Vol.2013-ARC-207 No.4
Vol.2013-HPC-142 No.4
2013/12/16

スレッド間空間的ブロッキングを利用した Xeon Phi上の姫野ベンチマークの最適化

小林 広和[1,a)]　中田　真秀[2,b)]

概要：本稿ではメニーコア・プロセッサーの Xeon Phi に向けて行った姫野ベンチマークの最適化手法について述べる．姫野ベンチマークを OpenMP を用いて並列化したコードに対し，Xeon Phi 向けにメモリーバンド幅を最適化する手法として公開されている最適化や配列のアライメント，配列の次元の入れ替えを適用し，最適化を行った．それに対し，スレッド間空間的ブロッキングを提案し適用した．スレッド間空間的ブロッキングとは複数のスレッドのデータを利用して空間的ブロッキングを行いキャッシュのヒットミスを軽減する効果を期待する手法である．これを最適化した姫野ベンチに適用することでキャッシュのヒット率が改善され性能が約 10.9%向上できることが確認できた．この結果，L サイズのベンチマークで 85.7GFlops（ECC On 時）および 96.3GFlops（ECC Off 時）の性能を達成することができた．

Optimization of Himeno Benchmark on Xeon Phi using Inter-thread Spatial Blocking

Abstract: This paper presents the way about optimizaton of Himeno benchmark on Xeon Phi. Himeno benchmark code is parallelized using OpenMP, and it is optimized by application of several optimization methods. These methods are the memory bandwidth optimization which is published for Xeon Phi, alignment of array and exchange of array dimensions. And then Inter-thread spatial blocking is proposed to optimize Himeno benchmark. Inter-thread spatial blocking is a method which does spatial blocking using data from several threads to reduce cache misses. It is applied to the optimized Himeno benchmark code, and the cache hit rate is improved and the performance is increased by 10.9%. As a result, Himeno benchmark performance is 85.7GFlops (ECC On) and 96.3GFlops (ECC Off) for L size benchmark.

1. はじめに

近年プロセッサーの多コア化が進み，61 コアのプロセッサーコアを持つ Xeon Phi™ コプロセッサーが Intel 社より発売された（以下 Xeon Phi と略す）．Xeon Phi は IA-32 のコアをベースとしているために既存の CPU 向けのコードを若干修正してリコンパイルすることでプログラムを実行することが可能である．また，IA-32 に用いた既存の開発環境やプログラム手法を利用することが可能であるため，プログラムの移植性は良いと考えられている．しかし，プログラムの移植性が良いからといって，簡単に高い性能が得られるわけではない．既に発表されたいくつかの論文 [7], [10] において Xeon Phi の性能評価がなされているが，理論的な性能から予測される性能に比べ性能が出ていないと考えられる評価も在る．このように Xeon Phi のようなメニーコアプロセッサーにおいては，ハードウェアの特性にあわせた最適化や実行環境の設定が必要となる．

本稿では，姫野ベンチマークを対象とし Xeon Phi 向けに最適化を行なった結果を報告する．姫野ベンチマークはメモリーバンド幅に大きく影響されることが知られている [8] ため，OpenMP で並列化を行った後に，Intel 社がメモリーバンド幅を最適化する方法として公開している手法を適用した．さらに配列のアライメント，配列の次元の入れ替えを適用した．それに対し，キャッシュミスを軽減できるようにスレッド間空間的ブロッキングを提案し，適用した．その結果，提案したスレッド間空間的ブロッキングを用いることで，それを適用しない場合に比べ約 10.9%の

[1] インテル (株)
Intel K.K.
[2] 理化学研究所
RIKEN
[a)] hirokazu.kobayashi@intel.com
[b)] maho@riken.jp

表 1　Xeon Phi のキャッシュ階層
Table 1　Cache hierarchy of Xeon Phi

	Size per core	Latency
L1 Data	32KB	1
L1 Instruction	32KB	1
L2	512KB	11

性能向上が達成できることが確認できた．

以降，第 2 章で Xeon Phi の特徴について述べる．さらに，第 3 章で姫野ベンチマークに対して行った最適化について述べる．第 4 章において，本稿で提案するスレッド間空間的ブロッキングについて説明し，それを利用した最適化を行った．第 5 章で性能評価結果を示し，第 6 章で関連研究について述べ，第 7 章では結論と今後の課題を示す．

2. Xeon Phi の特徴

Xeon Phi は，Xeon サーバーの PCI 拡張スロットに装着するカードとして提供されており，Xeon で動作するホスト OS からは MPSS と呼ばれるサービスを通してアクセスできる．

2.1 Xeon Phi のアーキテクチャー

Xeon Phi は最大 61 個のインオーダーのコアを持ち，各コアはハイパースレッド [*1] により 4 つのハードウェア・スレッドを同時に実行可能である．つまり最大 244 スレッドを実行できる．512bit 幅のベクトルを処理できる Vector Processing Unit(VPU) を持ち，倍精度小数では 8 要素，単精度小数では 16 要素を処理できる．

Xeon Phi のキャッシュ階層は**表 1** に示すようになっている．L1 キャッシュはデータとインストラクションのキャッシュが分離されており，L2 キャッシュでは共通となっている．キャッシュラインのサイズは L1 キャッシュ，L2 キャッシュともに 64 バイトである．キャッシュは各コアごとにプライベートとなっており，キャッシュとコアの組がリングインターコネクトによって相互に接続されている．リングインターコネクトにはさらにメモリーコントローラーと PCIexpress のクライアントロジックが接続されている（**図 1**）．

また，Xeon Phi では 4KB と 2MB のページテーブルをサポートしている．各コアは**表 2** に示すエントリーを持っている．

2.2 Xeon Phi のプログラミングモデル

Xeon Phi ではプログラムの実行方法としてオフロード実行とネイティブ実行がサポートされている．オフロード実行はホスト上のプログラムから Xeon Phi 上のサブルーチンや関数を呼び出す方法で，オフロード呼び出しのためのプラグマをソースコードに追加して Xeon Phi 上で実行する部分を指定し，インテルコンパイラーでコンパイルすることで実行ファイルを生成する．

それに対し，ネイティブ実行は Xeon Phi 上で直接プログラムを実行する方法でありインテルコンパイラーで Xeon Phi 向けにプログラムをコンパイルすることでネイティブ実行のための実行ファイルが生成される．

本稿で実装した姫野ベンチマークはネイティブ実行モデルで動作する．

[*1] Xeon プロセッサーのハイパースレッディングとは異なる仕組みで実装される
[*2] ©2013 Jim Jeffers and James Reinders. 文献 [2] より許可を得て転載

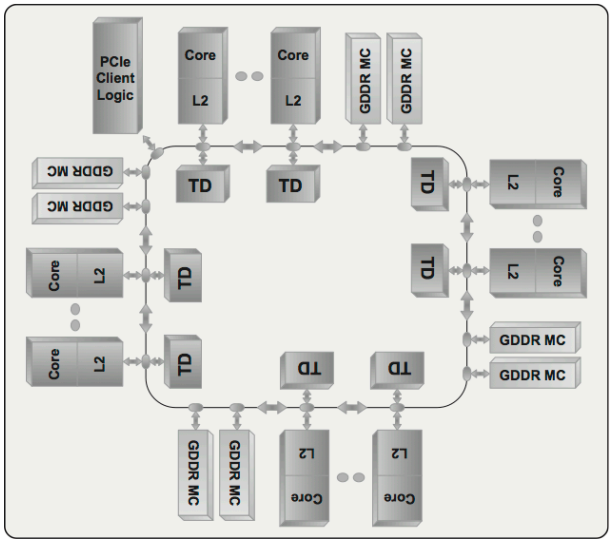

図 1　Xeon Phi のマイクロアーキテクチャー [*2]
Fig. 1　Microarchtechture of Xeon Phi

表 2　Xeon Phi のページテーブル
Table 2　Page Table of Xeon Phi

	Page Size	Entries
L1 Data TLB	4KB	64
L1 Data TLB	2MB	8
L1 Instruction TLB	4KB	32
L2 TLB	4K, 2M	64

3. 姫野ベンチマークの最適化

姫野ベンチマークは，理化学研究所の情報基盤センターセンター長の姫野龍太郎氏が非圧縮流体解析コードの性能評価のために考えたものでポアッソン方程式解法をヤコビの反復法で解く場合に主要なループの処理速度を計るものである [9]．姫野ベンチマークは理研の Web サイトからダウンロードすることが可能であり，プログラム言語の違いや，メモリーのアロケーション方法，並列化の有無などにより複数のバージョンが提供されている．本稿では C の static allocation バージョンをベースとした．

姫野ベンチマークでは圧力 P に対し 19 点のステンシルをとり，係数をかけ合わせて更新後の圧力を求めて配列

wrk2 に代入するというカーネルを実行する．このカーネルは 3 重ループとして各方向を処理するので，外側のループからそれぞれ Z 方向，Y 方向，X 方向の処理とする．オリジナルのコードでは，次のステップで利用する圧力 P を更新するために，カーネルの実行後に wrk2 の内容を圧力 P にコピーする処理が実行されるが，本稿ではこれをポインターの入れ替えで済ませるように変更したコードを利用する．

姫野ベンチマークはカーネルの処理を最初に 3 回実行し，その実行時の性能から本番のベンチマークでカーネルを実行する回数を決定するが，キャッシュミスなどの性能を比較するためにカーネルの実行回数を 3500 回と固定した．

本稿では姫野ベンチマークの L サイズを用いる．L サイズは格子のサイズが 256 x 256 x 512 となっている．

3.1 姫野ベンチマークの並列化

姫野ベンチマークを並列化するために，OpenMP のプラグマを追加した．並列化した姫野ベンチマークのカーネルコードを**図 2** に示す．`gosa` に対しては OpenMP の `reduction` 節を用いて総和を求める．また，`s0`, `ss`, `i`, `j`, `k` はスレッドにプライベートとなるように指定し，`p`, `wrk2` は初期値をグローバルな値からコピーしたのちにスレッドにプライベートとなるように指定をする．`p`, `wrk2` はポインターでありこのカーネルループの後に入れ替え処理が行われる．

単純に OpenMP の並列化プラグマを追加した場合，外側のループのみで並列化されるが，外側ループは 254 回しか回らないため Xeon Phi の 244 個のスレッドで分割すると，ほとんどのスレッドは `i` に対し 1 個の範囲しか担当しないが，`i` を 2 個担当する一部のスレッドが処理を完了するのを待つため性能が大幅に悪化する，これを避けるために，`collapse` 節を用いて Z 方向（インデックス値 `i`）と Y 方向（インデックス値 `j`）のループをバインドして並列化するように指示することで性能の向上が見込める [*3]．

3.2 メモリーバンド幅を最適化する手法の適用

Xeon Phi においてメモリーバンド幅を最適化するための手法は，文献 [5] において詳しく述べられてる．その手法をまとめると以下のようになる．

(1) 2MB ページを利用する [*4]．
(2) スレッド数は 60 で実行し，KMP_AFFINITY 環境変数を scatter とする．つまり OS のプロセス用に 1 コアを空けておき残りの各コアに 1 スレッドとなるようにスレッドを割り付ける．
(3) コンパイラーのオプションとして “`-O3 -mmic -openmp -opt-prefetch-distance=64,8 -opt-streaming-stores always -opt-streaming-cache-evict=0` ” を指定する．

指定された各コンパイラーオプションについて以下に説明する．

- “`-O3`” は最適化レベルを 3 に設定するオプションである．最適化レベル 3 はインテルコンパイラーがサポートしている中で最高レベルの最適化を行うことを指定する．
- “`-mmic`” は Xeon Phi 向けのネイティブコードを出力することを指定する．
- “`-openmp`” は OpneMP のプラグマで指示されたコードを並列化してコンパイルすることを指定する．
- “`-opt-prefetch-distance=64,8`” はメモリーから L2 キャッシュに対しては 64 キャッシュライン先を，L2 キャッシュから L1 キャッシュに対しては 8 キャッシュライン先をプリフェッチする命令を出力するということを指定する．
- “`-opt-streaming-stores always`” はストア命令としてストリーミングストア命令を利用することを指定する．
- “`-opt-streaming-cache-evict=0`” はストリーミングストアされたキャッシュラインに対して，キャッシュを追い出すための命令を出力しないことを指定する．

本稿ではメモリバンド幅向け最適化手法の適用として以下のことを行った．

スレッド環境変数の指定 スレッドの割り付け方法とスレッド数を設定するためにスレッド環境変数を指定した．スレッドの割り付け方法については `KMP_AFFINITY` 環境変数に `compact` を指定した．compact は OpenMP のスレッドを順に割り付けるときに，一つ前のスレッドになるべく近いハードウェアスレッドに割り付けるための指定である．Xeon Phi の場合は，各コアに順に 4 スレッドずつ割り付けられる．実行スレッド数は 1 コアを OS のカーネルのためにあけるために，240 とした．

コンパイラーオプションの指定 これらのオプションを姫野ベンチマークのコンパイル時に指定した．ただし，最適なプリフェッチの距離はアプリケーションによって変わるため，複数のパラメータを試した．その結果，性能が良かった `4,1` を指定することにした．

3.3 配列のアライメント

Xeon Phi では，64 バイトにアライメントされたロードストアとそうでないロードストアで異なった命令が利用され，アライメントされたロードストアの方が効率が良い．32bit 浮動小数を 1 レジスタ分ロードする場合の例を挙げ

[*3] Intel の堀越将司氏の指摘による
[*4] 最新の MPSS では 2MB ページを自動的に利用できるので，特に意識する必要はない

```
#pragma omp parallel for private(s0,ss,i,j,k) firstprivate(p,wrk2) reduction(+:gosa) collapse(2) schedule(static)
    for(i=1 ; i<imax-1 ; i++)
      for(j=1 ; j<jmax-1 ; j++)
        for(k=1 ; k<kmax-1 ; k++){
          s0 = a[0][i][j][k] * p[i+1][j  ][k  ]
             + a[1][i][j][k] * p[i  ][j+1][k  ]
             + a[2][i][j][k] * p[i  ][j  ][k+1]
             + b[0][i][j][k] * ( p[i+1][j+1][k  ] - p[i+1][j-1][k  ]
                               - p[i-1][j+1][k  ] + p[i-1][j-1][k  ] )
             + b[1][i][j][k] * ( p[i  ][j+1][k+1] - p[i  ][j-1][k+1]
                               - p[i  ][j+1][k-1] + p[i  ][j-1][k-1] )
             + b[2][i][j][k] * ( p[i+1][j  ][k+1] - p[i-1][j  ][k+1]
                               - p[i+1][j  ][k-1] + p[i-1][j  ][k-1] )
             + c[0][i][j][k] * p[i-1][j  ][k  ]
             + c[1][i][j][k] * p[i  ][j-1][k  ]
             + c[2][i][j][k] * p[i  ][j  ][k-1]
             + wrk1[i][j][k];

          ss = ( s0 * a[3][i][j][k] - p[i][j][k] ) * bnd[i][j][k];
          gosa+= ss*ss;
          wrk2[i][j][k] = p[i][j][k] + omega * ss;
    }
```

図 2 並列化した姫野ベンチマークのカーネル
Fig. 2 Parallelized Himeno benchmark kernel

ると，アライメントの揃ったロードは vmovaps の 1 命令で実行できるが，アライメントの揃っていないロードでは，vloadunpackld と vloadunpackhd の 2 命令を使用する必要がある．インテルのコンパイラーはアライメントの揃ったロードストアを活用するために，ループのピーリングを行う．

姫野ベンチマークのカーネル内のすべての配列アクセスに対しアライメントを揃えたアクセスとすることは不可能であるが，なるべく多くの配列のアライメントが揃った状態とすることは可能である．このために 4 次元配列として宣言されている配列 a,b,c を一番左側の次元で分解して 3 次元配列とした．例えば a については a0,a1,a2,a3 の 4 つの 3 次元配列として宣言した．さらに，3 次元配列化された 4 次元配列も含むすべての 3 次元配列に__declspec(align(64,60))を追加した．これは 64 バイトにアライメントされたアドレスから 60 バイトずらしたアドレスに配列の先頭を配置するという指定である．これを追加することで，コンパイラーがピーリングを行った後は，P 以外のすべての配列へのアライメントが揃い，P の配列へのアクセスも 19 点のうち 9 点に対しアライメントの揃ったアクセスとすることができる．

3.4 配列の次元の入れ替え

文献 [8] において指摘されているように，配列の次元の宣言順序を変更することにより TLB ミスを削減することが可能である．係数配列は 4 次元として宣言されているが，これを 3 次元の配列が 12 個存在すると考えることができる．宣言の順序を変更すると，coef[12][MIMAX][MJMAX][MKMAX] という宣言が coef[MIMAX][12][MJMAX][MKMAX] という宣言となる．さらに，前節で行った配列のアライメントの最適化を適用するために coef[MIMAX][12][MJMAX*MKMAX+PADDING] というように宣言を行った．PADDING は MJMAX*MKMAX+PADDING が 16 の倍数になる最小値とした．

4. スレッド間空間的ブロッキングの提案と適用

前章において最適化した姫野ベンチマークではブロッキングなどの処理を行っておらず，メモリーアクセスの局所性についてはほとんど考慮していない．それに対し，メモリーアクセスの局所性を増す手法としてスレッド間空間的ブロッキングという手法を提案する．スレッド間空間的ブロッキングはデータを空間によって分割する空間的ブロッキングの一種であり，複数のスレッドのデータを利用したブロッキングによりキャッシュミスを削減することを目的とする．

ここでは，Z 方向のキャッシュミスを減らすためのスレッド間空間的ブロッキングについて考える．ここで，スレッド番号 n のスレッドが z=n となる領域の圧力の計算を担当するように処理を分割すると仮定する．そうすると，z=n のときの圧力を計算するスレッドの担当する領域は**図 3** の濃い色の部分となり，そのスレッドが圧力を求めるためにアクセスする圧力は z=n-1 と z=n+1 の領域であるので，図 3 の薄い色の部分となる．このとき，アクセスする圧力の領域は n-1 番目のスレッドと n+1 番目のスレッドが計算を担当する領域となる．つまり，あるスレッドの

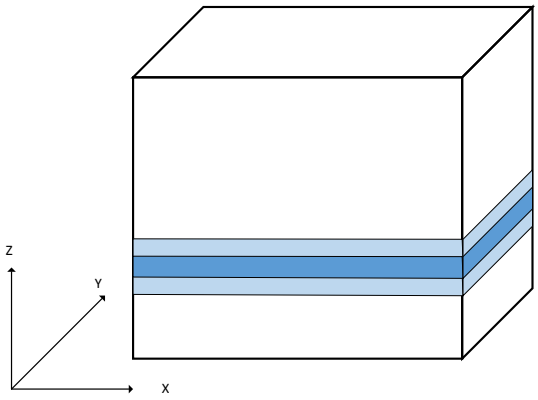

図 3　計算を担当する領域と圧力をアクセスする領域
Fig. 3　Responsible region for computation and region which access pressure

担当する計算に必要なデータは隣のスレッドの必要とするデータとオーバーラップする状態となる．したがって，スレッド番号が隣接するスレッド同士がなるべく近くになるようにスレッドを配置すると，隣のスレッドのキャッシュの内容を活用できるようになり，メモリーへのアクセスが減少すると考えられる．つまり複数のスレッドにまたがってデータの再利用を促すブロッキングを行うことになる．

以上の手法を前章までで最適化した姫野ベンチマークに適用した．L サイズの姫野ベンチマークでは Z 方向は 254 回のループが回るのに対し，Xeon Phi で利用するスレッド数は 240 となる．そのため，各スレッドが Z 方向を 1 個担当することにすると 14 個のあまりが発生する．したがって，まず各スレッドが Z 方向を 1 個ずつ担当するように処理を行った後に，発生した余りを全スレッドで均等に分割して処理を行うようにした．つまり，Z 方向を 1 個ずつ担当するループと余りを処理するループの二つのループを順に実行して，1 ステップの処理を完了する．それぞれのループには `collapse(2)` を指定した．これを余りのループに指定するのは処理を均等に分割するためであるのに対し，最初のループに指定するのは，その方が性能が良くなるからである．`schedule(static)` を指定していると最初のスレッドから順に均等に処理を分割するので，`collapse` によりループをバインドしても各スレッドが Z 方向を 1 ずつ担当することになる．この指定で性能が良くなる理由は，分岐が減るためであると考えられる．

スレッドの割り付けの環境変数は `compact` を指定した．こうすると，各コアに順に 4 スレッドずつスレッド番号順に割り付けられるようになる．

図 4 にこのときのスレッドとコアの関係を示す．図の四角形はコアを表し，その上に楕円で表されたスレッドが割り付けられている．スレッド内のテキストは担当する Z の値を表している．スレッド間の片方向の矢印はデータのアクセスを表し，矢印の始点のスレッドから矢印の終点のスレッドのデータをアクセスすることを示す．また図の右側

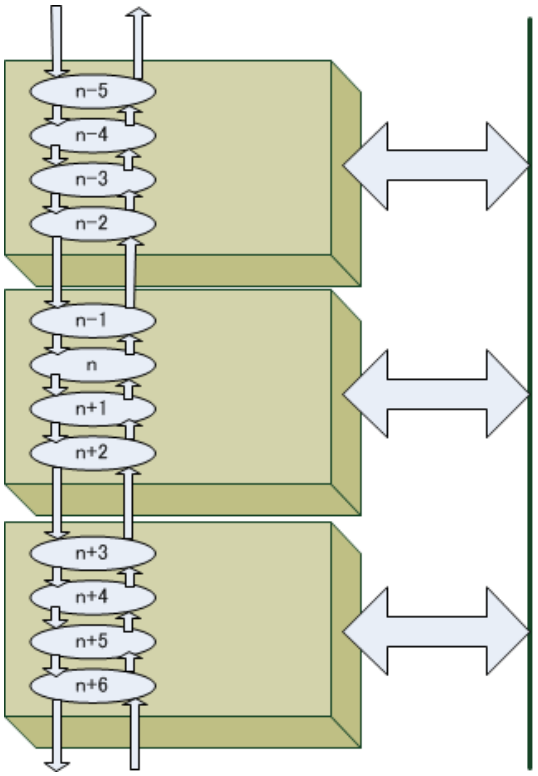

図 4　コアへのスレッドの割り付けの例
Fig. 4　Example of thread allocation to core

の縦線はリングインターコネクトを表し，リングインターコネクトとコアの間の両方向の矢印がこれらが相互に接続されていることを示している．以下の説明では Z の値が n の領域を担当するスレッドをスレッド n と呼称する．

スレッド n に着目するとスレッド n-1 とスレッド n+1 のデータをアクセスし，逆にスレッド n-1 とスレッド n+1 からはデータをアクセスされている．これらのアクセスは同一コア上のアクセスであるので，キャッシュ上に存在する他のスレッドが担当するデータを直接利用することができる．図の例ではスレッド n-1 から n+2 までは同一コア上に存在し，これらのスレッド間のデータアクセスはキャッシュにヒットする．それに対しスレッド n-1 からスレッド n-2 へのデータのアクセスは，スレッドが同一コア上に存在しないので，リングインターコネクトを経由することになる．Xeon Phi はインクルーシブキャッシュとなっているので，スレッド n-2 の存在するコアの L2 キャッシュからデータを取得することになる．このとき図に示すように，スレッド n-2 が割り付けられたコアとスレッド n-3 の割り付けられたコアはリングインターコネクトを介して最も近い距離にあるコア同士であり，隣接するコアとなっている．同様なことがスレッド n+2 とスレッド n+3 に関しても言える．このように任意のスレッドのアクセスするデータは同一コア上のスレッドか隣接コアのスレッドが担当するデータとなっている．

ここで，圧力 P の使用するキャッシュメモリーのサイズ

について検討する．X 方向には 512 要素存在するので，X 方向を一列処理するためには 2KB のメモリーが必要となる．Y 方向には同時に 3 列アクセスする必要があるので，X 方向について分割を行わない場合は 6KB のメモリーが必要となる．このとき，Z 方向は隣接スレッドがデータを保持しているので自分のスレッドのキャッシュ上に載せる必要はない．したがって，X 方向に 1 列ずつ処理を行う場合には，6KB のキャッシュメモリーが必要となる．Xeon Phi の 1 次データキャッシュのサイズは 32KB であるので，1 コアに 4 スレッド割り付けた場合には各スレッドは 8KB のキャッシュを利用できることになる．つまり各スレッドが利用可能な 1 次キャッシュのサイズより必要なメモリーが小さいので，圧力 P は 1 次キャッシュ上に載っていることを期待できる．つまり 1 つのコアに注目すると `(4,3,512)` のサイズで空間を分割しブロッキングしたのと同等とみなすことが可能である．ただ，この条件が成立するのは隣接スレッドと処理の進行状況が同期している場合である．隣接スレッドと処理の進行状況にずれが生じると，隣接スレッドが保持しているはずの Z 方向のデータがキャッシュ上から追い出されているためにキャッシュミスとなる可能性がある．したがって，隣接スレッドと進行状況を同期しながら処理を行うのが望ましいが，OpenMP には軽量にスレッド間の同期を行う仕組みがないので本稿では処理の同期は行わないことにした．

5. 性能評価

前章までで実装した姫野ベンチマークについて性能評価を行った．キャッシュミスの測定には VTune Amplifier の API を用いて性能評価の対象となる部分のみの数値を測定した．Xeon Phi の stream ベンチマークの測定結果では ECC の on 時と off 時によって性能の違いが見られたので，本稿でも両方の場合について性能を測定した．

5.1 性能評価に利用した環境

本節は性能評価に利用した環境について述べる．まず Xeon Phi コプロセッサーのハードウェアは，1.243GHz の 61 コア，メモリー搭載量が 16GB のエンジニアサンプルを使用した．これは，Xeon Phi コプロセッサー 7120P と同一のスペックのサンプルであり，7120P との違いは空冷ファンの搭載の有無である [*5]．本稿ではターボモードを利用しなかったので，冷却システムによる性能の違いはなく 7120P と同じ性能と考えて差支えない．ホストマシンは，Xeon E5-2687W(Sandybridge-EP 3.1GHz) を搭載したマシンを利用したが，Xeon Phi のネイティブモードで実行したためホストマシンのハードウェア構成はテストに影響を与えないと考えることができる．

[*5] エンジニアリングサンプルには空冷ファンが搭載されている．

表 3 テストマシンのソフトウェア環境
Table 3 Software environment of test machine

ソフトウェア名	バージョン
ホスト OS	RHEL6.1
MPSS	gold_update3_3
Intel コンパイラー	2013 SP1
VTune Amplifier XE	2013 Update14

ソフトウェアの環境は，コンパイラーと MPSS についてはインテル社の提供している製品を利用した．さらにパフォーマンスカウンタを計測するツールとしてインテル社の VTune Amplifier を利用した．またホストの OS は Red Hat Enterprise Linux 6.1 を用いた．これらのソフトウェア環境については**表 3** にまとめる．

5.2 性能測定と評価

姫野ベンチマークにポインター入れ替えと並列化とを適用したプログラムをベースとし，第 3 章の最適化を順次適用した場合に第 4 章のスレッド間空間的ブロッキングがある場合とない場合の性能を測定した．測定は各 3 回行い，最も性能の良かった結果を GFlops 値で小数以下 2 桁目を四捨五入した．

その結果を**表 4** に示す．表の単位は GFlops である．性能比はスレッド間ブロッキングのない場合にスレッド環境変数の最適化を実施した場合を基準とし，この性能を 100 として示している．“並列化のみ” ではコンパイラーオプションとして `-openmp`, `-O3`, `-mmic` のみを指定し，スレッド割り付けやスレッド数を指定する環境変数はデフォルト値を使用している．

この結果から，並列化を行っても適切なスレッド環境変数の設定を行わないと大きく性能が劣ることが分かる．さらに基準値とすべての最適化を実施し最高の性能となった場合との性能の違いが 18.2%であることから，適切な環境変数やコンパイラーオプションの設定により，大きくソースコードを変更する最適化を行わなくてもかなりの性能が得られるということが分かる．これは汎用 CPU をベースにしているため既存の汎用 CPU 用のコードがそのまま利用できる Xeon Phi の利点と言える．また，スレッド間ブロッキングのある場合とない場合を，それ以外の最適化をすべて適用した場合の性能で比較すると約 10.9%の違いとなることが分かる．スレッド環境変数のみを適用した場合のスレッド間ブロッキングの効果は，他の場合に比べて大きくない．これは適切なプリフェッチ距離が設定されていないため，必要なデータがキャッシュ上にない場合が存在するからだと考えられる．

次にスレッド間ブロッキングのキャッシュの利用に関する効果を測定するために，VTune を用いてキャッシュの利用に関連するパフォーマンスカウンターを計測して比較し

表 4　姫野ベンチマークの性能測定結果 (ECC ON)
Table 4　Performance result of Himeno benchmark (ECC ON)

最適化名	性能 (GFLops)	
	スレッド間ブロッキング	
	なし (性能比)	あり (性能比)
並列化のみ	27.7 (38.2)	
メモリーバンド幅最適化		
[スレッド環境変数]	72.5 (100.0)	74.9 (103.3)
[コンパイラーオプション]	76.2 (105.1)	85.0 (117.2)
配列のアライメント	76.8 (105.9)	85.1 (117.4)
配列の次元入れ替え	77.3 (106.6)	85.7 (118.2)

表 5　キャッシュ利用に関するパフォーマンスカウンタの比較
Table 5　Comparison of performance couter about cache utilization

イベント名	スレッド間ブロッキング	
	なし	あり
`L1 Miss`	8.7E+10	7.7E+10
(L1 HIT Ratio)	0.78	0.81
`L2_DATA_READ_MISS_CACHE_FILL`	2.8E+08	4.0E+09
`L2_DATA_READ_MISS_MEM_FILL`	1.0E+11	9.3E+10
`L2_DATA_WRITE_MISS_CACHE_FILL`	7.0E+09	6.9E+09
`L2_DATA_WRITE_MISS_MEM_FILL`	2.0E+08	2.2E+08

表 6　姫野ベンチマークの性能測定結果 (ECC OFF)
Table 6　Performance result of Himeno benchmark (ECC OFF)

最適化名	性能 (GFLops)	
	スレッド間ブロッキング	
	なし (性能比)	あり (性能比)
並列化のみ	27.7 (34.5)	
メモリーバンド幅最適化		
[スレッド環境変数]	80.3 (100.0)	83.2 (103.6)
[コンパイラーオプション]	84.9 (105.7)	95.6 (119.1)
配列のアライメント	85.3 (106.2)	94.7 (117.9)
配列の次元入れ替え	85.8 (106.8)	96.3 (119.9)

た．比較では，VTune の API を用いて性能測定に用いられる部分のみのカウンターを取得するようにした．この結果を**表 5** に示す．表の単位は L1 HIT Ratio 以外は回数であり，L1 HIT Ratio は 1 を最大とした割合である．`L1 Miss` を見るとスレッド間ブロッキングを行った方が L1 キャッシュのミスが約 12%減少していることが分かる．キャッシュのヒットレートで見ると 0.03 の改善である．また L2 のキャッシュミスに関しては，ライトはスレッド間ブロッキングの影響はほぼないが，リードは他のコアのキャッシュからデータを取得する場合（`L2_DATA_READ_MISS_CACHE_FILL`）が 1 桁増加し，メモリーからデータを取得する場合（`L2_DATA_READ_MISS_MEM_FILL`）が約 9%減少している．これはスレッド間ブロッキングが意図したように隣接コアのデータをうまく利用できていることを示している．

文献 [5] では，ECC OFF の時に stream ベンチマークの値が向上することを指摘しているため，ECC OFF の時の性能を**表 6** に示す．その結果 ECC OFF の時は最高性能が 96.3GFLOPS となることが分かった．スレッド間ブロッキングがありのときに配列のアライメントを適用した場合に性能の低下が見られるが原因については不明である．

6. 関連研究

Xeon Phi 上の姫野ベンチマークは論文 [7] によって性能評価を行われている．この論文では OpneMP での並列化を行った結果 L サイズで 43.6GFLOPS の性能が得られたとしている．最適化については言及されていないので，特別な最適化はなされていないと考えられる．また，ECC についても言及されていないのでデフォルト値である ECC ON にて測定したと考えられる．これに対し，本研究では ECC ON 時に 85.7GFLOPS の性能が得られ 1.97 倍の性能となった．

GPU 向けに姫野ベンチマークを最適化した研究として [4], [8] がある．これらの研究の手法はアーキテクチャーが異なるので本稿にはそのまま適用することはできなかったが，その一部については Xeon Phi 向け最適化においても有効であった．

ステンシルに対する空間的ブロッキングとして文献 [1] では自動チューニングを用いてブロッキングのパラメータを求める手法を提案している．本稿の手法は空間的ブロッキングの一種であるが，1 スレッド内でブロッキングを行うのではなく複数のスレッドにまたがってブロッキングを行うことが新しい点である．また，ブロッキングに必要なループネストの増加がなくそれに伴うオーバヘッドがないことが利点であるが，任意のサイズでブロッキングができないのが欠点である．

さらに空間的ブロッキングと組み合わせて時間方向にブロッキングを行う方法も提案されている [3], [6] ．この方法はテンポラルブロッキングと呼ばれる．文献 [3] では，テンポラルブロッキングはデータの再利用効率が高いがプログラミングが複雑になることが指摘されている．本稿の手法はテンポラルブロッキングと比較してプログラミングが複雑でないことが利点である．また，[6] ではテンポラルブロッキングを複数のスレッドでパイプライン処理することにより実現している．複数のスレッドでブロッキングしていることは本稿の手法と同じであるが，本稿の手法は空間的ブロッキングを利用しているのに対し，テンポラルブロッキングを利用していることが異なる．

7. 結論と今後の課題

本稿では，Xeon Phi 上において姫野ベンチマークに対して並列化，メモリバンド幅最適化，配列のアライメント，配列の次元の入れ替を行いその結果を測定した．その結果，適切に並列化された姫野ベンチマークではスレッド

に関する環境変数を適切に指定することでかなりの性能を得ることが可能であり，スレッド間空間的ブロッキングも含めたすべての最適化を実施した場合との性能の違いは 18.2%であることが分かった．さらにスレッド間空間的ブロッキングの手法を提案し，それがキャッシュミスを減少させるのに有効に働き，その手法を適用すると性能を約 10.9%改善できることを示した．これらの最適化の結果，姫野ベンチマークの性能が 85.7GFlops(ECC On 時) および 96.3GFlops(ECC Off 時) となることが分かった．

今後の課題として以下の 3 点が挙げられる．

- 姫野ベンチマークの他のサイズに対して提案したスレッド間空間的ブロッキングを適用してみること．
- Xeon などの他のアーキテクチャのマシンにおけるスレッド間空間的ブロッキングの効果を計測すること．
- 他のステンシル計算を行うアプリケーションに対して本稿で提案した方法を用いた場合の効果を測定すること．

本稿で対象とした姫野ベンチマークの L サイズでは外側ループのイテレーションの回数が 254 回と本稿で用いたスレッド数の 240 よりわずかに大きいという都合のいい回数であった．これが例えば M サイズになると 128 回となるので，どのように分割するのが最良であるのか考慮する必要がある．

Xeon Phi は 1 コア 4 スレッドとなっているのに対し Xeon も 1 コア 2 スレッドをサポートしておりスレッド間空間的ブロッキングを適用することが可能である．コア数が Xeon Phi 比べ少なくコア当りのスレッド数も少ないマシンにおける効果を評価する必要がある．

本稿で提案したスレッド間空間的ブロッキングは他のステンシルでも効果が発揮できると考えられるので，実際の効果を測定し評価する必要がある．

謝辞 本稿に対しインテルの堀越将司氏，池井満氏，菅原清文氏によりご意見を頂き，内容の充実を図ることができたことを深謝する．

参考文献

[1] Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J. and Yelick, K.: Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures, *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pp. 1–12 (2008).

[2] Jeffers, J. and Reinders, J.: *Intel Xeon Phi Coprocessor High Performance Programming*, Morgan Kaufmann (2013).

[3] Nguyen, A., Satish, N., Chhugani, J., Kim, C. and Dubey, P.: 3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs, *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pp. 1–13 (2010).

[4] Phillips, E. and Fatica, M.: Implementing the Himeno benchmark with CUDA on GPU clusters, *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pp. 1–10 (2010).

[5] Ramen, K.: Optimizing Memory Bandwidth on Stream Triad, Intel (online), available from ⟨http://software.intel.com/en-us/articles/optimizing-memory-bandwidth-on-stream-triad⟩ (accessed 2013-2-19).

[6] Wellein, G., Hager, G., Zeiser, T., Wittmann, M. and Fehske, H.: Efficient Temporal Blocking for Stencil Computations by Multicore-Aware Wavefront Parallelization, *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, Vol. 1, pp. 579–586 (2009).

[7] 高橋航平，塙 敏博，朴 泰祐，児玉祐悦，扇谷 豪，佐藤三久：各種アプリケーションにおける GPGPU 対 Many Core Processor の性能比較，情報処理学会研究報告. [ハイパフォーマンスコンピューティング]， Vol. 2013, No. 21, pp. 1–7 (2013).

[8] 成瀬 彰，住元真司，久門耕一：GPGPU 上での流体アプリケーションの高速化手法：1GPU で姫野ベンチマーク 60GFLOPS 超 (高性能計算とアクセラレータ)，情報処理学会研究報告. [ハイパフォーマンスコンピューティング]， Vol. 2008, No. 99, pp. 49–54 (2008).

[9] 理化学研究所情報基盤センター：姫野ベンチマーク，理化学研究所（オンライン），入手先 ⟨http://accc.riken.jp/2145.htm⟩ （参照 2013-10-18）.

[10] 大島聡史，金子 勇：メニーコアプロセッサ Xeon Phi の性能評価，情報処理学会研究報告. [ハイパフォーマンスコンピューティング]，Vol. 2013, No. 20, pp. 1–6 (2013).