メモリアクセス最適化を適用した汎用プロセッサとCellの性能比較 —Cell Speed Challengeを題材にして—

原 健太朗[†] 塩谷亮太[†] 田浦健次朗[†]

Cellが汎用プロセッサに対して圧倒的な高性能を記録したという報告が多い．しかし従来の性能比較は，SIMDやDMAダブルバッファリングなどでCellを高速化する一方，比較対象の汎用プロセッサに対してはどの程度の高速化手法を導入したのか明確でない場合が多かった．そこで本実験では，SACSIS 2007で行われたCell Speed Challenge 2007の決勝記録にXeon E5335 8コアマシンで挑戦し，汎用プロセッサに対してSIMDやメモリアクセス最適化などの手法を駆使した上で，Cellとの性能比較を行った．その結果，10個中5個のデータセットで処理時間が25〜39% Cellを上回り，最も遅いものでもCellの2.7倍程度という結果を得た．また，Xeonの高速化に当たって作成したメモリアクセス最適化ツールであるTulipに関して，その最適化の効果を8種類の汎用プロセッサ上で検証した結果，最適化前と比較して18〜48%の速度向上を確認した．

Comparing Performance of General Purpose Processors with Memory Access Optimizations and the Cell —A Case Study with Cell Speed Challenge—

Kentarou Hara,[†] Ryouta Shioya[†] and Kenjirou Taura[†]

There are many reports that the Cell processor recorded overwhelming high performances compared with general purpose processors. However, while they achieved impressive performances on the Cell by SIMD and DMA double buffering, many of them are not clear about how much speedup techniques were introduced for the general purpose processors. Therefore, we challenged using Xeon E5335 8-core machine to the final result of the Cell Speed Challenge 2007, which was held in SACSIS 2007, and we applied performance enhancement techniques including SIMD and memory access optimizations on the Xeon. Then we compared the Xeon's performances with the Cell's. As a result, the Xeon outperformed the Cell by 25-39% on 5 out of 10 datasets, and even in the slowest dataset the Xeon was slower than the Cell only by a factor of 2.7. In addition, we inspected with 8 kinds of general purpose processors the effect of the optimizations of Tulip, which is an automatic memory access optimizer we developed for general purpose processors, and confirmed speedup of 18-48%.

1. はじめに

近年，汎用プロセッサに対して，特定の処理に特化したプロセッサを活用するテクノロジーが高い注目を浴びている．たとえば，演算性能に優れたGPUを汎用CPUに統合することでチップ処理能力の総合的な向上を図る技術[1)]がある．また，PlayStation3に搭載されたことで話題を集めているCell Broadband Engine（以下Cell）もその一例だと言えよう．

Cellは，強力な演算性能と広帯域幅を実現した次世代型のアーキテクチャで，数値解析やマルチメディア処理における将来性が期待されている．Cellが汎用プロセッサに対して圧倒的な高性能を記録したという報告も多い[2)~6)]．しかし，従来の研究はSIMDやDMAダブルバッファリングなどでCellを高速化する一方，比較対象の汎用プロセッサに対してはどの程度の高速化手法を導入したのか明確でない場合が多かった．

汎用プロセッサの強みは，一般への普及度の高さ，汎用性，プログラミングの容易さにある．現在の汎用プロセッサは急速に進化しており，マルチコアが主流となり，SIMDも利用可能である．また，プリフェッチやストリーミングストアなど，キャッシュコヒーレンシに基づくアーキテクチャにおいても高効率な通信を実現するための手段が提供され，メモリバンド幅の向上もめざましい．したがって，Cellを高速化するのと

† 東京大学
The University of Tokyo

同等のプログラミングエフォートを投入すれば，Cellと同等かそれ以上の結果が出たとしてもなんら不思議ではない．そこで本実験では，SACSIS 2007で開催されたCell Speed Challenge 2007（以下Cellチャレンジ）[7]の決勝記録にXeon E5335 8コアマシンで挑戦し，Cellとの性能比較を行うとともに，メモリアクセス最適化を中心とした汎用プロセッサの高速化手法について分析した．

本稿の構成は以下の通りである：

第2節 関連研究について述べる．

第3節 Cellチャレンジを題材にして，Cellと汎用マルチプロセッサを性能比較する．Xeon E5335 8コアマシンに対してSIMDやストリーミングストアなどの高速化手法を適切に駆使した結果，10個中5個のデータセットで処理時間が25〜39% Cellを上回り，最も遅いものでもCellの2.7倍程度という結果を得た．

第4節 第3節においてXeonを高速化するに当たって作成した，汎用プロセッサ上のメモリアクセス最適化ツールであるTulipを紹介する．8種類の汎用プロセッサ上でTulipによる最適化の効果を検証した結果，最適化前と比較して18〜48%の速度向上を確認した．

第5節 結論を述べる．

2. 関連研究

Cellと汎用プロセッサとの性能比較に関しては以下のような関連研究がある：

- Cellチャレンジ報告 4)〜6)によると，ソーティング[4]，合成ホログラムの計算[5]，神経回路網の計算[6]において，Cellが，Athlon XP 2800+，1スレッドのOpteron（1.8 GHz），1スレッドのCore2Quadなどに対して9〜29倍の性能を発揮した．
- 研究 2)では，SpMVやFFTなどの数値科学演算を題材に，Cellを，スーパースカラ型のOpteron 248やVLIW型のItanium2と性能比較している．その結果，CellがOpteronの4〜37倍の高性能を記録した．
- 研究 3)では，Cell上でのレイトレーシングの高速化を論じ，その成果を，x86システム上において最速とされるレイトレーシングアルゴリズム[8]をOpteron（2.4 GHz）上で実行した結果と比較している．その結果，CellがOpteronの5〜15倍の性能を発揮した．

これらの研究は主にCellの高速化を論じており，比較対象の汎用プロセッサが現時点の最新プロセッサでなかったり，汎用プロセッサに対してどの程度の高速化手法を施した上での比較かが明確でない．これに対して本実験では，Xeon E5335 8コアマシンに高速化のための実装を駆使した上で，Cellとの性能比較を行う．

また，ターゲットマシン上での自動最適化に関しては以下のような関連研究がある：

- ATLAS 9)は，線形代数ライブラリBLASの各種パラメータを，L1キャッシュ関連や浮動小数点数演算などを対象にターゲットマシン上で自動最適化する．研究 10)によれば，Origin 3400を16個用いたGEMMにおいてATLAS BLASが通常のBLASに比較して38倍の性能を発揮した．
- 研究 11)では，レイトレーシングのプリミティブ演算に関して1スレッドのOpteron（2.4 GHz，2コア），1スレッドのXeon（3.2 GHz，2コア），Pentium4（3.0 GHz），Core Duo（1.83 GHz）で遺伝的アルゴリズムを用いて各プロセッサに最適なコードを生成した結果，最大2倍程度の高速化を実現した．

これらの研究が主に数値演算の最適化を目標とするのに対し，本実験で作成したTulipはL2キャッシュ（以下単にキャッシュ）とメモリ間のデータ転送の最適化を目標とする．

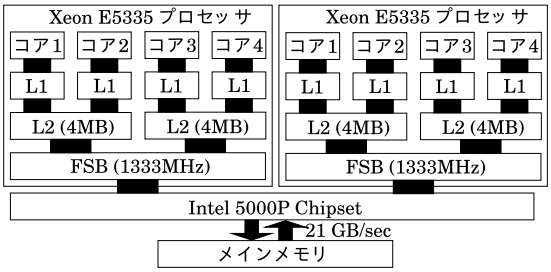


図1 本実験で使用した8コアマシンのアーキテクチャ

3. 汎用マルチプロセッサでCellチャレンジに挑む

3.1 汎用マルチプロセッサとCell

3.1.1 汎用マルチプロセッサのアーキテクチャ

本実験では，汎用マルチプロセッサとしてIntel Xeon E5335（2.0 GHz，4コア）を2基搭載した8コアマシンを使用した．そのアーキテクチャを**図1**に，実験環境を**表1**に示す．汎用マルチプロセッサは，アウトオブオーダ実行，動的分岐予測，ハードウェアプリフェッチ，キャッシュ管理などの高度なハードウェア機構を備えており，容易なプログラミングで効率的なプログラムを開発できる環境を提供する．その反面，単一バスに複数のプロセッサを接続することによるバスの輻輳や，スヌープ方式によるキャッシュコヒーレンシ保持に伴うフォルスシェアリングなどが原因となって，スケーラビリティは良くない．また，暗黙的なハード

表 1 本実験で使用した 8 コアマシンの仕様

製品名	JCS Type 1U-XEK/XEW
プロセッサ	Intel Xeon E5335（2.0GHz，4 コア）×2
チップセット	Intel 5000P Chipset
メモリ	2GB×8
OS，カーネル	GNU/Linux，2.6.18-6-amd64
コンパイラ	g++ 4.2.2

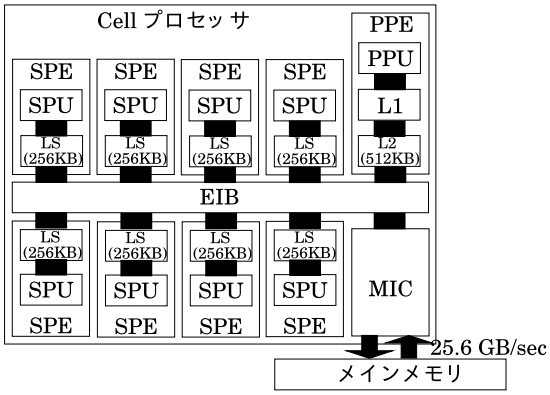


図 2 Cell のアーキテクチャ

ウェア機構の動作の詳細をプログラムから制御できないため，性能を引き出しにくいという欠点を持つ．

3.1.2 Cell のアーキテクチャ

Cell は，PPE（PowerPC Processor Element）1 個と SPE（Synergistic Processor Element）8 個☆を**図 2** のようにリング状に接続したヘテロジニアスマルチコアプロセッサである．各プロセッサは強力な SIMD 命令セットを備えており，複雑なスケジューリング機構を実装しない．また各 SPE に専用の LS（Local Store）は，明示的な DMA 転送によるメインメモリとの通信しか行わないため，キャッシュコヒーレンシ保持の機能を必要としない．このように Cell は予測不可能性を回避したアーキテクチャを採用しており，その動作を明示的に制御可能なため，挑戦的なアーキテクチャとして注目されている．その一方で，特別なプログラミング技術を要したり，複雑な条件分岐などは汎用プロセッサよりも劣るなどの弱点を持つ．

Xeon E5335 と Cell のスペックを**表 2** に比較した．

3.2 Cell チャレンジの規定課題

Cell チャレンジの規定課題はソーティングである．規定課題のルール[7] の概要を以下に記す：

- **表 3** に示す 10 個のデータセットが与えられる．
- 各データセットは，成分が `float` 型の `m` 次元ベクトル `n` 個から構成され，これが要素数 `(m+1)*n*2` の配列 `data[]` に格納されて与えられる．`i` 番目のベクトルは，`data[(m+1)*i+1...(m+1)*(i+1)-1]` に格納されており，その他の要素は `0` に初期化されている．
- 各ベクトルに対してキーを計算し，このキーの昇順にベクトルをソートする．キーとしては，ベクトル成分の二乗和もしくは最大成分が指定される．
- ベクトルをソートした結果を `data[0...(m+1)*n-1]` に格納してソーティングは完了である．`data[(m+1)*i]` には `i` 番目のベクトルのキーを書き込む．
- 利用可能なメインメモリの容量は，配列 `data[]` と 16 キロバイトに制限される．

☆ ただし，Cell チャレンジは PPE 1 個と SPE 7 個を使用して行われた．

表 2 Xeon E5335 と Cell のスペックの比較

	Xeon E5335	Cell
コア数	8	PPE:1，SPE:8
動作周波数	2.0 GHz	3.2 GHz
キャッシュ	L2:4MB × 4	LS:256KB × 8
浮動小数点数演算性能	128 Gflops	204.8 Gflops
最大メモリバンド幅	21 GB/sec	25.6 GB/sec

表 3 10 個のデータセットにおける `m`，`n`，キーの種類

No.	m	n	キーの種類
1	3	5898240	最大成分
2	7	2621440	二乗和
3	1023	20480	二乗和
4	2047	10240	最大成分
5	255	102400	最大成分
6	1023	20480	二乗和
7	15	1572864	最大成分
8	7	2621440	二乗和
9	7	2621440	最大成分
10	31	819200	最大成分

本実験も上記のルールに準拠して行った．この規定課題は，キーを計算する演算処理とデータを移動するデータ転送処理の 2 つの局面を含んでおり，高速化のためには演算処理と転送処理の両方の効率化が欠かせない．

3.3 並列ラディックスソート

本実験では並列ラディックスソートを行った．ラディックスソートは，固定長のキーを持つ要素に対して安定で高速なソーティングを行う，時間計算量 $O(n)$ のアルゴリズムである[14]．また，効率的な並列化が可能であり，シーケンシャルアクセスが中心となるためキャッシュにやさしいなどの優れた特徴を持つ．

本実験で使用したアルゴリズムは，Cell チャレンジ 2 位チームの報告[4] に依った．概要を以下に述べる：

キー計算ステージ 領域の前半部からベクトルを読んでキーを計算し，インデックス（ベクトルの位置とキー

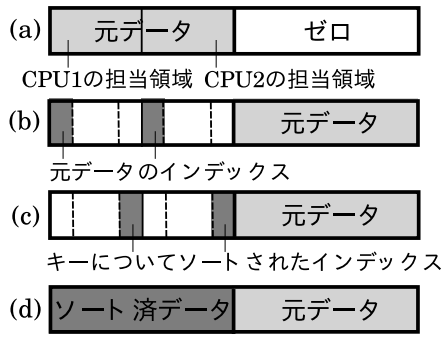


図 3 並列ラディックスソートのアルゴリズムの流れ（CPU 2 個で行う場合）

のペア）を作成する．同時に，読み出したベクトルを領域の後半部へ書き込む（**図 3**（b））．

並び替えステージ 領域の前半部を利用してインデックスをラディックスソートする（図 3（c））．

データ移動ステージ ソート済みのインデックスを読み，そのインデックスに対応するベクトルを領域の後半部から読み出して領域の前半部に書き込む（図 3（d））．

並び替えステージでは，32 ビット単精度浮動小数点数をキーとしたラディックスソートを行うが，本実験環境では，n が 102400 以上の場合には 8 ビット-8 ビット-8 ビット-8 ビットの 4 段階ソートが最速であり，n が 102400 未満の場合には 11 ビット-11 ビット-10 ビットの 3 段階ソートが最速であることがわかった☆．

3.4 汎用マルチプロセッサにおける高速化手法

本実験では，ループアンローリングなどのコーディングレベルの最適化や各種演算部分での細かいアルゴリズムの工夫などの基本的な高速化手法を施した上で，さらなる高速化を実現するため，SIMD，ストリーミングストア，ソフトウェアプリフェッチ，データアラインメントに関して，汎用マルチプロセッサ上での効率的な実装を分析した．以下ではこれら 4 つの手法について論じる．

3.4.1 SIMD

SIMD は，1 命令で複数のデータを並列に演算処理する命令レベルの並列化手法であり，大量のデータに対して似たような処理を施すマルチメディア処理などで効力を発揮する．Xeon は SSE（Streaming SIMD Extentions）やそれを拡張した SSE2，SSE3 を実装している．

図 4 に，キー計算ステージとデータ移動ステージ

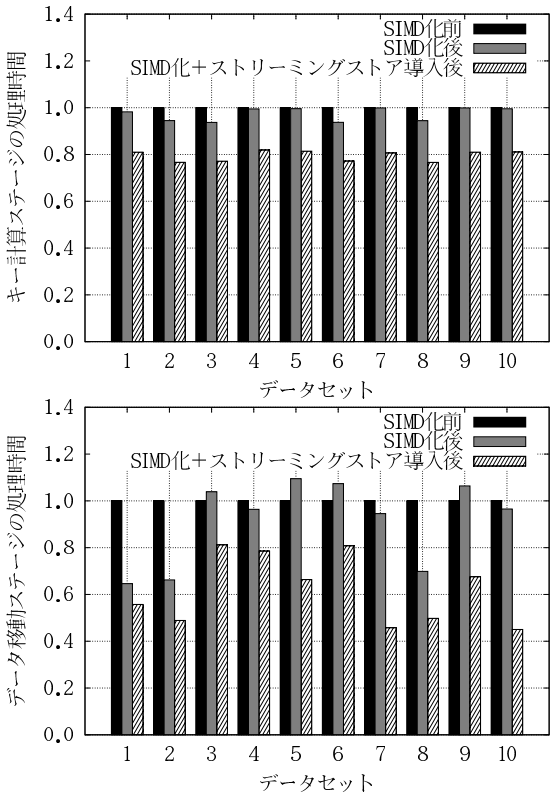


図 4 SIMD とストリーミングストアの効果（上図：キー計算ステージ，下図：データ移動ステージ．各データセットに関して，左の棒グラフが SIMD 化前の処理時間，中央の棒グラフが SIMD 化後の処理時間，右の棒グラフが SIMD 化に加えてストリーミングストアを導入した場合の処理時間．これらの処理時間は SIMD 化前の処理時間を 1 に規格化した場合の数値である．）

に関して，SIMD の効果とストリーミングストア（後述）の効果を示す☆☆．また，**図 5** には各ステージの処理時間比率を示す．図 4 における SIMD の効果を見ると，データ移動ステージで 35%高速化しているデータセットがあるものの，SIMD の効果がほとんど現れていないものが多い．これは処理がバンド幅に律速されているためで，ただ単に演算を高速化するだけでは十分なパフォーマンスを期待できない．このような場合には，プリフェッチによって現在の演算と次に使うデータの読み込みをオーバーラップさせ，ストリーミングストアによって現在の演算と前回のデータの書き込みをオーバーラップさせることによって，並行実行性を高め，バンド幅を向上させることが不可欠である[16]．

☆ ただし，使用可能メモリ量の制限により，データセット 1 では 4 段階ソートを行った．

☆☆ 原則として，本稿に掲載するグラフやデータは，同一条件下で 50 回実行させて，速い方から 10 個と遅い方から 10 個を除いた 30 個のデータの平均値に基づいている．

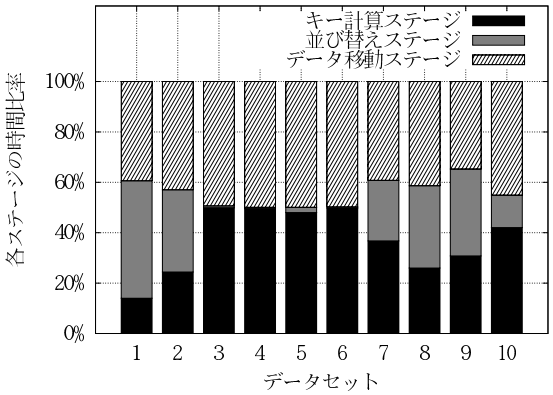


図 5 SIMD とストリーミングストア導入後のコードに関する，キー計算，並び替え，データ移動の各ステージに要する処理時間の割合

3.4.2 ストリーミングストア

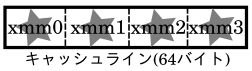
本実験のソーティングや各種ストリーミング処理のように，一度書き込んだデータに対して近い将来に再び書き込みを行う可能性がない処理においてはストリーミングストアが有効な場合がある．ストリーミングストアでは，ライトコンバイニングバッファを経由したノンテンポラルなライトをキャッシュを介さずに 1 回のバストランザクションで実現できる．したがって，キャッシュフィルとライトバックの 2 回のバストランザクションが必要になる通常ストアと比較すると，バスの輻輳を抑えられる，キャッシュの汚染を防げる，並行実行性が高まるなどの利点がある．

Xeon に実装されている SSE2 では，キャッシュをバイパスして 128 ビットレジスタからメインメモリに直接データを書き込む `movntps` 命令などが用意されている．ただし Xeon における `movntps` 命令は，ライトコンバイニングバッファを介した書き込みを行うため，キャッシュライン単位で連続的に書き込みを行う場合のみ効果を発揮し，キャッシュラインの一部のみ書き込みを行う場合に使用すると著しい速度低下を招くという特性がある[17]．具体的には，1 スレッドで実行した場合，**図 6**（a）に示すコードは 0.68 sec で終了するが，図 6（b）に示すコードは 14.9 sec も要する．したがって，適度なアンローリングを行った上で命令の順序を交換するなどして常にキャッシュライン単位で `movntps` 命令が発行されるよう工夫することが重要となる．なお，キャッシュライン単位で命令を発行できない場合には，キャッシュを利用する通常ストアを使用する方が高速である．

ストリーミングストアによる効果を図 4 で確認すると，キー計算ステージで 19〜23%，データ移動ステージでは 19〜55%高速化しており，効果は十分現れていると言える．

(a)

```
char buf[SZ],*p;
for(p=buf; p<buf+SZ; p+=64)
{
    movntps p , xmm0
    movntps p+16 , xmm1
    movntps p+32 , xmm2
    movntps p+48 , xmm3
}
```



キャッシュライン(64バイト)

(b)

```
char buf[SZ],*p;
for(p=buf; p<buf+SZ; p+=64)
{
    movntps p , xmm0
    movntps p+16 , xmm1
    movntps p+32 , xmm2
}
```

キャッシュライン(64バイト)

図 6 movntps 命令の特性（(a) キャッシュライン単位で発行する場合，(b) キャッシュラインの一部に対して発行する場合）

3.4.3 プリフェッチ

Xeon には，ハードウェアプリフェッチとソフトウェアプリフェッチの 2 種類がある．本実験のソーティングでは，データへのアクセスパターンを把握できるので，不確定要素の大きいハードウェアプリフェッチは無効化し，データを格納するキャッシュ階層やスケジューリング距離を制御可能なソフトウェアプリフェッチを明示的に発行した．その結果，1 スレッドではプリフェッチの効果が十分に出るものの，2 スレッドではわずかな効果が出るにとどまり，3 スレッド以上では効果がほとんど出ないことがわかった．これは処理がバンド幅の限界値に律速されていることによる．

3.4.4 データアラインメント

図 7 に `data[]` の先頭論理アドレスと処理時間の関係を示す．図 7 より，先頭論理アドレスを Xeon E5335 のキャッシュラインサイズである 64 バイトの整数倍にアラインした場合のみ高速化しているのがわかる．これは，64 バイト境界にアラインされない場合には，ベクトル単位でランダムアクセスを行うデータ移動ステージにおいて，1 個のベクトルが 2 個のキャッシュラインに分割されるためにバスの転送量が増大するためである．本実験のプログラムの開発段階では malloc 関数によるデフォルトのアラインでは不十分な場合が観測されており，明示的に 64 バイト境界にアラインさせることが重要と言える[17]．

3.5 結果と分析

3.5.1 結　　果

以下では，3.4 で述べた高速化手法を実装せずに書いたコードを簡易コード，高速化手法を用いて Xeon E5335 上で最適化したコードを最適コードと呼ぶ．**表 4** と**図 8** に，Cell チャレンジの最速記録☆，Xeon E5335 上での最適コードの記録☆☆，Xeon E5335 上での簡易

☆ 優勝チームの記録ではなく，各データセットごとに全チームの記録の中で最速の記録を載せている．

☆☆ コンパイラとしては `g++`を用いたが，`icc` を用いても処理時間

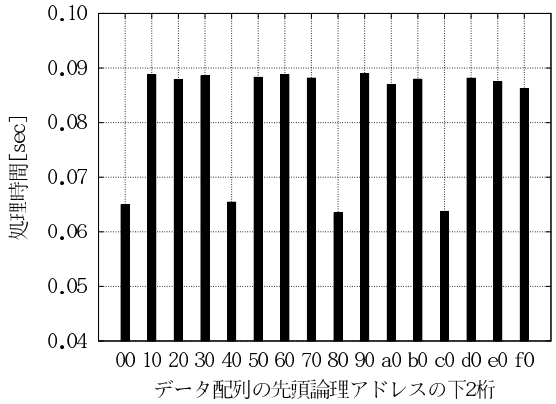


図 7 配列 `data[]` の先頭論理アドレスの下 2 桁と処理時間の関係（データセット 7 のデータ移動ステージ）

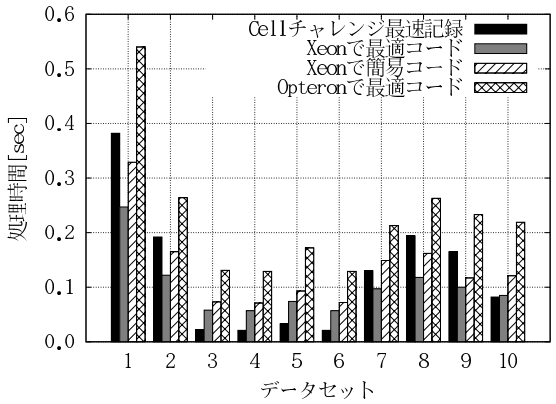


図 8 Cell，Xeon E5335，Opteron 880 の処理時間の比較（表 4 中の（2）（3）（4）（6）をグラフ化したもの）

表 4 Cell，Xeon E5335，Opteron 880 の処理時間の比較（(1) データセット，(2) Cell チャレンジの最速記録，(3) Xeon E5335 上の最適コードの記録（8 スレッド），(4) Xeon E5335 上の簡易コードの記録（8 スレッド），(5) Xeon E5335 上の qsort 関数の記録（1 スレッド），(6) Opteron 880 上の最適コードの記録（16 スレッド）（単位は sec））

(1)	(2)	(3)	(4)	(5)	(6)
1	0.382	0.247	0.329	9.55	0.540
2	0.192	0.122	0.165	5.18	0.264
3	0.022	0.058	0.073	1.42	0.131
4	0.021	0.057	0.071	1.37	0.129
5	0.033	0.074	0.093	1.92	0.172
6	0.021	0.057	0.072	1.15	0.129
7	0.130	0.097	0.149	2.93	0.213
8	0.194	0.118	0.162	4.81	0.263
9	0.165	0.100	0.117	2.61	0.233
10	0.082	0.085	0.121	2.29	0.219

コードの記録，Xeon E5335 上での qsort 関数の記録，Opteron 880 16 コアマシン☆上での最適コードの記録☆☆を比較する．また，最適コードを利用して，Xeon E5335 8 コアマシンと Opteron 880 16 コアマシンのスケーラビリティを測定した結果を**図 9** に示す．

3.5.2 Xeon E5335 と Cell の比較

表 4 および図 8 より，Xeon E5335 上での最適コードはベクトルの成分数が小さいデータセット 1，2，7，8，9 で Cell を 25〜39%上回った．また，簡易コードは最適コードと比較して 17〜54%遅いが，データセット 1，2，8，9 に関しては Cell より処理時間が短い．これに対してベクトルの成分数の大きいデータセット

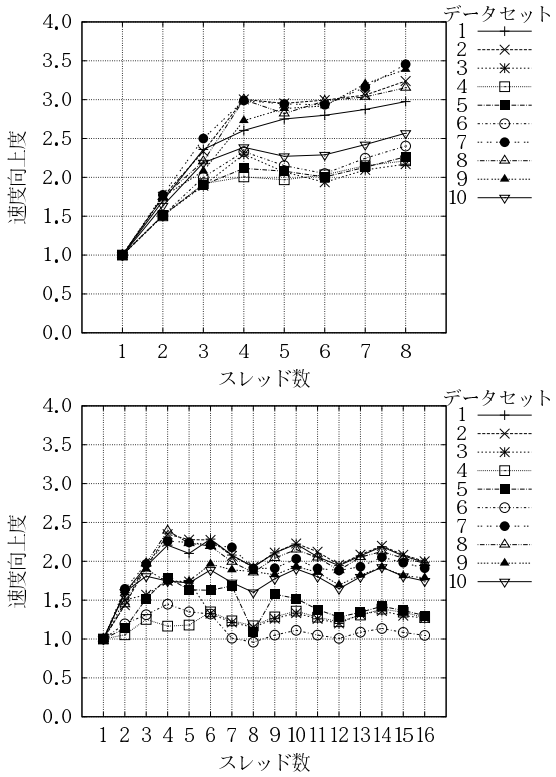


図 9 10 個のデータセットに対するスケーラビリティ（上図：Xeon E5335 8 コアマシン，下図：Opteron 880 16 コアマシン）

3，4，5，6 では，最適コードも Cell に及んでおらず，最も遅いものは Cell の 2.7 倍の時間を要している．さらに，Xeon E5335 上での最適コードの最長記録は最短記録の 4.3 倍であるのに対し，Cell における最長記録は最短記録の 18.2 倍である．これらの結果は，Cell では，ベクトルの成分数が小さい場合には DMA 転送を利用しても性能が出にくいが，ベクトルの成分数が大きい場合には DMA 転送の効果が大きく現れるとい

はほとんど変化しなかった．

☆ Opteron 880（2.4 GHz，2 コア）を 8 基搭載した 16 コアマシンである．

☆☆ Opteron 880 上での実験では Xeon E5335 上での最適コードをほぼそのまま用いた．

う傾向を示している．

3.5.3 Xeon E5335 と Opteron 880 の比較

図 9 を見ると，Xeon E5335 でも Opteron 880 でも 4 スレッド以上では処理時間がほぼ横ばいになっており，速度向上度は 8 スレッドの Xeon E5335 で 2.2〜3.5 程度，16 スレッドの Opteron 880 で 1.0〜2.0 程度である．このようなスケーラビリティの悪さはメインメモリへのアクセス競合に起因しており，キャッシュベースの汎用マルチプロセッサでは単純にコアを追加するだけでは相応の性能向上が望めないことを示唆している．また，マルチコア化が加速する汎用マルチプロセッサの流れとしては，Xeon のようなシステムバス方式の SMP（Symmetric Multi Processing）に代わって，各プロセッサにローカルなメモリを設ける NUMA（Non-Uniform Memory Access）が注目されており，Opteron は SMP と NUMA のメリットを併せ持った SUMO（Sufficiently Uniform Memory Organization）を採用している[15]．ところが本実験では，Opteron 880 は Xeon E5335 の 2.2〜2.6 倍の処理時間を要し（表 4），Opteron 880 の方が Xeon E5335 よりもスケーラビリティが悪い（図 9）という結果になった．この原因は，Opteron はローカルメモリ（同一ダイ上のメモリ）アクセス中心の処理に対しては優れたスケーラビリティを示す一方で，本実験のようにリモートメモリ（異なるダイ上のメモリ）へのアクセスを頻繁に伴う処理に対するスケーラビリティが鈍いことにある．したがって，マルチコア Opteron の性能を発揮させるためには，リモートメモリアクセスを低減するようにデータ配置を工夫することが重要になるが，本実験を含め，プログラムの仕様上の理由からこのようなデータ配置の書き換えが不可能な場合も多い．

4. メモリアクセス最適化ツール

4.1 Tulip

プリフェッチの最適なスケジューリング距離やストリーミングストア利用の是非は，マシン環境やメモリアクセスパターンに依存する．また，スレッドの本数を減らすことがバスの輻輳を抑止して高速化につながる可能性もある．しかし各データセットごとにこれらパラメータの最適値を手動で探索するのは労力が要るため，最適値を自動で提案するツールとして Tulip を作成し，前節で述べた各種分析やソーティングの高速化に役立てた．ソーティングに関しては，試行錯誤による手動でのコードの最適化も試みたが，Tulip の最適化結果にそのまま従ったコード以上に高速化することはできなかった．つまり Tulip の最適化の妥当性を確認できた．

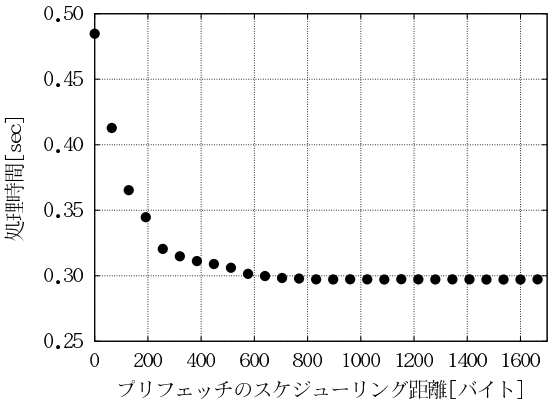


図 11 プリフェッチのスケジューリング距離と処理時間の関係

4.2 Tulip の仕様

Tulip の仕様は以下の通りである：

入力 CPU の個数およびメモリアクセスパターンを入力する（**図 10**（a）（b）（c））．パターンとしては，配列 `src[][]` の 1 次の添字に関して｛シーケンシャル or ランダム｝にリード，配列 `dst[][]` の 1 次の添字に関して｛シーケンシャル or ランダム｝にライトなどに応じて 8 パターンが用意され，配列の 2 次の要素数を 2^m（$4 \leq m \leq 17$）から選択できる（図 10（d）が入力例）．

出力 入力されたメモリアクセスパターンに対して，ターゲットマシンで最大のバンド幅を実現するためのプリフェッチのスケジューリング距離，ストリーミングストア利用の是非，最適なスレッド本数を出力する（図 10（f））．またチューニングの過程も出力する（図 10（e））．

適用条件 プロセッサが SSE および SSE2 をサポートすること，最適化対象となるプログラムを SIMD 化したものがバンド幅にほぼ律速されること，行うメモリアクセスが Tulip に用意されているパターンのいずれかに近似できることが適用可能条件である．

4.3 Tulip の探索アルゴリズム

1 スレッドで 1 次元配列をシーケンシャルにリードするコードに関して，プリフェッチのスケジューリング距離と処理時間の関係を調べると**図 11** のようになる．また，図 11 に見られるグラフ特性は，他のマシン環境や多少ランダムな配列アクセスに対してもほぼ成立する．この性質を利用して Tulip では以下の探索を行い，数十秒〜数分で最適化結果を出力する：

（1） スレッド本数を，コア数 → コア数−2→ コア数−4→ ···→2→1 の順に変化させ，各スレッド本数に

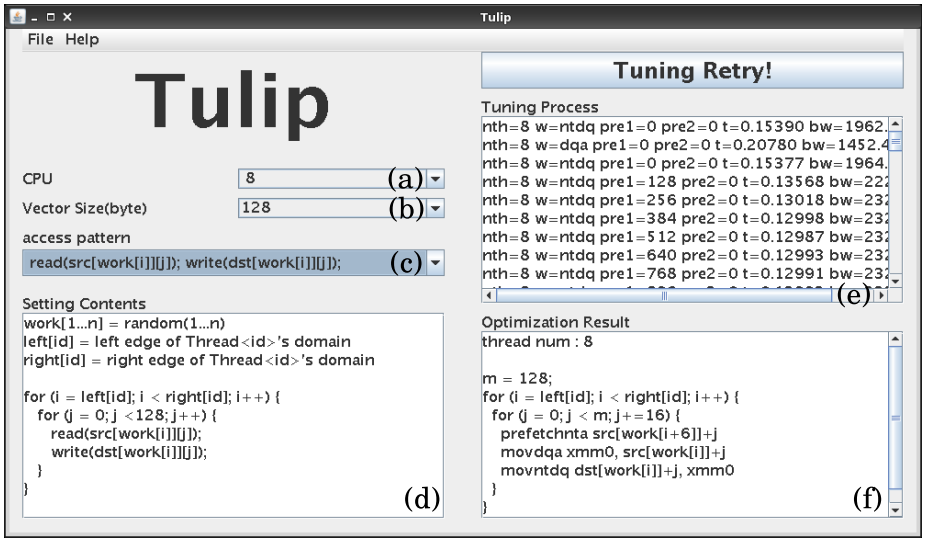


図 10 Tulip

対して（2）を行う．

（2） まずストリーミングストアを利用するとして，配列 `src[][]` に対するプリフェッチのスケジューリング距離を 128 バイトずつ増加させていき，処理時間が十分に飽和するときの処理時間を記録する．次にストリーミングストアを利用しないとして，配列 `src[][]` と配列 `dst[][]` に対するスケジューリング距離を独立に増加させていき，十分に飽和するときの処理時間を記録する．この過程で，最良解に達しないと予測される探索は飽和する前に早期に枝刈りする．

（3） 以上で得られた探索結果において，処理時間を最小化するときのパラメータの組合せ（スレッド本数，スケジューリング距離，ストリーミングストア利用の是非）を選択して出力する．この選択時には，処理時間が同程度ならばスレッド本数が多くスケジューリング距離が長いものほど優先する．

4.4 Tulip を利用したプログラムの最適化と評価法

Tulip による最適化結果をプログラムに反映させた後，図 10（f）中の最も内側の `for` ループに相当するループの内部にダミー命令を挿入し，挿入個数と処理時間のグラフを作成することにより，高速化の余地があるか否かを見極めることができる．ここでダミー命令とは，1 命令の消費クロック数が小さく，かつプログラムの挙動に影響を及ぼさないような命令を意味する．たとえばデータセット 4 のデータ移動ステージの場合には，ダミー命令☆の挿入個数と処理時間の関係は**図 12** になる．図 12 より，このステージはバンド幅の限界値に律速されており，各ループに対してダミー命令 600 個に相当する無駄な演算を追加したとしても処理時間は変化しないことが読み取れる．

また，表 4 において，データセット 3，4，5，6 に関しては Xeon E5335 上での最適コードが Cell に及んでいないが，これらのデータセットではキー計算ステージとデータ移動ステージの 2 つが全実行時間の約 99%を占めており（図 5），しかもこの 2 つのステージがバンド幅の限界値に律速されていることが今述べた評価法によって判明したため，Xeon E5335 上でのこれ以上の高速化は困難であると判断した．

以上のように Tulip を利用することで，SIMD やビット演算などで闇雲に演算処理を高速化するのではなく，バンド幅との兼ね合いを観察した上での適切な評価を加えることができる．

4.5 さまざまな汎用プロセッサ上での検証

表 5 に示す 8 種類の汎用プロセッサに関して，Tulip による最適化の効果を検証する．

まず，データセット 7 を題材にして，キー計算ステージとデータ移動ステージに対して Tulip による最適化を施した結果を**表 6** および**図 13** に示す．図 13 より，Tulip の最適化により 18〜40%の高速化が実現されている．また最適化の効果を内容別に見ると，ストリーミングストアが効くプロセッサと，スケジューリング距離の最適化が効くプロセッサと，全ての最適化が揃った場合に複合的に効くプロセッサがあり，傾向はさまざまだと言える．

次に，ストリーミング処理の簡易モデルとして，静的に確保して初期化した 320 MB の `float` 型配列 `src[]`

☆ ここでは `cmp %eax,%ebx` を使用した．

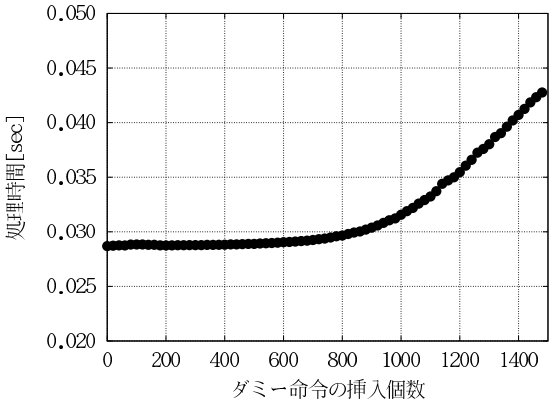


図 12 ダミー命令の挿入個数と処理時間の関係（データセット 4 のデータ移動ステージ）

表 5 Tulip を検証する 8 種類の汎用プロセッサ

(a)	Xeon E5335（2.0 GHz，4 コア）× 2
(b)	Xeon 5140（2.33 GHz，4 コア）
(c)	Core2Duo E6400（2.13 GHz，2 コア）
(d)	Pentium M（1.7 GHz，1 コア）
(e)	Celeron D（2.93 GHz，1 コア）
(f)	Opteron 880（2.4 GHz，2 コア）× 8
(g)	Opteron 875（2.2 GHz，2 コア）× 4
(h)	Athlon64 X2 3800+（2.0 GHz，2 コア）

表 6 データセット 7 のメモリアクセスパターンに関して Tulip が出力した最適化結果（左表：キー計算ステージ，右表：データ移動ステージ（1）プロセッサ，(2）最適なスレッドの本数，(3）ストリーミングストアを利用するか否か，(4）プリフェッチのスケジューリング距離（バイト）☆）

(1)	(2)	(3)	(4)
(a)	8	yes	256
(b)	4	yes	128
(c)	2	yes	384
(d)	1	yes	384
(e)	1	yes	512
(f)	4	yes	128
(g)	4	yes	640
(h)	2	yes	640

(1)	(2)	(3)	(4)
(a)	8	yes	256
(b)	4	yes	128
(c)	2	yes	256
(d)	1	yes	128
(e)	1	yes	128
(f)	16	yes	128
(g)	4	yes	128
(h)	2	yes	128

と `dst[]` に対して，`src[]` の各要素の値を 2 倍にして配列 `dst[]` に格納する処理に関して Tulip の最適化を施した結果を**図 14** に示す．図 14 を見ると，20〜48%の高速化が実現されており、Tulip による最適化の有効性を確認できる．

☆ データ移動ステージではベクトル単位でのランダムなリードを行うため，たとえば 128 バイト先をプリフェッチする場合には，(論理アドレスとしての単純な 128 バイト先ではなく，）将来ベクトルを読み取る順番を勘案した上での 128 バイト先をプリフェッチする必要がある．

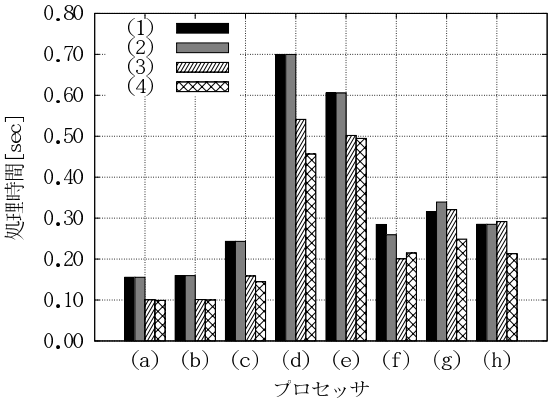


図 13 データセット 7 に関する 8 種類の汎用プロセッサ上での Tulip による最適化の効果（凡例において，(1）は SIMD 化したコードをコア数に等しいスレッド数で実行した場合，(2）は SIMD 化したコードを Tulip の出力した最適スレッド数で実行した場合☆，(3）は（2）に加えて Tulip の出力したストリーミングストアを導入した場合，(4）は（3）に加えて Tulip の出力した最適スケジューリング距離でプリフェッチを発行した場合（=Tulip による最適化を全て適用した場合））

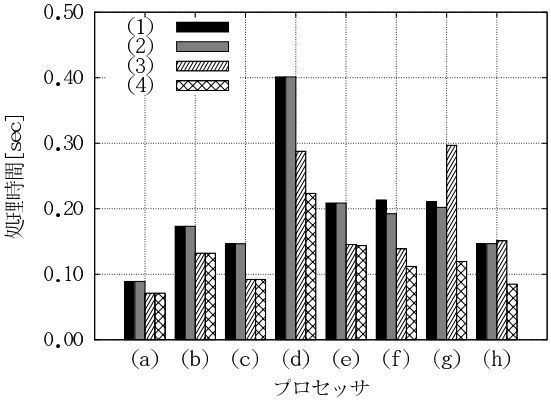


図 14 `src[]` の各要素の値を 2 倍にして `dst[]` に格納する処理に関する 8 種類の汎用プロセッサ上での Tulip による最適化の効果（凡例は図 13 と同じ）

5. 結　　論

Cell でも汎用マルチプロセッサでも，アルゴリズム並列化，SIMD 化，並行実行性の向上（Cell では DMA ダブルバッファリング，汎用マルチプロセッサではプリフェッチとストリーミングストア），データアラインメントが高速化の鍵になる[4),12),13)]．つまり高速化のための「手段」は共通である．しかも同程度の記録が出た．違うのは，「高速化のためのプログラミングエフォートのメインが，初期的なプログラミングの難しさにあるのか，それとも高速化手法の実装のしにくさにあるのか」である．

Cell 上でのプログラミングでは，そのアーキテクチャ故に SIMD の利用は必然であるし，DMA 転送の効率化を考えれば DMA ダブルバッファリングや 16 バイト境界へのアラインも必然になる．これらの実装には特殊なプログラミング技術を要するが，明示的な制御が可能なため，実装すればそれがそのまま効果として現れやすくチューニングも行いやすいと言われる．これに対して汎用マルチプロセッサでは，高度に備わった暗黙的なハードウェア機構のおかげで容易なプログラミングで効率的なプログラムを開発できる．しかし高速化手法を単純に実装するだけでは十分な効果が望めず，キャッシュ機構の動作を見極めた上でのストリーミングストア，キャッシュラインサイズに応じたデータアラインメント，SUMO においてはローカルメモリとリモートメモリを意識したデータ配置など，暗黙的な機構と親和性の高いプログラミング手法を導入することが不可欠になるが，これらの暗黙的動作を見越した上での実装を施すのは一般に困難であり，チューニングも難しい．

本稿では，Cell が圧倒的に速いという従来の比較結果に対して，汎用マルチプロセッサでも高速化手法を適切に実装すれば Cell と同程度の性能が見出せる事例を報告した．また，高速化手法の実装を支援するツールとして Tulip を作成してその効果を検証した．今後，進化する汎用マルチプロセッサの性能がさらに引き出されていくことを期待したい．

謝辞 本実験を進めるに当たって有意義なアドバイスをくださった五島正裕准教授（東京大学情報理工学系研究科電子情報学専攻），Cell チャレンジと同等のデータ生成ルーチンを汎用プロセッサ用に提供してくださった吉瀬謙二講師（東京工業大学大学院情報理工学研究科）をはじめとする Cell Speed Challenge 2007 実行委員会の皆様，Cell チャレンジで優秀な記録を残してくださった参加チームの皆様に謹んで感謝の意を表します．

参 考 文 献

1) AMD: AMD Completes ATI Acquisition and Creates Processing Powerhouse. http://www.amd.com/us-en/Corporate/VirtualPressRoom/0,,51_104_543~113741,00.html.
2) Williams, S., Shalf, J., Oliker, L., Kamil, S., Husbands, P., Yelick, K.: The Potential of the Cell Processor for Scientific Computing, *Computing Frontier 2006 Paper* (2006).
3) Benthin, C., Wald, I., Scherbaum, M., Friedrich, H.: Ray Tracing on the Cell Processor, *Interactive Ray Tracing 06* (2006).
4) D.Hung, L.: Implementation of a Parallel Radix Sort on Cell Processor, Cell Speed Challenge 2007 規定課題部門 (2007).
5) 拓殖宗範, 白木厚史, 増田信之, 伊藤智義: Cell BE における計算機合成ホログラムのソフトウェア開発, Cell Speed Challenge 2007 自由課題部門 (2007).
6) 五十嵐潤: Cell Broadband Engine^{TM} による神経回路網モデルの並列計算の有効性, Cell Speed Challenge 2007 自由課題部門 (2007).
7) Cell Speed Challenge2007 実行委員会: Cell Speed Challenge 2007. http://www.hpcc.jp/sacsis/2007/cell-challenge/.
8) Wald, I., Boulos, S., Shirley, P.: Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies, *ACM Transactions on Graphics* (2007).
9) Whaley, R., Petitet, A., J.Dongarra, J.: Automated Empirical Optimization of Software and the ATLAS Project, *Parallel Computing* (2001).
10) 木下靖夫, 片桐孝洋, 本多弘樹, 弓場敏嗣: SMP マシン上での BLAS ライブラリ用自動チューニング機構の設計と実装, 電子情報通信学会総合大会講演論文集 (2004).
11) Kensler, A., Shirley, P.: Optimizing Ray-Triangle Intersection via Automated Search, *The 2006 IEEE Symposium on Interactive Ray Tracing* (2006).
12) 花岡俊行: Cell Speed Challenge 2007 規定課題部門でおこなった開発のすべて, Cell Speed Challenge 2007 規定課題部門 (2007).
13) 木下正喬, 藤原賢二, 岩見宏明, 浜田悠樹, 指導教員：早川潔: Cell Speed Challenge 2007 規定課題部門 開発記録, Cell Speed Challenge 2007 規定課題部門 (2007).
14) 近藤嘉雪: Java プログラマのためのアルゴリズムとデータ構造，ソフトバンクパブリッシング (2004).
15) Ludvig, M.: Linux on AMD64 (2005). http://www.logix.cz/michal/doc/LinuxAMD64/Linux-on-AMD64.pdf.
16) Thakkar, S.T., インテル・コーポレーション, マイクロプロセッサ・プロダクト・グループ, Huff, T., インテル・コーポレーション, マイクロプロセッサ・プロダクト・グループ: インターネット・ストリーミング SIMD 拡張命令, *Intel Technology Journal Q2* (1999).
17) Intel: IA-32 インテル® アーキテクチャ最適化リファレンス・マニュアル (2006). http://download.intel.com/jp/developer/jpdoc/IA32_Final_i.pdf.

☆ (f) の場合の最適なスレッド本数は 4 とした．