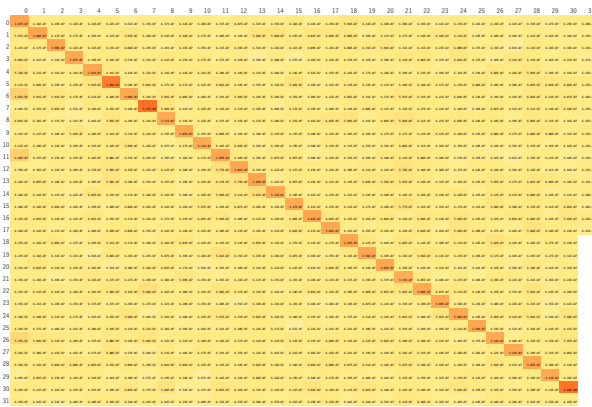


図 7 コアの組み合わせによる pingpong 往復時間

ルの番号を示しており、同一のタイル番号の場合、実際には異なるコアでそれぞれ実行している。赤が濃いものが値が小さいもの、色が薄いものは値が大きいものを視覚的に示している*1。

その結果、同一タイルのコア同士の場合には、 $RTT = 0.05 \sim 0.3 \mu$ 秒と他の組み合わせに比べて顕著に高速であり、識別することができた。しかしながら、他の組み合わせの場合には、いずれも 0.6μ 秒前後であり、コアの配置を特定できるような有意な差は得られなかった。逆に、OpenMP のレベルでは、コアのアフィニティは、(少なくとも Flat モード, Quadrant モードにおいては) 同一タイル同士か否か、だけを気にすれば良いことになる。

3.3 MPI バンド幅性能のコア依存性

続いて、OPA を用いた MPI 通信において、KNL のコア ID 毎の通信バンド幅性能を調べた。

ofp-mini の 2 ノードを用いて、オハイオ州立大による OSU MPI ベンチマーク [6] に含まれる osu_bw により、バンド幅を測定した。

図 8 に結果を示す。多くのコアでは $8.3 \sim 8.5$ GB/sec 程度の性能が得られているが、コア 9 では性能が高く 10.7 GB/sec の性能が得られている。その反対に、コア 8 では性能が低く 6 GB/sec 弱しか出ていない。

文献 [7] によると、OPA デバイスドライバ (hfi1 ドライバ) の送信用 DMA エンジンの割り込みハンドリング用に、登録するコアの割り当てを工夫すると改善されるとのことである。そこで、ofp-mini の hfi1 ドライバに登録する送信用 DMA エンジン 16 チャンネルに対し、順番に、コア番号 5, 4, 7, 6, 9, 8, ..., 19, 18 と割り当てたところ、図 9 のような結果を得た。その結果、コア 4~19 の 16 個のコアに対しては、 10.6 GB/sec 程度の性能を得ることができ、最適化前には存在していた、性能の低いコアが見られなくなった。

今後は、アプリケーションにおいて、これらの高いバン

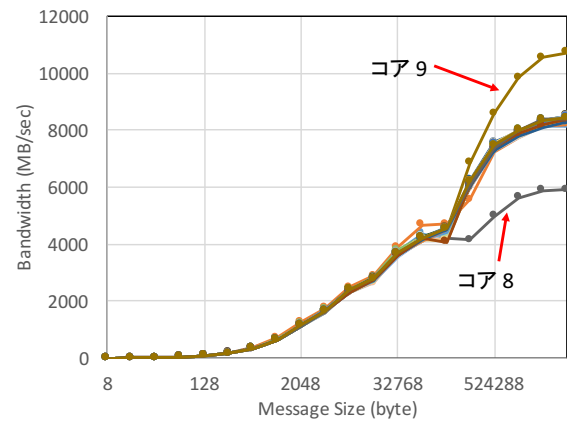


図 8 コア毎の MPI 通信性能の違い

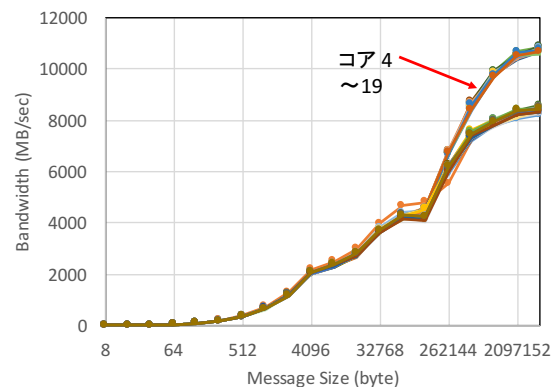


図 9 最適化後のコア毎の MPI 通信性能

ド幅が提供できるコアに MPI 処理を任せるなどのチューニングやコアへのマッピングについて検討する予定である。

3.4 OpenMP/MPI マイクロベンチマーク

最後に、OpenMP/MPI ハイブリッドプログラミング環境において、MPI 通信方法による通信性能の違いについて測定した。OpenMP マイクロベンチマークと同様、エジンバラ大学 EPCC によって配布されている、OpenMP/MPI マイクロベンチマークを用いた [8], [9]。

図 10 に示すように、マルチスレッド非対応の MPI 実装の場合には、(a) のように parallel 文を閉じてから通信を開始する必要がある。マルチスレッド対応の MPI では、(b) のように、マスタースレッドだけが通信処理を行う Funnelled モードか、(c) のように全スレッドが通信処理できる Multiple モードがある (他にも (c) のように見えるが実際は逐次処理を行う Serialized もあるが、このベンチマークではサポートしていない)。

OFP において 64 スレッドを使い、2 ノードの間で同じスレッド番号同士で pingpong をした結果を図 11 に示す。この結果より、MasterOnly と Funnelled の差はほとんどなく、全体としてわずかに Funnelled の性能が高い程度である。Multiple については性能が不安定であることがわ

*1 図中タイル 31 と 18~31 の組み合わせは、ジョブがタイムアウトして結果が取れなかったため白の表示にしている。


```

1  #pragma omp parallel
2  {
3      ....
4  }
5  MPI_Send( ... );
6  #pragma omp parallel
7  {
8      ....
9  }

```

```

1  #pragma omp parallel
2  {
3      ....
4      #pragma omp master
5          MPI_Send( ... );
6      ....
7  }

```

```

1  #pragma omp parallel
2  {
3      ....
4      MPI_Send( ... );
5      ....
6  }

```

図 10 OpenMP/MPI における通信の方法: (a) MasterOnly(上)
(b) Funnelled(中) (c) Multiple(下)

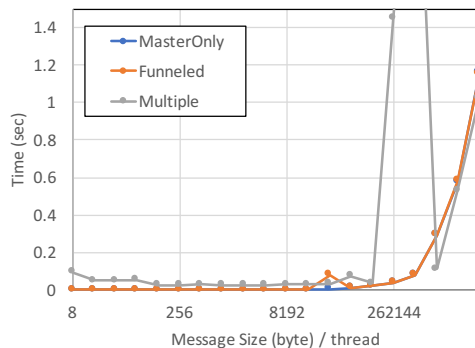


図 11 OpenMPI/MPI において MPI 通信方法を変えた時の ping-pong 性能

かった。

4. 実アプリケーションコードによる性能評価

OpenMP マイクロベンチマークの結果から、KNL では OpenMP 各指示文の同期などのオーバーヘッドが比較的大きいことがわかった。本章では、OpenMP のオーバーヘッドが実アプリケーションに与える影響について評価を行なう。実アプリケーションとしては、三次元ポアソン方程式を解く、不完全コレスキー分解前処理付き共役勾配法 (ICCG 法) ソルバー、およびパイプライン型共役勾配法を用いる。

4.1 ICCG 法ソルバーへの適用

我々は様々な環境において ICCG 法ソルバーの評価をしてきており、[10] においては KNL による性能評価を行っている。多色順序付け手法・疎行列格納手法は、[10] 同様 CM-RCM(k) 法・SELL- C - σ 法を用い、解析対象領域の要素数も同じく 128^3 とした。

4.1.1 KNL における OpenMP の同期削減

OpenMP 指示文の指定方法が、ICCG 法ソルバーの性能にどのような影響を与えるか評価を行う。ICCG 法ソルバーの OpenMP 実装概要を図 12 に示す。基本的には、並列化対象ループを `!$omp parallel do` により並列化しているが、前進後退代入、SpMV 部では、色ループの外側で `!$omp parallel` を指定し、並列化対象ループには `!$omp do` を指定するという方式をとっている。この実装をベースラインとし、以下の変更を順に適用した。

- (1) (Baseline) 図 12 の実装。
- (2) (nowait) SpMV 部に対する `!$omp end do nowait` の適用による同期の削減。
- (3) (mv-parallel) `!$omp parallel` の収束判定ループ外への移動。
- (4) (rm-ompdo) `!$omp do` に頼らない手動によるループ分割。(リダクション部以外、図 13)
- (5) (rm-reduction) `!$omp do reduction` に頼らない手動によるリダクション実装による同期の削減。(図 14)

この変更による OpenMP 指示文の出現数の変化を表 2 まとめる。parallel do と parallel と doそれぞれの終了時に 1 回、reduction 中 (コア毎のローカルリダクションとグローバルリダクションの間) に 1 回、暗黙の同期が入るものとしてカウントしている。また、この変更が性能に与えた影響を図 15 に示す。

表 2 1 イテレーションあたりの OpenMP 指示文出現数 (色数 12 のとき)。カッコ内の数字は上記最適化番号。

	(1)	(2)	(3)	(4)	(5)
parallel do	5	5	0	0	0
parallel	3	3	0	0	0
do	35	23	28	3	0
do (nowait)	0	12	12	0	0
reduction clause	3	3	3	3	0
barrier (explicit)	0	0	1	26	29
barrier (implicit)	43	31	28	3	0

同期コストを軽減する nowait、スレッドの生成・破棄コストを軽減する mv-parallel の適用は大きな効果があった。rm-ompdo の適用は同期コストを減らさないため優位な効果が見られなかったが、rm-reduction は同期コストを軽減するために効果が確認できた。全てを適用することで、Baseline から 8.8% の性能向上を達成した。

```

1  do itr = ... ! 収束判定ループ
2  !$omp parallel do private(...)
3      do i= 1, N
4          ...
5      end do
6  !$omp end parallel do
7
8  !$omp parallel private(...)
9      do ic= 1, NCOLORTot ! 色ループ
10     !$omp do
11         do ip= 1, PEsmptTOT
12             ! 前進代入
13         end do
14     !$omp end do
15     end do
16 !$omp end parallel
17 ...
18 RHO= 0.0d0
19 !$omp parallel do reduction(+:RHO)
20     do i= 1, N
21         RHO= RHO + ...
22     end do
23 !$omp end parallel do
24 ...
25 !$omp parallel private(...)
26     do ic= 1, NCOLORTot ! 色ループ
27     !$omp do
28         do ip= 1, PEsmptTOT
29             ! SpMV
30         end do
31     !$omp end do
32     end do
33 !$omp end parallel
34 ...
35 end do ! 収束判定ループ

```

図 12 ICCG 法ソルバーの OpenMP 実装概要. NCOLORTot: 総色数, PEsmptTOT: 総スレッド数

```

1  ip = omp_get_thread_num()+1
2  nth= omp_get_num_threads()
3  ls = (N+nth-1)/nth
4  do i= (ip-1)*ls+1, min(ip*ls,N)
5      ...
6  enddo
7  !$omp barrier

```

図 13 図 12 の 2-6 行目への手動によるループ分割の適用.

4.2 パイプライン型 CG 法への適用予備評価

ICCG 法ソルバーと同様に, パイプライン型 CG 法についても [11] において KNL による性能評価を行った. [11] では, 4 種類のアルゴリズムを比較したが, ここでは Chronopoulos/Gear アルゴリズムについて, OpenMP のバリア同期等の修正を行った.

特に, 時間測定を行っている部分は全て, 同一の parallel リージョンとすることで, parallel - end parallel の頻発によるオーバーヘッド削減を期待している. 通信に関しても, Funnelled モードを用いることで, parallel リージョンを終

```

1  W_RHO(ip)= 0.0d0
2  do i= (ip-1)*ls+1, min(ip*ls,N)
3      W_RHO(ip)= W_RHO(ip) + ...
4  enddo
5  RHO= 0.0d0
6  !$omp barrier
7  do i= 1, nth
8      RHO= RHO + W_RHO(i)
9  end do

```

図 14 図 12 の 18-23 行目への手動によるリダクションの適用. 変数 RHO は private 変数としており, 全てのスレッドが RHO を独自に計算する.

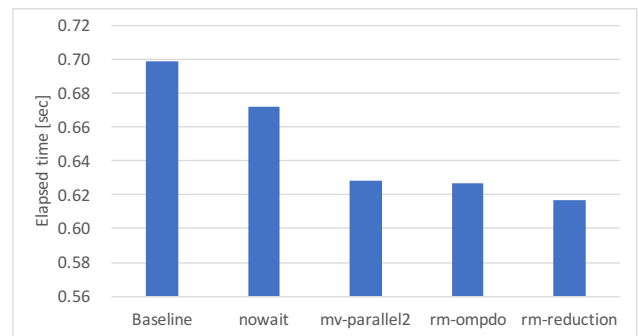


図 15 OpenMP の同期等コスト削減の効果

表 3 Chronopoulos/Gear アルゴリズムにおける OpenMP/MPI 最適化の効果

	最適化前	最適化後
実行時間 (sec)	2.981339	2.815626

わらせることなく通信が実現できる.

OFP 128 ノードを用いて, ノードあたり 4MPI プロセス, 16 スレッド, $256 \times 128 \times 144$ 節点, ノード内オーダリングは CM-RCM(10) を用いた. 最適化前と最適化後の実行時間 (試行の中で最速のもの) を表 3 に示す. 結果から, 最適化により 5.8% の性能向上が得られたことがわかる. 今後は, 他のアルゴリズムへの最適化の適用や, 性能向上の分析等を行う必要がある.

5. おわりに

本稿では, Intel Xeon Phi (Knights Landing) プロセッサと Omni-Path アーキテクチャから構成されるクラスタである, 最先端共同 HPC 基盤施設 Oakforest-PACS および ofp-mini システムを用いて, OpenMP や, MPI, これらのハイブリッド実行に関するマイクロベンチマークを用いて基本性能を明らかにした. 一般的な Xeon プロセッサと比べると, コア性能が低いことや, メモリアクセス遅延が大きいことから, 異なる特性を示すことがわかった.

さらにこれらの知見を元にアプリケーションに適用し, 最適化を行った. その結果, ICCG 法ソルバーでは 8.8%, パイプライン型 CG 法では 5.8% の性能向上が得られた.

今後は, OpenMP/MPI の組み合わせで, 詳細な調査を

行い、アプリケーションのさらなる最適化を実施していく予定である。

謝辞 Oakforest-PACS の利用・運用にあたっては、富士通株式会社および JCAHPC の皆様に感謝致します。Xeon Phi 向けのチューニングについて数多くの助言をいただいたインテル株式会社 堀越 将司博士に感謝いたします。

参考文献

- [1] 最先端共同 HPC 基盤施設：最先端共同 HPC 基盤施設 (JCAHPC) について, 最先端共同 HPC 基盤施設 (オンライン), 入手先 (<http://jcahpc.jp>) (参照 2017-02-07).
- [2] Sodani, A.: Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor, *HotChips: A Symposium on High Performance Chips*, (online), available from (http://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.25-Tuesday-Epub/HC27.25.70-Processors-Epub/HC27.25.710-Knights-Landing-Sodani-Intel.pdf) (2015).
- [3] Jeffers, J., Reinders, J. and Sodani, A.: Intel Xeon Phi Processor High Performance Programming, Knights Landing Edition, *Elsevier* (2016).
- [4] EPCC: EPCC OpenMP micro-benchmark suite, The University of Edinburgh (online), available from (<https://www.epcc.ed.ac.uk/research/computing/performance-characterisation-and-benchmarking/epcc-openmp-micro-benchmark-suite>) (accessed 2017-02-07).
- [5] Bull, J. M., D.O' Neill: A microbenchmark suite for OpenMP 2.0, *SIGARCH Comput. Archit. News*, Vol. 29, No. 5, pp. pp. 41–48 (2001).
- [6] Team, M.: OSU Micro-Benchmarks 5.3.2, The Ohio State University (online), available from (<http://mvapich.cse.ohio-state.edu/benchmarks/>) (accessed 2017-02-07).
- [7] Intel: Omni-Path Fabric Performance Tuning User Guide, Intel Corp. (online), available from (http://www.intel.com/content/dam/support/us/en/documents/network-and-i-o/fabric-products/Intel_OP_Performance_Tuning_UG_H93143_v6.0.pdf) (accessed 2017-02-07).
- [8] EPCC: EPCC OpenMP/MPI micro-benchmark suite, The University of Edinburgh (online), available from (<https://www.epcc.ed.ac.uk/research/computing/performance-characterisation-and-benchmarking/epcc-openmpmpi-micro-benchmark>) (accessed 2017-02-07).
- [9] J. M. Bull, J. Enright, X. G. C. M. and Reid, F.: Performance Evaluation of Mixed-Mode OpenMP/MPI Implementations, *International Journal of Parallel Programming*, Vol. 38, No. 5–6, pp. pp. 41–48 (2010).
- [10] 中島研吾, 大島聡史, 塙敏博, 星野哲也, 伊田明弘: ICCG 法ソルバーの Intel Xeon Phi 向け最適化, 研究報告ハイパフォーマンコンピュティング (HPC), Vol. 2016-HPC-157, No. 16, pp. 1–8 (2016).
- [11] 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, 研究報告ハイパフォーマンコンピュティング (HPC), Vol. 2016-HPC-157, No. 6 (2016).