Vol.2015-ARC-216 No.32
2015/8/6

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

佐藤 真平[1,2,a)]　佐藤 幸紀[1,2,b)]　遠藤 敏夫[1,2,c)]

概要：大規模並列処理をおこなう場合においても CPU およびメモリに対するチューニングはアプリケーション全体の性能を決め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ルーフラインモデルに基づいて到達可能な性能幅を推定し，性能を最大化するための戦略と実際に施したチューニングの実例を報告する．性能最大化の戦略としてコードの最適化の余地を検証するために，CPU におけるハードウェア性能カウンタや我々が開発を進めるアプリケーション性能解析ツール Exana を用いてコード実行の性能解析と分析を行った．分析結果に基づき，メモリレイアウトに関する最適化をコードに適用した結果，最適化を行わないコードと比較して 3 倍程度の性能向上を達成した．

1. はじめに

エクサスケール時代に向けた高性能計算システムにおいて，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は，大規模並列化を行っても同程度しかシステム全体の性能を引き出せないということになる．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 1 ノードあたりの性能チューニングにおいては，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]．

1 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
2 JST CREST
a) sato.s.ae@m.titech.ac.jp
b) yukinori@el.gsic.titech.ac.jp
c) endo@is.titech.ac.jp

本稿では，ステンシル計算コードを対象に，ルーフラインモデルによる性能幅推定とメモリレイアウトの最適化による性能チューニングを行う．ステンシル計算は，流体シミュレーションなどの分野の重要カーネルである．一般に，シミュレーションする領域を格子で表し，それぞれの格子点の値を隣接する格子点の値を用いて計算し更新する処理を時間ステップとして繰り返し実行する．ステンシル計算は，並列性が高く，その性能はメモリバンド幅に大きく影響されることが知られている．

ルーフラインモデル [2] は，アプリケーションの潜在的なチューニングの可能性とボトルネック解析による性能限界を視覚的に示す性能モデルである．この性能モデルは，計算量と DRAM へのアクセス量を用いた抽象化で表現されており，メモリバンド幅に強く影響されるステンシル計算の性能解析ツールとして有用である．

我々が開発を進めるアプリケーション性能解析ツール Exana [3], [4] は，実行時にアプリケーションの解析を行うツールで，ハードウェアカウンタによる解析では得ることのできない詳細な性能解析が可能である．現在，メモリ階層性能シミュレータとしてキャッシュの性能を解析する機能の追加を進めている [5]．

本稿では，ステンシル計算のベンチマークとして知られる姫野ベンチマーク [6] を利用する．姫野ベンチマークを対象に，ハードウェアカウンタを用いたアプリケーションの性能解析および，ルーフラインモデルに基づく到達可能な性能幅を推定する．次に，Exana を用いたアプリケー

ⓒ 2015 Information Processing Society of Japan

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/s × 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
コンパイラ	icc 14.0.2
HW カウンタ取得ツール	likwid 3.1.3

ションの解析で，キャッシュにおける競合ミスを解析し，メモリレイアウトに関する最適化を行う．

2. ルーフラインモデルを用いた性能幅推定

2.1 ルーフラインモデル

ステンシル計算は，メモリアクセスの量に対して演算が少なく，その性能はメモリバンド幅に強く依存する．メモリバンド幅を考慮した性能モデルとしてルーフライモデル [2] が知られている．このモデルでは，演算性能とメモリバンド幅のどちらか一方がアプリケーションの性能を律速すると仮定し，メモリバンド幅，ピーク演算性能，演算強度（Operational Intensity）を用いてアプリケーション性能を見積もる．演算強度は，アプリケーションにおける浮動小数点演算量と DRAM アクセスのデータ量の比（Flops/Byte）である．仮定から，ルーフラインモデルによる演算性能は次の式で表される．

$$\text{演算性能} = min\{\text{ピーク演算性能}, \text{メモリバンド幅} \times \text{演算強度}\} \quad (1)$$

この式は，横軸に演算強度，縦軸に演算性能として図示すると，メモリバンド幅に律速され演算強度が高くなるにつれて演算性能が向上する線と CPU ピーク性能に律速され，演算強度にかかわらず演算性能が一定になる線で表現される．

ルーフラインモデルにおけるメモリバンド幅とピーク演算性能は，アプリケーションによって適切に設定する必要がある．例えばメモリバンド幅については，実行するスレッド数やメモリ構成によって変わる．ピーク演算性能についても，アプリケーションの性質や SIMD 命令使用の有無などによって変化する．

2.2 実験環境におけるルーフラインモデル

ここでは，本稿においてアプリケーションを実行する計算機環境についてまとめ，その環境におけるルーフラインモデルを示す．

表 1 に実験で使用する計算機環境をまとめる．この計算機は 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 スレッドとなっている．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性能は，AVX 命令を使用し，さらに加算と乗算を同時実行する場合で，CPU あたり $8 \times 2\ inst. \times 8\ cores \times 2.6\ GHz = 332.8\ GFlops/s$ となる．キャッシュは L1，L2 がプライベート，L3 が共有となっている．メモリは，8 GB のモジュールを各 CPU に 4 枚ずつ搭載し，4 チャネル すべてを利用している．コンパイラはインテルコンパイラ 14.0.2 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．実験において，アプリケーション実行時のメモリバンド幅などはハードウェアカウンタを用いて計測する．ハードウェアカウンタの取得には likwid 3.1.3 [7] を利用する．

likwid は，ハードウェアカウンタの取得のみならず，スレッド・アフィニティーの制御も行うことができるツールである．以降のすべての実験において，スレッドはコアに固定で割り当ててアプリケーションを実行している．

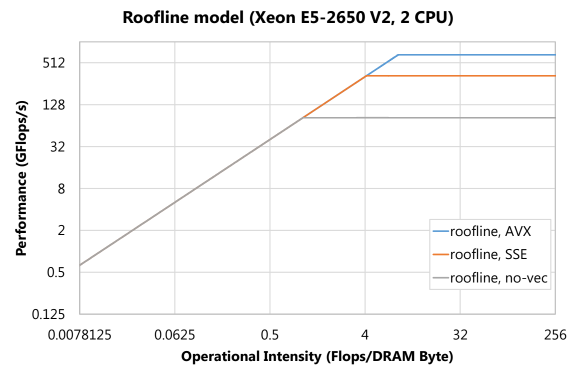

図 1　実験に使用する計算機環境におけるルーフラインモデル

図 1 に，実験で使用する計算機環境におけるルーフラインモデルを示す．モデルは，2 CPU（16 スレッド）実行時の AVX，SSE，SIMD 不使用（no-vec）の 3 つの場合について図示している．ピーク性能は，単精度浮動小数点演算で加算と乗算を同時実行したと仮定した場合の性能を用いている．式 1 より，図中の斜線の傾きはメモリバンド幅となる．メモリバンド幅は STREAM ベンチマーク [8] で計測した Triad の結果を用いている．

以降の議論において，1 スレッド実行時のルーフラインモデルを用いるが，ピーク性能とバンド幅は図 1 と同様に計測した結果を利用する．

2.3 姫野ベンチマークの性能幅推定

前述の計算機を用いて，姫野ベンチマークを実行しルーフラインモデルに基づき性能幅を推定する．姫野ベンチ

ⓒ 2015 Information Processing Society of Japan

表 2 姫野ベンチマークの 1 スレッドでの性能

	MFlops/s	MB/s
S Original	1,140.6	1,799.8
S Padding	1,634.2	2,182.7
L Original	497.8	1,172.8
L Padding	1,471.2	3,299.0

マークは C + OMP, dynamic allocate version を利用する．OpenMP による並列化が可能なバージョンではあるが，ここでは並列化のオーバーヘッドを除いた性能幅推定のために，1 スレッドにて計測を行う．また，SIMD 化のオーバーヘッドも除くために，コンパイル時に -no-vec オプションを使用し，SIMD 化を抑制する．

実験では，姫野ベンチマークのサイズ S およびサイズ L を用いる．サイズ S は使用する計算機において確保したメモリがすべてキャッシュにのるサイズで，サイズ L は確保したメモリがキャッシュにのらないサイズとなっている．ルーフラインモデルでは計算強度として DRAM アクセスの量を用いるため，データがキャッシュにのる場合は DRAM アクセスが発生せず，メモリバンド幅に律速されない性能を示すと推測される．逆に，データがキャッシュにのらないサイズでは，メモリバンド幅に律速される性能を示すと推測できる．以降の実験においても同様の理由からサイズ S とサイズ L の両方の性能を計測する．

性能幅推定にあたって，ソースコードにパディングによる最適化を実施する．姫野ベンチマークの static allocate version では，宣言されている配列の各次元の要素数が 1 要素ずつ余分に確保されており，パディングによる最適化が行われている．パディングにより，キャッシュにおける競合ミスを軽減し，性能向上が期待できる．しかし，dynamic allocate version では，パディングによるを行っていない．そのため，パディングを実施し，メモリ確保時に static allocate version と同様に各次元の要素を 1 つ余分に確保する．

姫野ベンチマークの性能幅推定では，ダウンロードしたままのコード（Original）とパディングを実施したコード（Padding）の 2 つについて性能を計測する．実行時のメモリバンド幅は likwid を用いて計測し，アプリケーションの性能は姫野ベンチマークが出力するスコアを用いる．

表 2 にサイズ S およびサイズ L の 1 スレッドの時の性能とメモリバンド幅の計測結果を示す．いずれの場合も，Padding により性能向上が見られる．パディングの効果がサイズ S よりサイズ L の方が大きい理由としては，サイズ S ではほぼすべてのデータがキャッシュにのっているため L1 でキャッシュミスが発生しても L3 までのアクセスで収まるのに対して，サイズ L では キャッシュミスにより DRAM アクセスが発生してしまい，キャッシュミスのペナルティが大きいことが考えられる．

次に，それぞれのサイズについてルーフラインモデルに

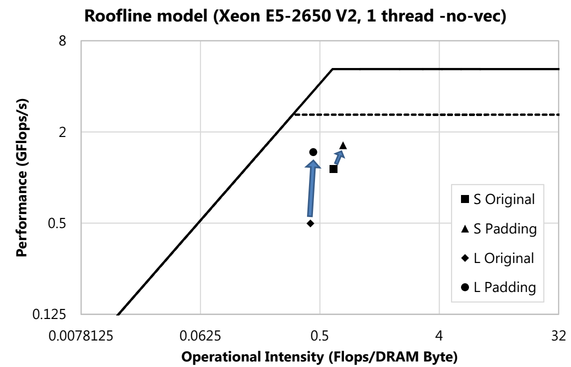

図 2 姫野ベンチマークの 1 スレッドでのルーフラインモデル

プロットし性能幅を推定する．**図 2** にサイズ S およびサイズ L の 1 スレッド実行時の性能をルーフライモデルで示す．ルーフラインの実線は加算と乗算を同時実行した場合のピーク性能，破線は同時実行をしない場合のピーク性能を示している．

図から，サイズ S とサイズ L のいずれの場合でもパディングの実施によりキャッシュミス率が改善し，Padding の演算強度が Original に対して右方向に移動していることが確認できる．

バイナリの解析から，姫野ベンチマークのカーネル部には 34 の浮動小数点演算があり，そのうち 13 が乗算であった．13 の乗算のうち加算と同時実行できる可能性のある乗算は 9 であった．ここから，姫野ベンチマークはメモリバンド幅に律速されない時は加算と乗算を同時実行しない場合のピーク性能より高い性能を達成することが期待できる．図 2 で，パディングを実施したコードの性能は加算と乗算を同時実行しない場合のピーク性能の 50 % 前後であり，高速化の余地が残されていると考えられる．

3. Exana による性能解析とメモリレイアウトの最適化

3.1 メモリ階層性能シミュレータによる競合ミスの計測

我々が開発を進める，アプリケーション性能解析ツール Exana [3], [4] を用いて姫野ベンチマークの解析を行う．Exana は，アプリケーションの実行時にループ構造や関数呼び出しの情報などを解析するツールである．現在，メモリ階層性能シミュレータとしてメモリアクセスに関する性能を解析する機能を開発している [5]．ここでは，性能解析のために追加されたキャッシュシミュレータを用いてキャッシュ競合ミスを計測する．

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．ライトスルー方式を採用し，置き換えアルゴリズムは LRU を用いている．簡単のために，インクルーシブポリシーで実装されており，またプリフェッチは未実装となっている．

ⓒ 2015 Information Processing Society of Japan

表 3 姫野ベンチマーク サイズ S の Exana によるキャッシュ競合ミスの計測

	Original	Padding
Cache miss		
L1	37.15 %	2.68 %
L2	7.06 %	98.96 %
L3	80.11 %	82.26 %
Conflict miss		
L1	92.85 %	1.04 %
L2	11.06 %	11.21 %
L3	4.75 %	8.05 %

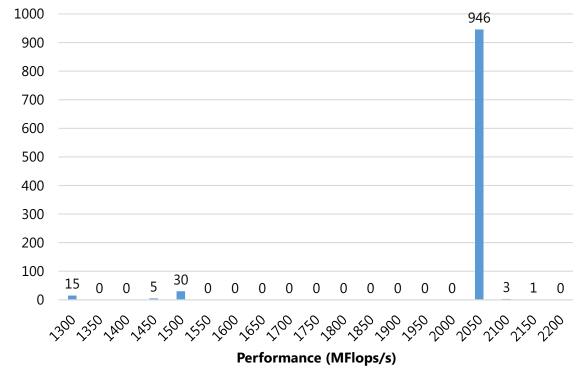

図 3 パディングで挿入するスペースを乱数により決定した姫野ベンチマーク サイズ S の性能の分布

キャッシュにおける競合ミスは，キャッシュミスの 1 つで複数のキャッシュラインが同じセットを使用し，ウェイが不足するこで発生するミスである．競合ミスが多いアプリケーションでは，確保しているメモリ領域に対してパディングを実施することで使用するセットをずらすことができ，競合ミスを減らすことが期待できる．

シミュレータでは，次の手法 [9] で競合ミスを判定している．

(1) キャッシュのセットごとに 32 エントリの FIFO を用意し，アクセスしたキャッシュのタグを保持
(2) LRU にてラインの置き換えが発生したときに，追い出されるラインのタグが FIFO に保持されていれば競合ミスと判定

表 3 に，前節で用いた姫野ベンチマークの Original と Padding について Exana のメモリ階層性能シミュレータを用いてキャッシュ競合ミスを計測した結果を示す．ベンチマークのサイズは S である．

L1 のキャッシュミス率と競合ミス率に注目すると，Padding は Original に対して大幅に競合ミスが減少し，キャッシュミス率について改善されていることがわかる．

姫野ベンチマークでは 14 個の配列が宣言されており，計算カーネルですべての配列を連続に参照している．8 ウェイのキャッシュにおいて，連続にアクセスする 14 個の配列を扱うと単純には競合ミスが発生すると考えられる．しかし，適切にパディングを実施することで競合ミスが軽減し高速化が可能であることがわかる．

3.2 姫野ベンチマークのメモリレイアウトの最適化

ここでは，姫野ベンチマークにパディングを実施することでさらなる高速化を目指す．

前節で実施したパディングは，配列の各次元の要素を 1 つ余分に確保するという方法であった．しかし，この方法は SIMD 化のためのアライメントを考慮すると適切とは言えない．そこで，valloc を用いてすべての配列をページ境界で確保し，その先頭にスペースを挿入しセットをずらすという方法でパディングを実施する．

姫野ベンチマークでは 14 個の配列が使用されているが，ベースとして使用しているコードでは，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．確保したメモリ領域の先頭にスペースを挿入する方法によるパディングでは，それらの配列も個別の 3 次元配列として扱いたい．そこで，コードを変更し 14 個の配列すべてが 3 次元の配列としてメモリを確保するようにする．このコード変更で，すべての配列をページ境界に配置した場合の性能は 1,180.0 MFlops/s で，前節で使用した Original の性能 1,140.6 MFlops/s から大きな変化はない．

実験で用いる計算機の CPU の L1 データキャッシュはサイズ 32 KB，ウェイ数 8，ラインサイズ 64 B なのでセット数は 64 となる．ページ境界に配置されたデータは，L1 キャッシュに置かれるときにはセット 0 から格納される．パディングで挿入するスペースの大きさをラインサイズ × N（N は 0 から 63）とし，配列の先頭が格納されるセットをずらすことで競合ミスの軽減をねらう．

図 3 に，サイズ S の姫野ベンチマークで 14 個の配列に挿入するパディングの大きさを乱数で決定し，1,000 パターン実行したときの性能の分布を示す．横軸は性能で単位は MFlops/s である．例えば 1,300 MFlops/s の場合，性能が 1,300 MFlops/s 以上 1,350 MFlops/s 未満のパディング挿入パターンが 15 個あることを意味する．

図から，95 %近くのパターンの性能が 2,050 MFlops/s 以上 2,200 MFlops/s 未満の範囲に収まることがわかる．前節において計測したサイズ S の Padding は 1,634.2 MFlops/s であった．パディングのために実装の変更を行ってはいるが，さらなる高速化を達成するメモリレイアウトがあることがわかる．

表 4 に，サイズ S の実験において最も高い性能を示した挿入パターン（**Max**）と最も低い性能を示したパターン（**Min**）について，Exana による競合ミスの計測結果を示す．Min については，すべての配列をページ境界に配置した場合（挿入するスペースがすべて 0 の場合）が最も低い性能となったため，実験した 1,000 パターンには現れていないパターンではあるが，このパターンを Min として

ⓒ 2015 Information Processing Society of Japan

表 4 パディングで挿入するスペースを乱数により決定した姫野ベンチマーク サイズ S の性能

	Min	Max
MFlops/s	1,180.0	2,173.9
Cache miss		
L1	39.78 %	2.43 %
L2	6.10 %	100.00 %
L3	87.47 %	87.47 %
Conflict miss		
L1	93.86 %	0.00 %
L2	12.44 %	12.44 %
L3	0.48 %	0.48 %

いる．

L1 キャッシュのミス率に注目すると，Max では競合ミス率が 0 % にまで減らすことができ，キャッシュミス率 2.43 % となる．前節の Padding では L1 の競合ミス率が 1.04 %，キャッシュミス率が 2.43 % であり，Max はさらなるミス率の改善を達成していることがわかる．

同様の実験をサイズ L でも行ったが，1,000 パターンすべての性能が 500 MFlops/s 以上 550 MFlops/s 未満の範囲に収まる結果となった．これは，前節にける Original の性能 497.8 MFlops/s とほぼ同等である．14 個の配列へのパディングの挿入パターンは非常に多くの場合があり，サイズ L ではパディングにより性能向上する場合が少なく，前節における Padding の性能 1,471.2 MFlops/s のような性能となるパターンが 1,000 パターンの実験では現れなかったと考えられる．

また，ステンシル計算の場合は隣接する格子を参照しながら計算をすすめる．隣接する格子もまた連続で参照されており，連続した参照それぞれを別の参照系列と見なすと，姫野ベンチマークでは 22 個の参照系列があると見なすことができる．同じ配列の別の参照系列どうしで競合ミスが発生している場合，確保したメモリ領域の先頭にスペースを挿入する方法によるパディングでは競合ミスを減らすことができない．サイズ L では，同じ配列の別の参照系列によって競合ミスが発生していることが考えられる．

4. 評価

ここまでは，1 スレッドかつ SIMD 化をせずに姫野ベンチマークの解析を行ってきた．ここでは，これまでの最適化を実施したコードについて並列化を行い評価する．SIMD 化はこれまでと同様に行っていない．

評価対象は以下の通りである．

- Original：ダウロードした dynamic allocate version のコード
- Padding：static allocate version と同様のパディングを適用したコード
- Min：すべての配列をページ境界に配置したコード
- Max：1,000 通りの実験で最も高い性能を示したパディングを適用したコード

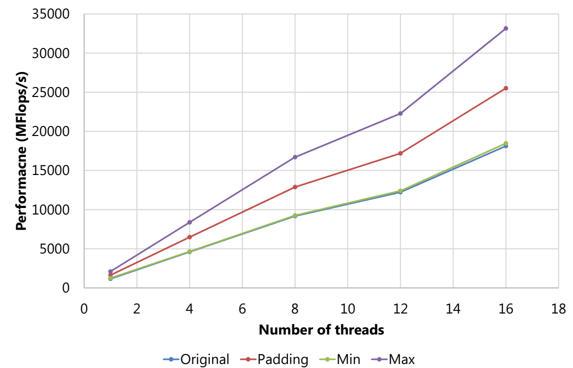

図 4 姫野ベンチマーク サイズ S でスレッド数を変化させたときの性能

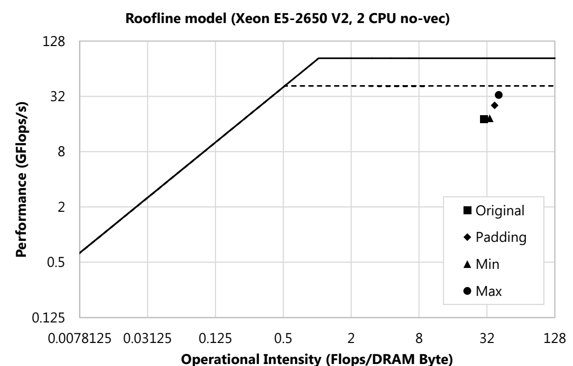

図 5 姫野ベンチマーク サイズ S で 16 スレッド実行時のルーフラインモデル

実験に使用する計算機は NUMA 型のマシンのため，すべてのコードでファーストタッチを実施している．また，並列実行時のスレッドは 2 つの CPU に均等に分配し，コアに固定して割り当てている．

図 4 に姫野ベンチマークのサイズ S でスレッド数を変化させたときの性能を示す．すべてのコードで 16 スレッド実行時の性能が最も高くなっている．Original と Min が同等の性能を示している．1 スレッドの場合で最も高い性能を示した Max は実行スレッド数を変えても最も高い性能を示しており，16 スレッド実行の場合で Original に対して約 1.8 倍の性能を達成する．

図 5 に，サイズ S で実行スレッド数 16 の性能をルーフラインモデルで示す．ルーフラインの実線は浮動小数点演算の加算と乗算を同時実行した場合のピーク性能，破線は同時実行しない場合のピーク性能である．サイズ S のデータはすべてキャッシュにのるサイズのため，DRAM アクセスが少なく演算強度（Operational Intensity）は大きな値になる．最も高い性能を示している Max は，浮動小数点演算を同時実行しない場合のピーク性能の約 80 % の性能となる．

図 6 に，サイズ L でスレッド数を変化させたときの性能を示す．サイズ L の乱数を使用したパディングの挿入に

ⓒ 2015 Information Processing Society of Japan

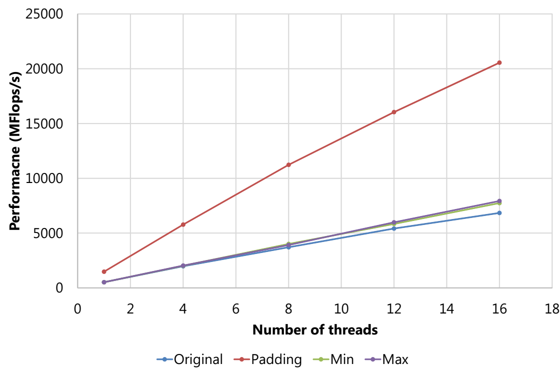

図 6 姫野ベンチマーク サイズ L でスレッド数を変化させたときの性能

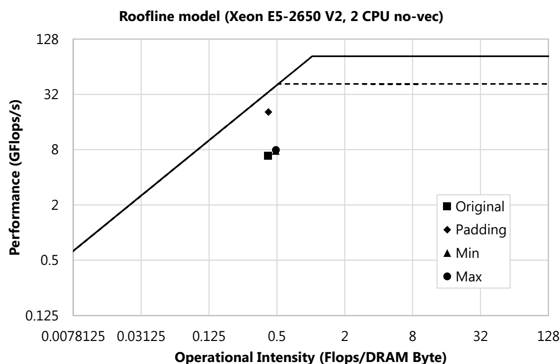

図 7 姫野ベンチマーク サイズ L で 16 スレッド実行時のルーフラインモデル

おいて，1,000 通りの実験ではサイズ S の場合のような高い性能を示すパターンは現れなかった．ここでは，実験した 1,000 パターンの中で最も性能が高いパターンを Max としている．

すべてのコードで 16 スレッド実行時が最も性能が高くなる．Original，Min，Max に大きな性能差はみられない．Padding が最も高い性能を示しており，16 スレッドの場合で Original に対して約 3 倍の性能を達成する．

図 7 にサイズ L の 16 スレッド実行時の性能をルーフラインモデルで示す．サイズ L は DRAM アクセスが多く演算強度は小さな値となる．最も高い性能を示している Padding は，その演算強度の時のピーク性能の約 60 % の性能となる．

5. まとめ

エクサスケール時代に向けた高性能計算システムにおいて，高速計算を実現するために大規模並列処理を行う場合においても，1 ノードあたりの性能チューニングはアプリケーション全体の性能を決め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，ルーフラインモデルに基づいた性能幅の推定を行い，姫野ベンチマークに高速化の余地があることを確認した．次に，我々が開発を進めるアプリケーション性能解析ツール Exana のメモリ階層性能シミュレータを用いた解析を行った．解析から，姫野ベンチマークが L1 キャッシュにおける競合ミスを軽減することで高速化が可能であることを示した．L1 キャッシュにおける競合ミスを軽減するためのメモリレイアウトを網羅的に調査し，最も性能が高いレイアウトにおいて大幅に競合ミスを軽減できることがわかった．評価では，並列化を行った場合においても競合ミスの少ないメモリレイアウトのコードが高い性能を示すことを確認した．ルーフラインモデルに基づく性能の推定では，競合が少ないメモリレイアウトの姫野ベンチマークのコードは，サイズ S の場合でピーク性能の約 80 %，サイズ L の場合でピーク性能の約 60 % の性能となった．競合ミスが多いメモリレイアウトのコードに対しては，サイズ S で約 1.8 倍，サイズ L で約 3 倍の高速化を達成した．

今後の課題としては，サイズ L においてより競合ミスが少ないメモリレイアウトを調査すること，メモリレイアウトについて SIMD 化も考慮した最適化を検討すること，メモリレイアウトの最適化についてプログラマへのフィードバック方法を検討することなどが挙げられる．

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?, *In Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12)*, pp. 440–451 (2012).
[2] Williams, S., Waterman, A. and Patterson, D.: Roofline: An Insightful Visual Performance Model for Multicore Architectures, *Communications of the ACM*, Vol. 52, No. 4, p. 65 (2009).
[3] Sato, Y., Inoguchi, Y. and Nakamura, T.: Whole Program Data Dependence Profiling to Unveil Parallel Regions in the Dynamic Execution, *In Proceedings of the 2012 IEEE International Symposium on Workload Characterization (IISWC '12)*, pp. 69–80 (2012).
[4] 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).
[5] 佐藤幸紀，佐藤真平：メモリ階層性能シミュレータを用いた CPU 単体性能チューニング，ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集（HPCS 2015），pp. 100–100 (2015).
[6] Himeno Benchmark: http://accc.riken.jp/2444.htm.
[7] Likwid: https://github.com/rrze-likwid/likwid.
[8] STREAM Benchmark: https://www.cs.virginia.edu/stream/.
[9] Jouppi, N. P.: Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers, *In Proceedings of the 17th International Symposium on Computer Architecture (ISCA '90)*, pp. 364–373 (1990).

ⓒ 2015 Information Processing Society of Japan