Vol.2016-HPC-155 No.37
2016/8/10

ステンシル計算コードの性能とメモリレイアウトの関係性について

佐藤 真平[1,a)]　佐藤 幸紀[2,b)]　遠藤 敏夫[2,c)]

概要：アプリケーションの性能チューニングにおいて，想定するピーク性能と実際の性能のギャップを生じさせる原因を明らかにすることが重要である．今後の高性能計算機ではメモリ階層が複雑化すると予測されており，メモリの性能を引き出すことがアプリケーションの性能チューニングにおいて大きな要素となる．我々はステンシル計算コードを対象にメモリレイアウト最適化による性能チューニングに取り組んでいる．本稿では，ステンシル計算コードにおいて様々なパターンのパディングを実施し，メモリレイアウト最適化の結果が実際の性能に与える影響を調査し，考察する．結果，メモリレイアウトによってはハードウェアプリフェッチが働くものと働かないものがあり，姫野ベンチマークにおいてはその性能差が2倍近くなることがわかった．

1. はじめに

エクサスケール時代に向けた高性能計算機システムにおいて，高速計算を実現するためには数千から数万に及ぶ並列性を抽出する大規模並列処理が必要である．一方で，1ノードもしくはCPU単体におけるアプリケーションの性能は，大規模並列化後の性能を決めるベースとなる重要な要素である．例えば，1ノードあたりの性能を1割ほどしか引き出せていないアプリケーションの大規模並列化後の性能は，たとえ並列化効率が良くても，高々システム全体の1割程度の性能しか引き出せないことになる．今後の高性能計算機システムにおいても，1ノードあたりの性能チューニングは重要な課題である．

近年の計算機システムにおける1ノードあたりの性能チューニングでは，メモリウォール問題のために複雑化するメモリ階層への対応，コア数の増加やSIMDを用いたプロセッサ内における並列性の抽出など，ハードウェアをいかに活用するかが求められる．Intelでは，CやC++で単に書かれたコードとマルチコア，メニーコアプロセッサ向けに最もチューニングされたコードの性能差を「Ninja Gap」と称し，コンパイラの支援などを利用したユーザーフレンドリーなチューニングでその性能差を縮める取り組みを進めている [1]．この取り組みでは，最もチューニングされたコードと比較して平均で24倍のスローダウンであったコードが，コンパイラへのヒントを挿入することにより平均で1.3倍のスローダウンまで性能向上を達成することが報告されている．

アプリケーションの性能チューニングを実施するにあたり，対象となるアプリケーションを実機で実行したときに何が起きているのかを把握することが第一歩となる．我々は，アプリケーション性能解析ツールExana [2], [3], [4]の開発を進めている．このツールは，ハードウェアカウンタなどの実機からは得られない詳細なアプリケーションの解析を基に，アプリケーションのどこをチューニングすればよいかというヒントをユーザーにフィードバックすることを目的としている．

性能チューニングのケーススタディとして，我々はステンシル計算コードを対象にメモリレイアウト最適化による性能チューニングに取り組んでいる [5], [6]．ステンシル計算の性能はメモリバンド幅に影響されることが知られており，キャッシュを効率よく使うことが性能チューニングの鍵となる．これまでの取り組みから，パディングによってキャッシュコンフリクトミスを解消し，ステンシル計算コードの性能向上が得られることを確認している．また，Exanaのメモリ階層性能シミュレータを用いて，パディングによってキャッシュコンフリクトが解消されていることを確認している．

[1] 東京工業大学 工学院情報通信系
Dept. of Information and Communications Engineering, Tokyo Institute of Technology
[2] 東京工業大学 学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology
[a)] satos@ict.e.titech.ac.jp
[b)] yukinori@el.gsic.titech.ac.jp
[c)] endo@is.titech.ac.jp

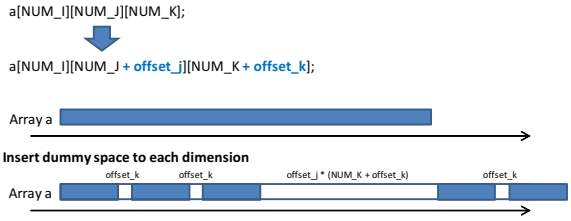

図 1 イントラアレイパディングの例

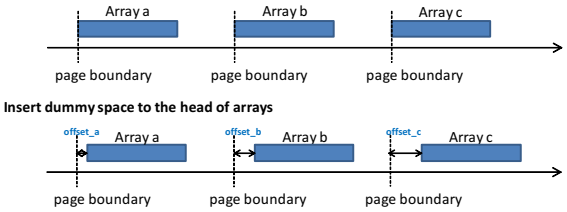

図 2 インターアレイパディングの例

しかし，メモリレイアウトによっては実機における性能に差がみられるものの，Exana では優位な差が確認できない場合がある．我々は，これをハードウェアプリフェッチによる影響であると考えている．近年のプロセッサではメモリアクセスのレイテンシを隠蔽するためにプリフェッチによりメモリからキャッシュに投機的にデータを移動させている [7], [8]．パディングによってメモリレイアウトを変更することで，プロセッサが不要なデータをプリフェッチするようになり，性能にばらつきがでることが考えられる．

本稿では，パディングによるメモリレイアウトがハードウェアプリフェッチに与える影響を調査する．ステンシル計算コードとして姫野ベンチマーク [9] を用い，パディングによる性能の変化とハードウェアプリフェッチによる性能の変化を Exana のメモリ階層性能シミュレータを用いて解析し議論する．

2. メモリレイアウト最適化

ステンシル計算のように，同時に複数の配列もしくは同時に 1 つの配列内の複数の領域を参照するようなアプリケーションでは，キャッシュコンフリクトミスが発生することが知られている [10]．キャッシュコンフリクトミスは，複数のキャッシュラインが同じセットを使用していしまい，連想度が不足するこで発生するキャッシュミスの一種である．キャッシュコンフリクトミスを軽減させる最適化として，多次元配列の要素間に適度にスペースを挿入するパディング [11] が一般的に用いられる．ここでは，メモリレイアウト最適化としてイントラアレイパディングとインターアレイパディングの 2 種類の方法 [6] について述べる．

2.1 イントラアレイパディング

イントラアレイパディングは，多次元配列の各次元にスペースを挿入するパディングである．例えば 3 次元 19 点ステンシル計算では 1 つの 3 次元配列内の 19 の要素を同時に参照する．連続となる参照方向を考慮しても，9 つの領域を同時に参照することになる．このような場合，1 つの配列内の参照同士でキャッシュコンフリクトミスが発生する可能性がある．そこで，配列の各次元にスペースを挿入することでキャッシュコンフリクトミスを軽減することができる．

図 1 を例にイントラアレイパディングの方法を説明する．コード中で 3 次元配列 $a[NUM\_I][NUM\_J][NUM\_K]$ が宣言されているとする．この配列はメモリ上の連続した領域に確保される．図では，イントラアレイパディングで 3 次元配列の 1 次元目と 2 次元目の要素に $offset\_k$，$offset\_j$ のサイズのスペースを挿入している．スペースを挿入された配列 a は，確保のためにより大きなメモリ空間が必要となる．実際にコード中で参照される領域はスペースを挿入する前と同じサイズとなり，図下部のような連続ではないメモリ参照パターンになる．

3 次元配列についてイントラアレイパディングを実施する場合には，3 次元目の要素にスペースを挿入する必要はない．配列の 3 次元目の要素にスペースを挿入しても，実際にコード中で参照される領域の後続の領域にスペースが挿入されるだけでメモリ参照パターンは変化しないためである．

ステンシル計算では，前述の配列 a を例に説明すると，$j+1$，$j-1$，$i+1$，$i-1$ といった要素を同時に参照する．そのような参照によって発生するキャッシュコンフリクトミスは，イントラアレイパディングによって解消することができる．

2.2 インターアレイパディング

インターアレイパディングは，配列の先頭にスペースを挿入するパディングである．例えばステンシル計算では，演算する値を保持する配列の他に係数が保持されている配列など，同時に複数の配列を参照する．同時に参照する配列が多ければ多いほど，その参照同士でキャッシュコンフリクトミスが発生する可能性がある．そこで，配列の先頭にスペースを挿入することでキャッシュコンフリクトミスを軽減することができる．

図 2 を例にインターアレイパディングの方法を説明する．コード中で配列 a，b，c が宣言されているとする．また，これらの配列はページ境界に確保されているとする．[*1] インターアレイパディングでは，それぞれの配列の先頭に $offset\_a$，$offset\_b$，$offset\_c$ という異なるサイズのスペースを挿入する．配列は，それぞれページ境界から異なったサイズ分ずれたメモリ上の領域に確保される．

ステンシル計算において，係数が保持されている配列を

[*1] 例えば valloc 関数を用いるとページ境界にデータを配置することができる．

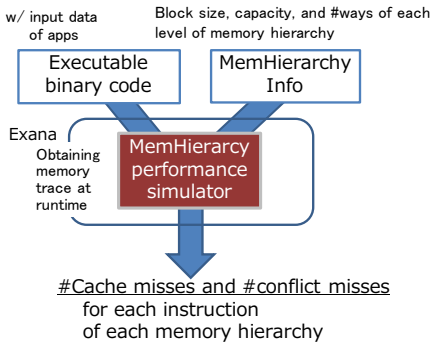

図 3 Exana のメモリ階層性能シミュレータの概要

参照する時は，同じインデックスの要素を参照することが多い．また，malloc などで大きなメモリ領域を確保する場合は，先頭がページ境界からの相対位置が同じになるように整列して確保されることが多く，異なる配列の同じインデックスの参照によってキャッシュコンフリクトミスが発生する可能性がある．そのような場合に，インターアレイパディングによってキャッシュコンフリクトミスを解消することができる．

3. Exana のメモリ階層性能シミュレータによる競合ミスの計測

アプリケーション性能解析ツール Exana [2], [3] のメモリ階層性能シミュレータ [4] を利用することで，キャッシュコンフリクトミスを計測することができる．**図 3** に Exana のメモリ階層性能シミュレータの概要を示す．性能を計測したいアプリケーションのバイナリと，メモリ階層の設定ファイルを入力として渡すと，シミュレーション結果としてキャッシュミスやコンフリクトミスといった統計データや，命令ごとの詳細なデータが出力される．

メモリ階層性能シミュレータとして実装されているキャッシュは，L1，L2，L3 キャッシュを搭載し，サイズ，ウェイ数，ブロックサイズを指定することができる．置き換えアルゴリズムは LRU を用いている．L3 キャッシュはインクルーシブポリシー，L1，L2 はノン・インクルーシブポリシーで実装されている．プリフェッチは未実装である．

キャッシュコンフリクトミスはキャッシュの容量に空きがある場合でも発生し，データを下層のメモリシステムに追い出してしまうため，アプリケーションの性能をチューニングにおいては，なるべく減らしたいミスである．また，キャッシュコンフリクトミスはハードウェアカウンタなどの実機から得ることは困難であり，アプリケーションの性能解析において重要な項目であるといえる．

Exana のメモリ階層性能シミュレータでは，設定された連想度によるシミュレーションと同時にフルアソシアティブによるシミュレーションを行い，タグの更新時にそれぞれの状態を比較してキャッシュコンフリクトミスを判定している．すなわち，キャッシュにおけるヒット・ミスの判定時に，設定された連想度のシミュレーションを行っているタグアレイではミスし，フルアソシアティブのシミュレーションを行っているタグアレイではヒットしたならばキャッシュコンフリクトミスとする．

4. メモリレイアウトとハードウェアプリフェッチ

ハードウェアプリフェッチ [7], [8] は連続的なメモリアクセスを検出して，投機的に下層のメモリから上層のメモリにデータを移動させる仕組みである．必要なデータを実際に参照されるより前に上層のメモリに移動させることができれば，そのメモリアクセスレイテンシを隠蔽することができ，アプリケーションの性能が向上する．

アプリケーションのメモリ参照パターンが連続もしくはストライドアクセスで一定の間隔でアクセスするようなパターンであれば，ハードウェアプリフェッチがうまく働き，性能向上が得られる．しかし，プリフェッチのアルゴリズムの設計に依存して検出できるストライドアクセスの間隔には限りがあり，ストライドアクセスではあるが間隔が大きい場合はプリフェッチが働かないことがある [7]．また同時に扱える連続アクセスの数にも限りがあり，同時に多数の連続な参照が発生する場合もプリフェッチが働かないことがある [8]．

ステンシル計算におけるハードウェアプリフェッチの働きを考える．ステンシル計算では複数の配列や 1 つの配列への連続な参照が多数発生している．連続なアクセスパターンとしてプリフェッチが働いていたメモリレイアウトが，イントラアレイパディングを実施し配列内にスペースを挿入することによって崩れ，プリフェッチが働かなくなることが考えられる．

5. 姫野ベンチマークを用いたメモリレイアウト最適化と性能の関係の調査

ステンシル計算コードとして姫野ベンチマーク [9] を用いてメモリレイアウトのハードウェアプリフェッチへの影響を調査する．メモリレイアウトの変更は，キャッシュやメモリにおいて競合を発生させる要因となり，それぞれのメモリ階層において性能に影響があると考えられる．本稿では，パディングによるキャッシュコンフリクトミスへの効果とハードウェアプリフェッチへの効果に注目して性能の調査，解析を行う．

5.1 実験環境

表 1 に実験で使用する計算機環境をまとめる．計算機は Ivy Bridge 世代の CPU，Intel Xeon E5-2650 v2 を 2 基搭載する NUMA 型のマシンである．コア数は CPU あたり 8 コア，16 スレッドである．実験では，ハイパースレッディングは使用せずに，1 コアに 1 スレッドを

表 1　実験に使用する計算機環境

プロセッサ	Intel Xeon E5-2650 v2 × 2
コア数	8（16 スレッド） × 2
動作周波数	2.60 GHz
単精度浮動小数点演算 ピーク性能（AVX 命令）	332.8 GFLOPS × 2
L1 データキャッシュ L2 キャッシュ L3 キャッシュ	32 KB, 8-way / core 256 KB, 8-way / core 20 MB, 20-way / CPU
メモリ	DDR3 (1866) 64 GB
OS	CentOS 7.2
コンパイラ	icc 15.0.2

割り当ててアプリケーションを実行する．動作周波数は 2.6 GHz で，ターボブーストは不使用とする．単精度の浮動小数点演算のピーク性能は，AVX 命令を使用し，かつ加算と乗算を同時実行する場合で，CPU あたり 8×2 inst. $\times 8$ cores $\times 2.6$ GHz $= 332.8$ GFLOPS となる．SSE 命令を使用する場合の単精度浮動小数点演算のピーク性能は，AVX 命令を使用する場合の半分の 166.4 GFLOPS である．キャッシュは L1，L2 がプライベート，L3 が共有となっており，サイズはそれぞれ 32 KB，256 KB，20 MB である．メモリは，64 GB 搭載している．8 GB のメモリモジュールを各 CPU に 4 枚ずつ搭載し，4 チャネルすべてを利用できる．OS は CentOS 7.2 である．アプリケーションのコンパイルには，インテルコンパイラ 15.0.2 を使用する．最適化オプションはすべての実験で O3 を使用する．また，SSE による SIMD 化を行う．

以降の実験において，アプリケーションは 1 スレッドで実行する．また，スレッドアフィニティ制御を行いスレッドは特定のコアに固定して実行する．

5.2　姫野ベンチマーク

ステンシル計算コードとして姫野ベンチマークを利用する．姫野ベンチマークは，非圧縮流体解析コードの性能評価のために考案されたベンチマークで，ポアソン方程式をヤコビの反復法を用いて解くものである．実験には，公開されている C + OMP, dynamic allocate version を利用する．また，キャッシュとハードウェアプリフェッチに注目するため，サイズは S を用いる．

姫野ベンチマークは 3 次元 19 点ステンシル計算を行うコードである．コード中では，計算領域や係数が格納されている 3 次元の領域が合計で 14 個使用されている．このうち，係数として用いられている 10 個の配列は 4 次元の配列としてメモリを確保する実装になっている．今回の実験では，イントラアレイパディングとインターアレイパディングを実施するためにすべての配列を 3 次元配列として扱いたい．そこで，14 個すべての配列を 3 次元配列としてメモリを確保するようにコードを変更する．また，後述するパディングで挿入するスペースのサイズをコマンドライン引数として入力できるようにコードを変更する．この変更で，すべての配列をページ境界に配置した場合の性能は，1 スレッド，SIMD 不使用のときに 1.18 GFLOPS となった．オリジナルのコードの性能は同じ環境下で 1.14 GFLOPS であり，コード変更による大きな性能変化はない．

実験は SSE による SIMD 化を施したコードを使用する．SIMD を使用した浮動小数点演算では，1 度に計算するデータのメモリアライメントが整っていると効率よくメモリ参照でき高い演算性能を達成することができる [12]．SSE 命令では 16 バイト境界にデータが整列しているとより演算性能が高くなる．姫野ベンチマークでは，演算は配列のインデックス (1, 1, 1) から演算をはじめる．したがって，SSE による演算性能を考慮して，インデックス (1, 1, 1) の要素が 16 バイト境界に整列するようにメモリアロケーションを調整する．

5.3　実機におけるメモリレイアウトと性能の調査

イントラアレイパディングとインターアレイパディングを施したメモリレイアウトを 1,000 通り用意し，CPU のハードウェアプリフェッチを使用しない場合と使用する場合の性能を計測し分析する．メモリレイアウトはパディングで挿入するスペースのサイズをランダムに決定し生成する．ハードウェアプリフェッチを使用しない場合の性能は純粋にキャッシュコンフリクトミスによる性能の変化を見ることができる．そして，同じメモリレイアウトを用いてハードウェアプリフェッチを使用する場合の性能から，メモリレイアウトごとのハードウェアプリフェッチの影響を見ることができる．

パディングで挿入するスペースのサイズ

イントラアレイパディングで挿入するスペースのサイズは，キャッシュの構成を考慮して以下の範囲で決定する．

- 1 次元目：[0 - 31] × 64 バイト
- 2 次元目：[0 - 15] 要素

実験で用いる計算機の CPU のキャッシュラインサイズは 64 バイトである．そのため，パディングによるキャッシュコンフリクトミスを軽減するには 1 ライン以上のスペースを挿入する必要がある．したがって，1 次元目に挿入するスペースは 64 バイト単位とする．また，L1 データキャッシュの構成を考えると，サイズ 32 KB，ウェイ数 8 のため，1 ウェイあたり 4 KB の連続したデータを格納することができる．したがって，1 次元目に挿入するスペースの上限は 4 KB の半分のサイズとし，0 から 31 の範囲でランダムに選択する．2 次元目に挿入するスペースは，1 次元目の要素数と挿入したスペースに依存する．したがって，スペースがあまり大きなサイズにならないように 0 から 15 の範囲でランダムに選択する．

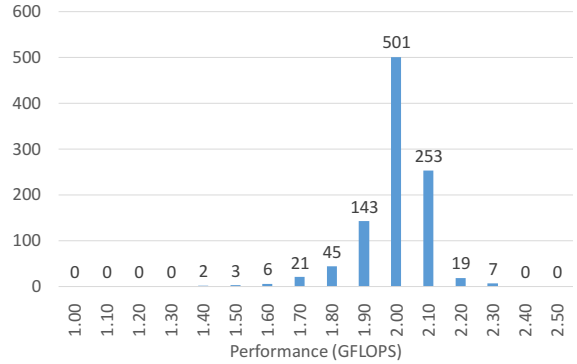

図 4 メモリレイアウトが異なる 1,000 通りの姫野ベンチマーク サイズ S の性能の分布．CPU のハードウェアプリフェッチを使用しない場合．

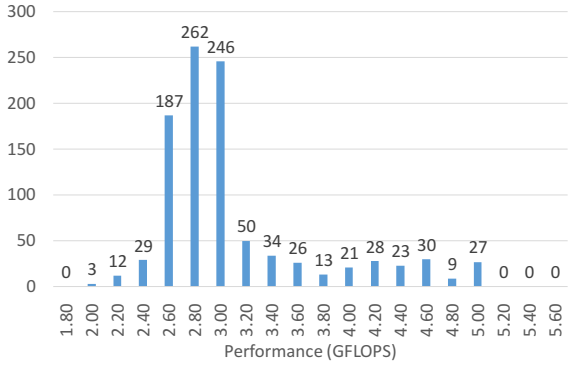

図 5 メモリレイアウトが異なる 1,000 通りの姫野ベンチマーク サイズ S の性能の分布．CPU のハードウェアプリフェッチを使用する場合．

インターアレイパディングで挿入するスペースのサイズもイントラアレイパディングと同様にキャッシュの構成を考慮して決定する．SIMD 化のためにメモリアライメントの調整を行っているが，valloc によりすべての配列がページ境界を基準に確保されている．1 キャッシュライン以上のスペースを挿入する必要があるため，64 バイト単位でスペースを挿入する．また，ページサイズが 4 KB のため，挿入するスペースの上限を 4 KB とし，0 から 63 の範囲でランダムに選択する．

イントラアレイパディングはメモリレイアウトごとに 1 パターン生成する．したがってコード中に宣言されている配列はすべて同じイントラアレイパディングを施される．インターアレイパディングによって挿入されるスペースは配列ごとに異なるサイズとする．姫野ベンチマークでは 14 個の配列が宣言されているため，ランダムに 14 個のサイズを生成する．したがって，生成可能なメモリレイアウトは $32 \times 16 \times 64^{14}$ 通りあり，そのうちの 1,000 通りのレイアウトを用いて実験を行う．1,000 通りのレイアウトの中にはパディングを実施しない，すなわちすべてのサイズを 0 とした場合（以降，このレイアウトを **Zero** と呼ぶ）は含まれない．

ハードウェアプリフェッチを使用しない場合の性能の調査

図 4 に，サイズ S の姫野ベンチマークについて，1,000 通りのメモリレイアウトを用いて実行したときの性能の分布を示す．CPU のハードウェアプリフェッチを使用しない場合の評価結果である．横軸は性能で単位は GFLOPS で示している．例えば，2.00 GFLOPS のラベルの場合，2.00 GFLOPS 以上，2.10 GFLOPS 未満の性能を示したメモリレイアウトが 501 個あること意味する．

最も低い性能は 1.45 GFLOPS，最も高い性能は 2.37 GFLOPS であり，約 1.6 倍の性能差がある．Zero の性能は 1.26 GFLOPS である．

ハードウェアプリフェッチを使用する場合の性能の調査

表 2 1,000 通りのメモリレイアウトの性能についての偏差値による分類．Prefetch ON の場合の性能の平均は 3.19 GFLOPS，標準偏差は 623.88．Prefetch OFF の場合の性能の平均は 2.05 GFLOPS，標準偏差は 102.04.

	Prefetch ON 偏差値 55 以上	Prefetch ON 偏差値 45 未満
Prefetch OFF 偏差値 55 以上	71	50
Prefetch OFF 偏差値 45 未満	32	109

図 5 に，サイズ S の姫野ベンチマークについて，1,000 通りのメモリレイアウトを用いて実行したときの性能の分布を示す．CPU のハードウェアプリフェッチを使用する場合の評価結果である．横軸は性能で単位は GFLOPS で示している．

最も低い性能は 2.05 GFLOPS，最も高い性能は 5.13 GFLOPS であり，約 2.5 倍の性能差がある．Zero の性能は 3.66 GFLOPS である．

考察

ハードウェアプリフェッチを使用しない時に，Zero の性能は 1,000 通りのレイアウトのどれよりも悪い性能を示している．しかし，ハードウェアプリフェッチを使用する時に，Zero の性能は 1,000 通りのレイアウトの中の最も悪い性能よりも高い性能を示している．これは，パディングを実施しておらず配列内などにスペースが挿入されていないため，それぞれの配列への参照が連続的になり，ハードウェアプリフェッチによる性能向上が得られているからであると考えられる．

ハードウェアプリフェッチを使用しない場合と使用する場合のそれぞれで，1,000 通りのレイアウトの性能について標準偏差を求め，偏差値を求める．ハードウェアプリフェッチを使用しない場合の性能の平均は 2.05 GFLOPS，標準偏差は 102.04 である．ハードウェアプリフェッチを使用する場合の性能の平均は 3.19 GFLOPS，標準偏差は

表 3 4 つのレイアウトにおいてパディングで挿入されるスペース

	j	k	p	bnd	$wrk1$	$wrk2$	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$b[0]$	$b[1]$	$b[2]$	$c[0]$	$c[1]$	$c[2]$
A	4	0	23	14	34	29	51	25	53	4	0	59	26	20	5	58
B	8	0	19	38	27	28	5	25	3	6	20	10	1	39	57	21
C	11	30	22	0	17	58	12	27	43	54	61	47	22	31	52	34
D	0	26	6	33	0	58	3	63	59	41	53	47	58	5	44	59

表 4 4 つのレイアウトとパディングを実施しない場合の姫野ベンチマークの Exana による性能解析

	A	B	C	D	Zero
性能 (GFLOPS)					
Prefetch ON	5.04	5.09	2.67	2.76	3.66
Prefetch OFF	2.37	2.34	2.17	2.16	1.26
Cache miss					
L1	3.99 %	3.88 %	4.64 %	4.88 %	7.36 %
L2	98.73 %	98.31 %	98.83 %	96.85 %	27.56 %
L3	93.73 %	94.00 %	93.54 %	93.63 %	94.60 %
Conflict miss					
L1	0.00 %	0.00 %	0.00 %	1.98 %	72.18 %
L2	0.15 %	0.09 %	0.11 %	0.14 %	0.08 %
L3	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %

623.88 である．**表 2** に，それぞれの場合について性能の偏差値が 45 未満と偏差値が 55 以上の場合に分類してレイアウトの数をまとめる．ここで，ハーウェアプリフェッチを使用しない場合に偏差値が高いレイアウトは，キャッシュコンフリクトミスをうまく解消できるレイアウトであるとみることができる．したがって，どちらの場合も偏差値が高いレイアウトは，キャッシュコンフリクトミスをうまく解消できかつハードウェアプリフェッチが働くレイアウトであると考えられる．また，ハードウェアプリフェッチを使用しない場合に偏差値が高く，ハードウェアプリフェッチを使用する場合に偏差値が低いレイアウトは，キャッシュコンフリクトミスを解消できるが，ハードウェアプリフェッチがうまく働かないレイアウトであると考えられる．

5.4 Exana によるキャッシュコンフリクトミスの解析

Exana によるキャッシュコンフリクトミスの解析からメモリレイアウトのハードウェアプリフェッチへの影響を調べる．前節における偏差値による分類において，どちらの場合も高い偏差値を示したレイアウトの中から 2 つ（A, B），プリフェッチを使用しない場合は高い偏差値を示したがプリフェッチを使用した場合に低い偏差値を示したレイアウトの中から 2 つ（C, D）の合計 4 つのレイアウトを選び，Exana を用いて解析する．また，プリフェッチを実施しない場合（Zero）についても Exana で解析する．

表 3 に，選択した 4 つのレイアウトについてパディングで挿入するスペースの量をまとめる．表中の数字は，前述した挿入するスペースを決める際のランダムに選択された数を表している．j と k のカラムはイントラアレイパディングで挿入するスペースを表しており，それぞれ 3 次元配列の 2 次元目に挿入するスペースと 1 次元目に挿入するスペースである．p から $c[2]$ のまでのカラムはインターアレイパディングで挿入するスペースを表しており，それぞれコード中の配列名に対応している．

表 4 に，それぞれのメモリレイアウトについて Exana で解析した結果を示す．4 つのレイアウトは，Zero と比較すると L1 キャッシュにおけるコンフリクトミスを大幅に解消している．4 つのレイアウト同士では，キャッシュに関する性能に大きな違いは見られない．

5.5 議論

表 4 より，A, B と C, D の L1 キャッシュミスで 1 %弱の差が見られる．これは，ハードウェアプリフェッチを使用しない場合の性能差として現れていると考えられる．しかし，Exana による性能解析からはハードウェアプリフェッチを使用する場合の A, B と C, D の性能差を説明できる優位な差は見られない．これは，Exana のメモリ階層性能シミュレータではプリフェッチが未実装のため，ハードウェアプリフェッチの効果が解析結果に反映されないことが理由として考えられる．

以上から，パディングによりキャッシュコンフリクトミスを解消するメモリレイアウトには，ハードウェアプリフェッチが働くものと働かないものがあるといえる．そして，姫野ベンチマークでの性能差は 2 倍近くなることがわかる．

表 3 より，ハードウェアプリフェッチが働いている A と B のレイアウトについて，イントラアレイパディングで挿入される k のスペースがともに 0 となっている．k のスペースは 3 次元配列の 1 次元目に挿入するスペースであ

り，このスペースが 0 ということは $j \times k$ の平面が連続な領域としてメモリ上に確保されることになる．また，ハードウェアプリフェッチが働いていない C と D のレイアウトでは，k に挿入されるスペースがとても大きい．ここから，1 次元目に挿入するスペースを小さくするとハードウェアプリフェッチを働かせることができると考えられる．

6. まとめと今後の課題

エクサスケール時代に向けた高性能計算機システムにおいて，高速計算を実現するためには数千から数万に及ぶ並列性を抽出する大規模並列処理が必要である．一方で，1 ノードもしくは CPU 単体におけるアプリケーションの性能は，大規模並列化後の性能を決めるベースとなる重要な要素である．

本稿では，ステンシル計算コードを対象に，パディングによってスペースを挿入したメモリレイアウトが実際の性能に与える影響を調査した．調査の結果，メモリレイアウトによってはパディングによる性能向上は得られるものの，ハードウェアプリフェッチがうまく働かなくなり，十分な性能向上が得られないメモリレイアウトがあることがわかった．姫野ベンチマークでは，その性能差が 2 倍近くになることが確認できた．

今後の課題としては，メモリレイアウトによる性能のばらつきに関するより詳細な分析，メモリレイアウトによって DRAM において発生する競合の性能への影響の調査，ハードウェアプリフェッチが働かないメモリレイアウトへのソフトウェアプリフェッチの適用の検討，などが挙げられる．

謝辞

本研究は，JST, CREST の支援を受けたものである．

参考文献

[1] Satish, N., Kim, C., Chhugani, J., Saito, H., Krishnaiyer, R., Smelyanskiy, M., Girkar, M. and Dubey, P.: Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications?, *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12)*, pp. 440–451 (2012).

[2] Sato, Y., Inoguchi, Y. and Nakamura, T.: Whole Program Data Dependence Profiling to Unveil Parallel Regions in the Dynamic Execution, *Proceedings of the 2012 IEEE International Symposium on Workload Characterization (IISWC '12)*, pp. 69–80 (2012).

[3] Sato, Y., Inoguchi, Y. and Nakamura, T.: Identifying Program Loop Nesting Structures during Execution of Machine Code, *IEICE TRANSACTIONS on Information and Systems*, Vol. E97-D, No. 9, pp. .2371–2385 (2014).

[4] Sato, Y., Sato, S. and Endo, T.: Exana: An Execution-driven Application Analysis Tool for Assisting Productive Performance Tuning, *Proceedings of the 2nd Workshop on Software Engineering for Parallel Systems (SEPS '15)* (2015).

[5] 佐藤真平，佐藤幸紀，遠藤敏夫：ルーフラインモデルによる性能幅推定とステンシル計算コードにおけるメモリレイアウト最適化による性能最大化，情報処理学会研究報告 2015-ARC-216， No. 32, pp. 1–6 (2015).

[6] Sato, S., Sato, Y. and Endo, T.: Investigating Potential Performance Benefits of Memory Layout Optimization Based on Roofline Model, *Proceedings of the 2nd Workshop on Software Engineering for Parallel Systems (SEPS '15)* (2015).

[7] Lee, J., Kim, H. and Vuduc, R.: When Prefetching Works, When It Doesn't, and Why, *ACM Transactions on Architecture and Code Optimization*, Vol. 9, No. 1, pp. 1–29 (online), DOI: 10.1145/2133382.2133384 (2012).

[8] Marin, G., McCurdy, C. and Vetter, J. S.: Diagnosis and Optimization of Application Prefetching Performance, *Proceedings of the 27th ACM International Conference on Supercomputing (ICS '13)*, pp. 303–312 (online), DOI: 10.1145/2464996.2465014 (2013).

[9] Himeno Benchmark: `http://accc.riken.jp/en/supercom/himenobmt/`.

[10] Hong, C., Bao, W., Cohen, A., Krishnamoorthy, S., Pouchet, L.-N., Rastello, F., Ramanujam, J. and Sadayappan, P.: Effective Padding of Multidimensional Arrays to Avoid Cache Conflict Misses, *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '16)*, New York, New York, USA, ACM Press, pp. 129–144 (2016).

[11] Ishizaka, K., Obata, M. and Kasahara, H.: Cache Optimization for Coarse Grain Task Parallel Processing Using Inter-Array Padding, *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC '03)*, pp. 64–76 (2003).

[12] A Guide to Vectorization with Intel C++ Compilers: https://software.intel.com/en-us/articles/a-guide-to-auto-vectorization-with-intel-c-compilers?language=en.