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—

東京大学 工学部 電子情報工学科 近山·田浦研究室　原 健太朗
東京大学 情報理工学系研究科 坂井·五島研究室　塩谷 亮太
東京大学 情報理工学系研究科 准教授　田浦 健次朗

2008. 6. 12

発表の流れ

- はじめに
- 関連研究
- 背景
- 汎用マルチプロセッサの高速化手法
- 結果と分析
- メモリアクセス最適化支援ツール
- 結論

1. はじめに

本研究の背景

- 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かすテクノロジーが人気
  - → GPU を汎用 CPU に統合する技術
  - → PlayStation 3 に搭載された Cell
- Cell：
  - → 強力な演算性能と広帯域幅
  - → 数値解析やマルチメディア処理での将来性
- 一方で汎用プロセッサの重要性も増大
  - → 高い汎用性
  - → 高い普及度
  - → プログラミングの容易さ

本研究の動機

- Cell が汎用プロセッサに対して圧倒的高性能を記録したという報告が多数
- しかし従来の研究の多くは，比較対象の汎用プロセッサに対してどの程度の高速化手法を導入した上での比較なのかが不明確
- 現在の汎用プロセッサの進化はめざましく，十分なエフォートを投入すれば Cell 並の性能を引き出せるかも

本研究の内容

- Cell チャレンジ 2007 の決勝記録を題材に，汎用プロセッサに対して十分な高速化手法を駆使した上での Cell との性能比較
  - ➜ 10 個中 5 個のデータセットで Xeon が Cell を 25〜39% 上回った
- ここで得られたノウハウを基に，汎用プロセッサ向けメモリアクセス最適化支援ツールとしての Tulip を作成し，その効果を検証
  - ➜ Tulip の最適化により 18〜48% 高速化

2. 関連研究

関連研究：Cell と汎用プロセッサの性能比較 (1)

- Cell チャレンジ 2007 報告 [Hanaoka et al,2007]
  - → ソーティング，合成ホログラムの計算，神経回路網の計算において，Cell が，Athlon XP 2800+，Opteron，Core2Quad に対して 9〜29 倍の性能を発揮
  - → 特にソーティングでは，SIMD 化による演算処理の高速化，DMA ダブルバッファリングによる並行実行性の向上，DMA 転送時の 16 バイト境界へのアラインが高速化の鍵

関連研究：Cell と汎用プロセッサの性能比較 (2)

- The Potential of the Cell Processor for Scientific Computing [Williams et al,2006]
  - ➜ SpMV や FFT などの数値科学演算を題材にした性能比較で，Cell が Opteron 248 に対して 4〜37 倍の高性能を記録
- Ray Tracing on the Cell Processor [Benthin et al,2006]
  - ➜ 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 システム上で最速とされるレイトレーシングアルゴリズムを Opteron 上で実装した場合と比較した結果，Cell が Opteron の 5〜15 倍の性能を発揮

関連研究：Cell と汎用プロセッサの性能比較 (3)

- 以上の研究は・・・
  - → Cell の高速化に焦点を当てており，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を施したのかが不明確
- 本研究では・・・
  - → Xeon E5335 8 コアマシンに対して高速化のための実装を駆使した上での Cell との性能比較

関連研究：ターゲットマシン上での最適化 (1)

- Automated Empirical Optimization of Software and the ATLAS Project [Whaley et al,2001]
  - → 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
  - → 木下らの研究 (2004) によれば，Origin 3400 を 16 個用いた GEMM において，ATLAS BLAS が通常の BLAS と比較して 38 倍の性能を記録
- Optimizing Ray-Triangle Intersection via Automated Search [Kensler et al,2006]
  - → レイトレーシングのプリミティブ演算に関して，Opteron，Xeon，Core Duo，Pentium4 などに対して遺伝的アルゴリズムを用いて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

関連研究：ターゲットマシン上での最適化 (2)

- 以上の研究は・・・
  - → 主に数値演算の最適化が目標
- 本研究では・・・
  - → L2 キャッシュとメインメモリ間のデータ転送の最適化が目標

3. 背景

- 実験環境
- Xeon と Cell のアーキテクチャ比較
- Cell チャレンジの規定課題（ソーティング）
- 並列ラディックスソートのアルゴリズム

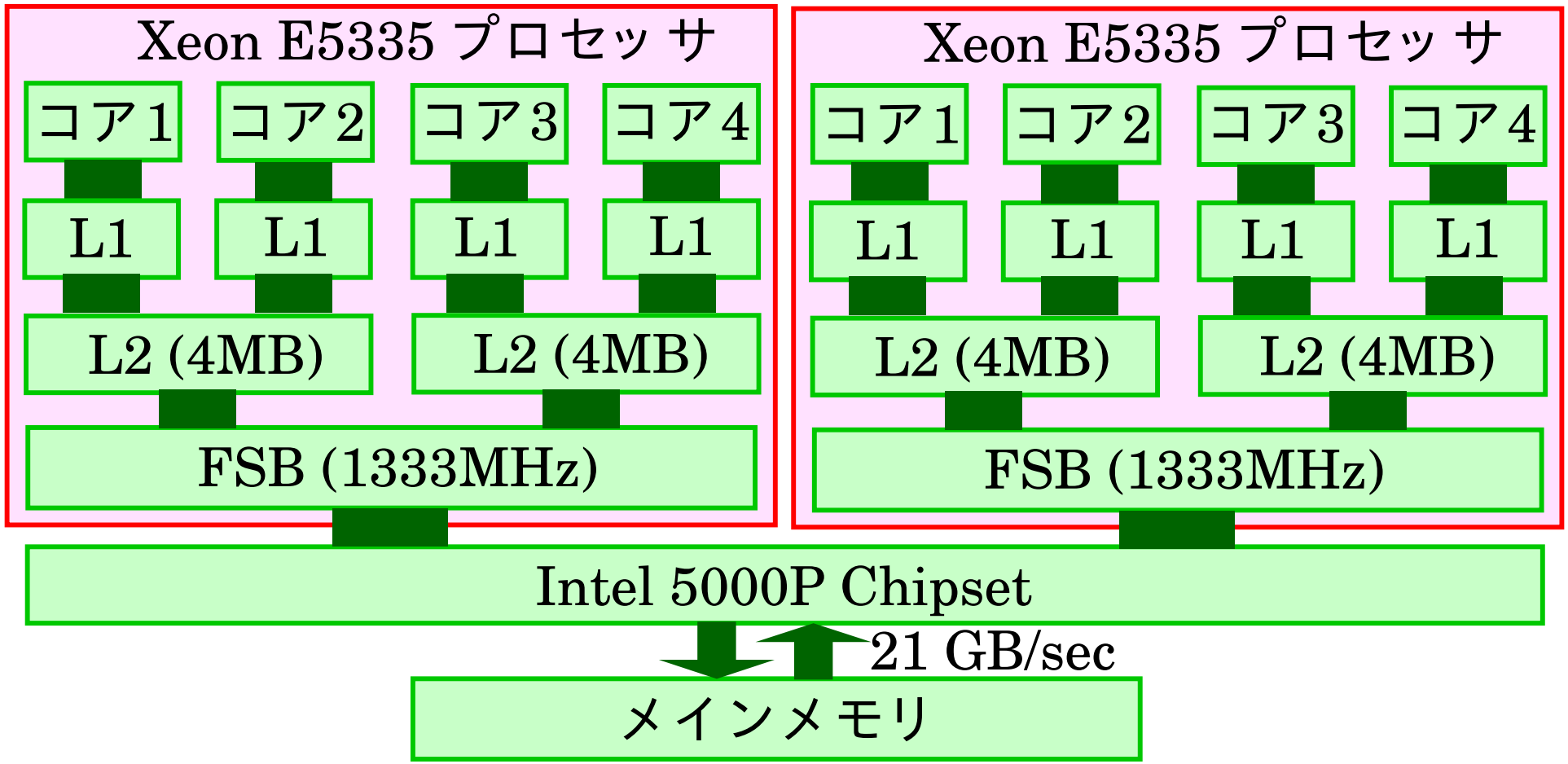
実験環境

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

- **以下，この** Xeon E5335 **の高速化を論じる**

アーキテクチャ：Xeon（図解）

- Intel Xeon E5335（4 コア ×2）による 8 コアマシン
- 4 MB の共有 L2 キャッシュが 4 個

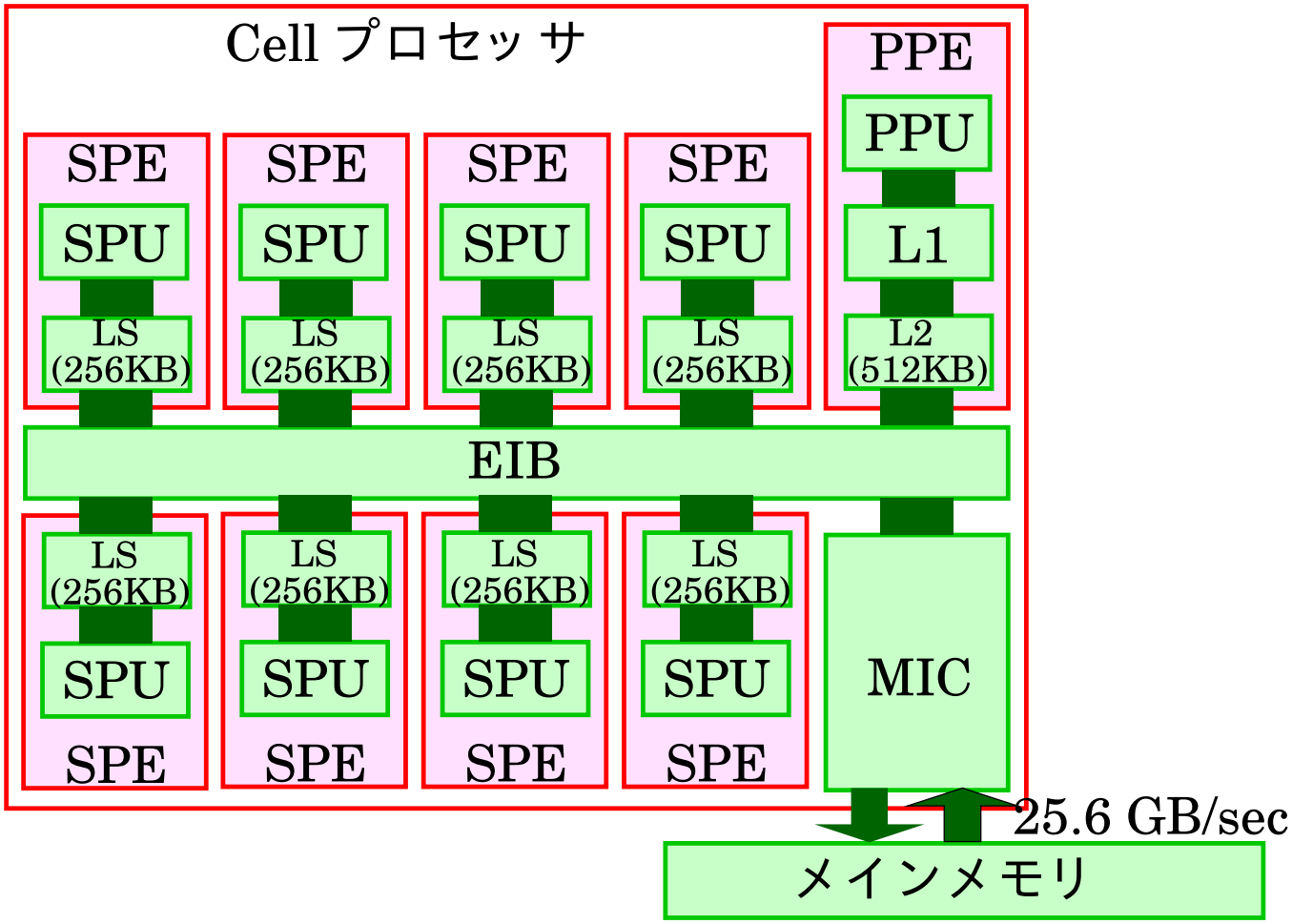


アーキテクチャ：Xeon（特徴）

- 利点：
  - → 高い普及度と汎用性
  - → 高度なハードウェア機構（アウトオブオーダ実行，プリフェッチ，キャッシュ管理 etc）のおかげで，効率的なプログラムの開発が容易
- 欠点：
  - → バスの輻輳やキャッシュコヒーレンシ保持の影響で，スケーラビリティが悪い
  - → 「暗黙的」なハード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挙動が読みにくい

アーキテクチャ：Cell（図解）

- 1 個の PPE（PowerPC Processor Element）と 8 個の SPE（Synergistic Processor Element）
- 各 SPE には 256 KB の LS（Local Store）



アーキテクチャ：Cell（特徴）

- 利点：
  - → 強力な SIMD 命令セットによる高い演算性能
  - → LS と主記憶とのデータ転送は「明示的」な DMA 転送によるため，キャッシュコヒーレンシ保持が不要でバンド幅が広い
  - → 予測不可能性を回避しており挙動が読みやすい
- 欠点：
  - → 手動での DMA 転送やスケジューリングなど，高度なプログラミング技術が必要
  - → 複雑な条件分岐に弱い

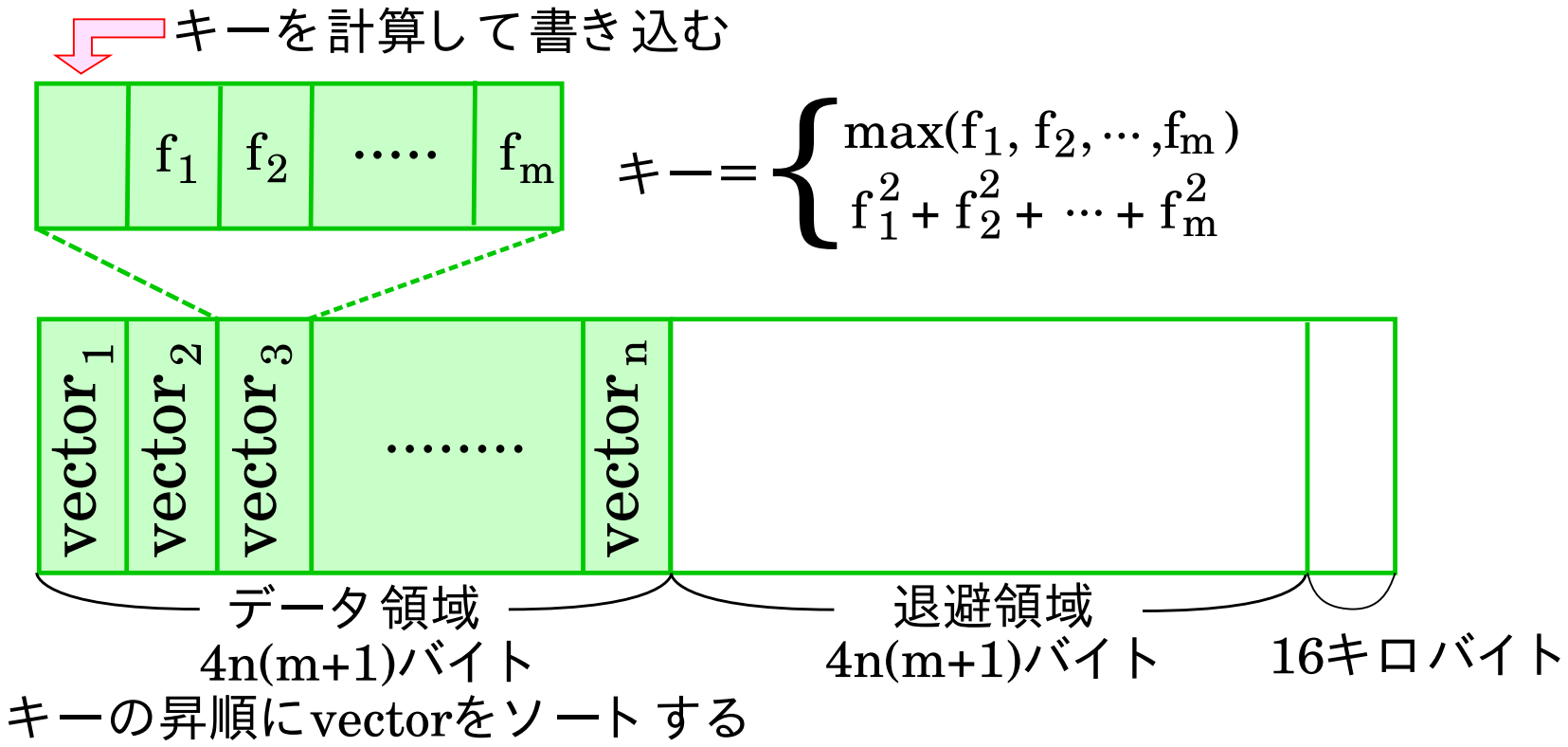
アーキテクチャ：Xeon と Cell のスペック比較

	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
単精度浮動小数点数演算性能	128 Gflops	204.8 Gflops
理論最大メモリバンド幅	21 GB/sec	25.6 GB/sec

- **ただし，**Xeon E5335 **の実効メモリバンド幅は** 5.6 GB/sec

Cell チャレンジの規定課題：ソーティング

- 各要素が浮動小数点数の m 次元ベクトル n 個を，各ベクトルのキー（最大値 or 二乗和）の昇順にソーティング
- 使用可能メモリ領域は入力データの 2 倍 + 16 KB のみ
- m,n, キーが異なる 10 個のデータセットに対する処理時間を競う



Cell チャレンジの規定課題：10 個のデータセット

- データセット 3,4,5,6 はベクトルの成分数 m が大きい
- データセット 1,2,7,8,9 は m が小さい

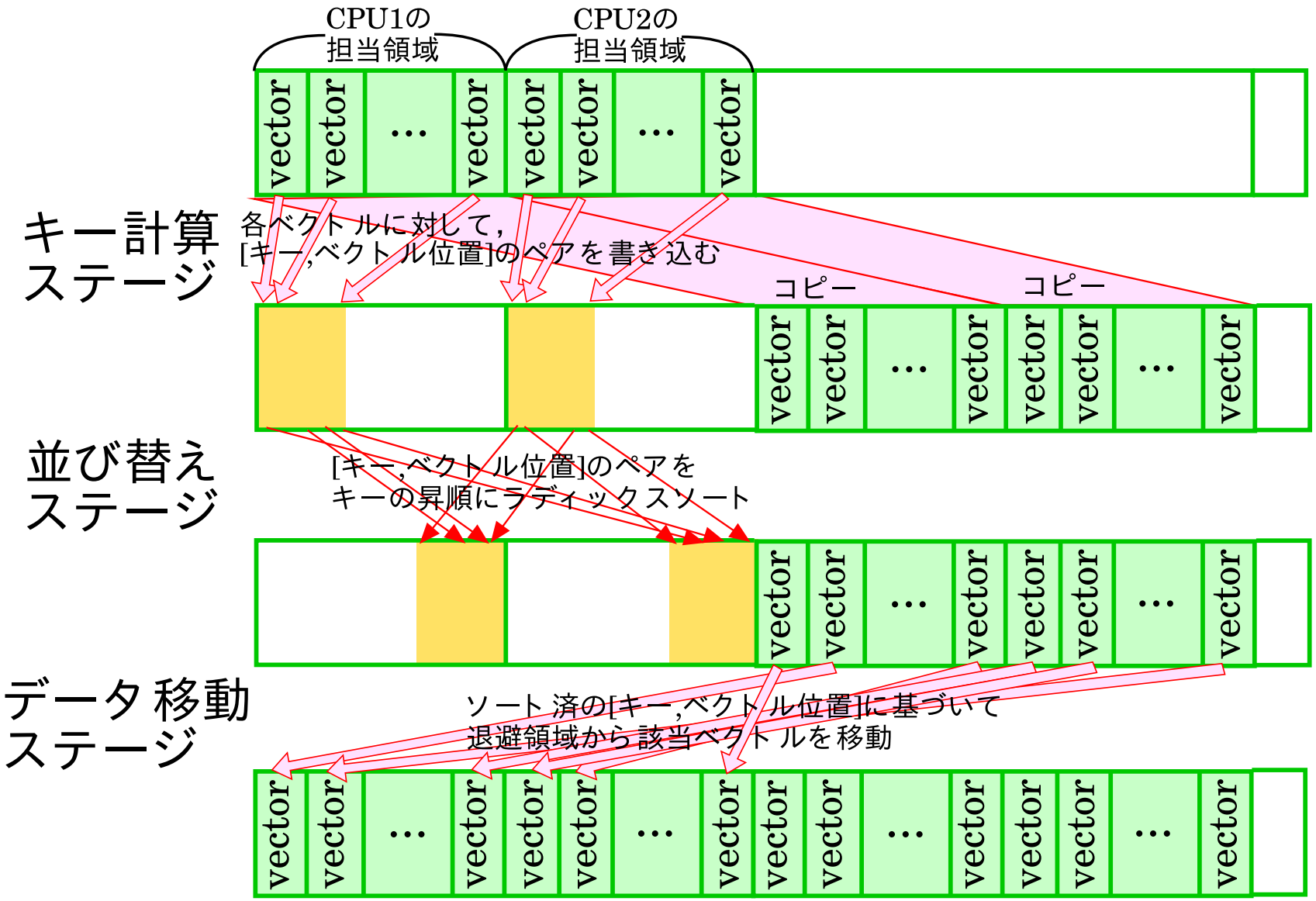
No.	m	n	キーの種類
1	3	5898240	最大成分
2	7	2621440	二乗和
3	1023	20480	二乗和
4	2047	10240	最大成分
5	255	102400	最大成分
6	1023	20480	二乗和
7	15	1572864	最大成分
8	7	2621440	二乗和
9	7	2621440	最大成分
10	31	819200	最大成分

アルゴリズム：並列ラディックスソート（特徴）

- 並列ラディックソートの特徴：
  - → 固定長のキーに対して安定で高速なソーティング
  - → 時間計算量 $O(n)$
  - → 効率的な並列化が可能

アルゴリズム：並列ラディックスソート（図解）

（ CPU2個で行う 場合）



4. 汎用マルチプロセッサの高速化手法

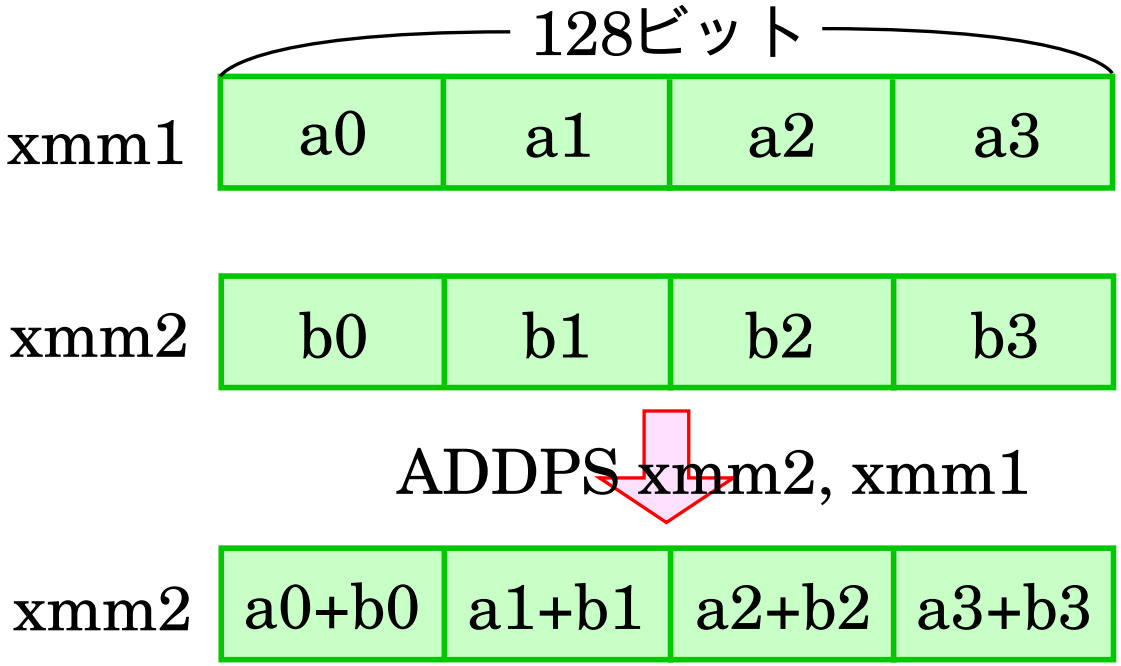
- SIMD
- プリフェッチ
- ストリーミングストア
- データアラインメント

高速化手法

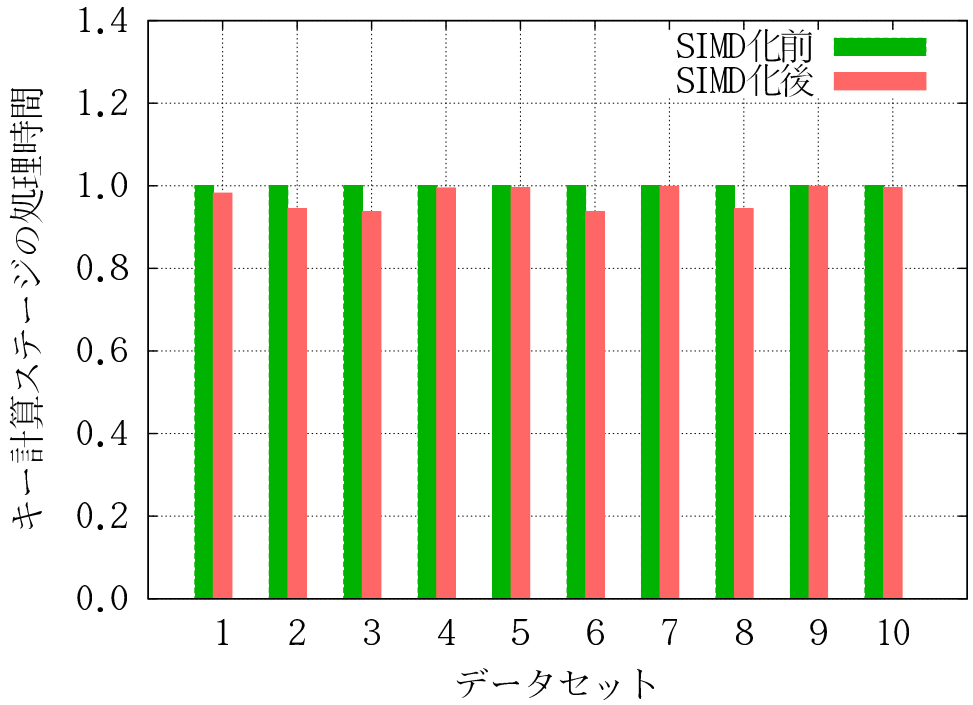
- **演算処理の高速化：**
  - → SIMD
- **データ転送の高速化：**
  - → **プリフェッチとストリーミングストアによる並行実行性の向上**
  - → **適切なデータアラインメント**

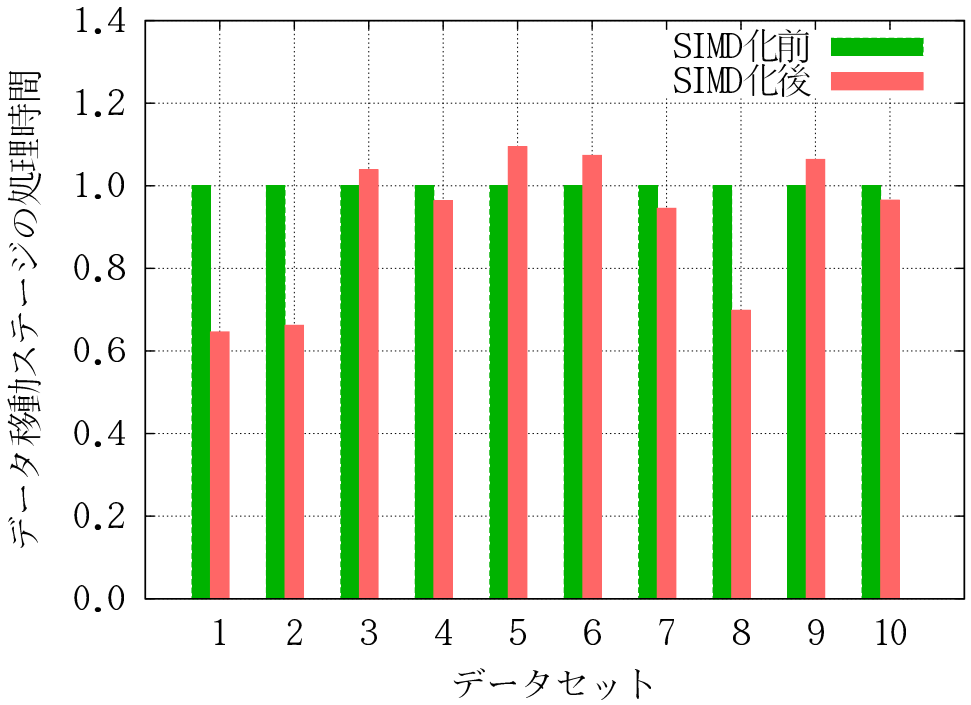
演算処理の高速化：SIMD（説明）

- 1 命令で複数のデータを演算処理する命令レベルの並列化手法



演算処理の高速化：SIMD（結果）

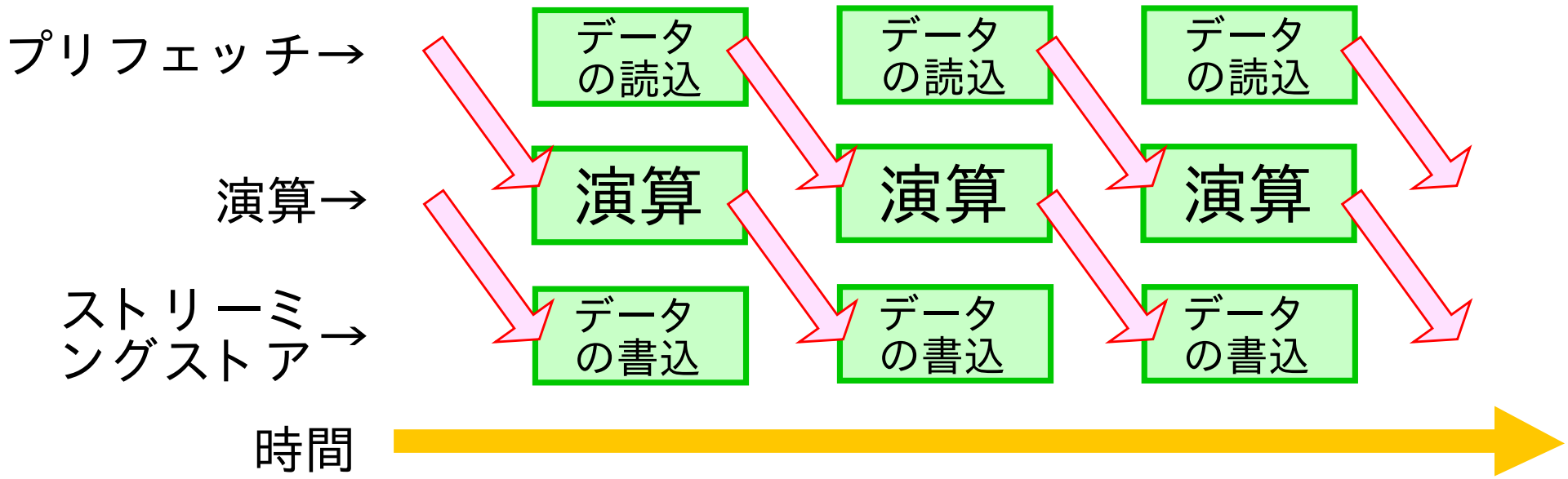




- 最大 35% 高速化したが，多くのデータセットでは変化なし
  - → 処理がデータ転送のバンド幅に律速されているため
  - → よって，本実験ではデータ転送の高速化が必須

データ転送の高速化：並行実行性の向上

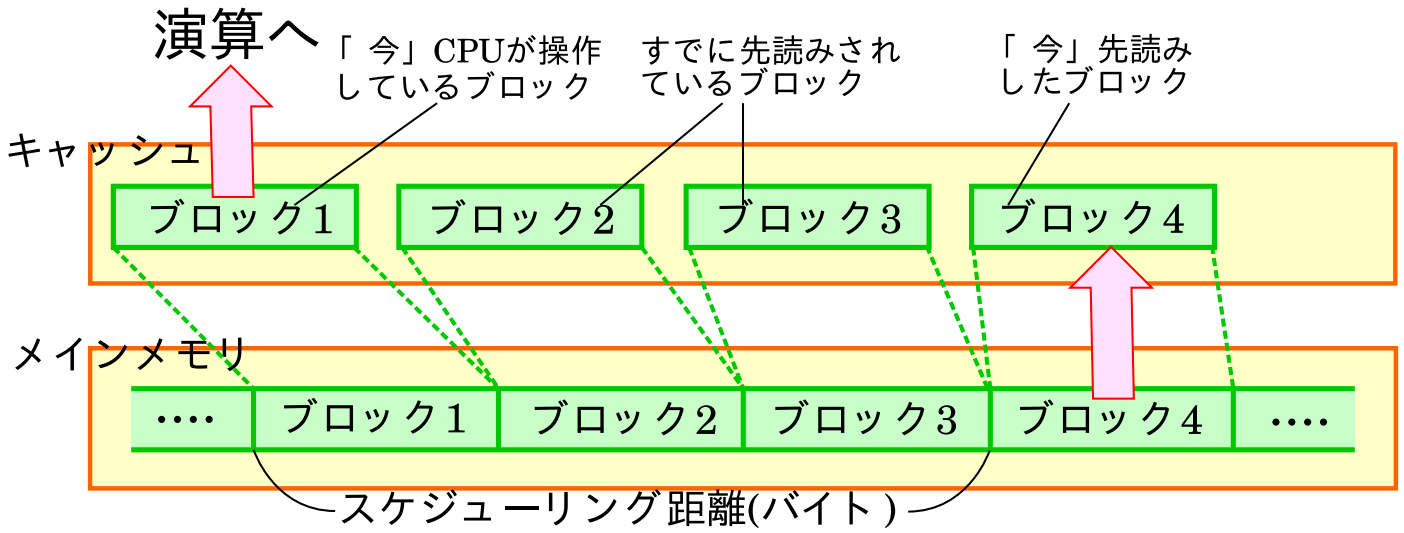
- プリフェッチ：
  - → 次のデータの読込と現在の演算をオーバーラップ
- ストリーミングストア：
  - → 前のデータの書込と現在の演算をオーバーラップ



データ転送の高速化：プリフェッチ（説明）

- 「暗黙的」なハードウェアプリフェッチと「明示的」なソフトウェアプリフェッチがある
  - → ソフトウェアプリフェッチではデータを読み込むキャッシュ階層やスケジューリング距離が制御可能

```
for(i=0; i<n; i++)
{
    prefetch(data[i + d ]);
    read(data[i]);
}
```

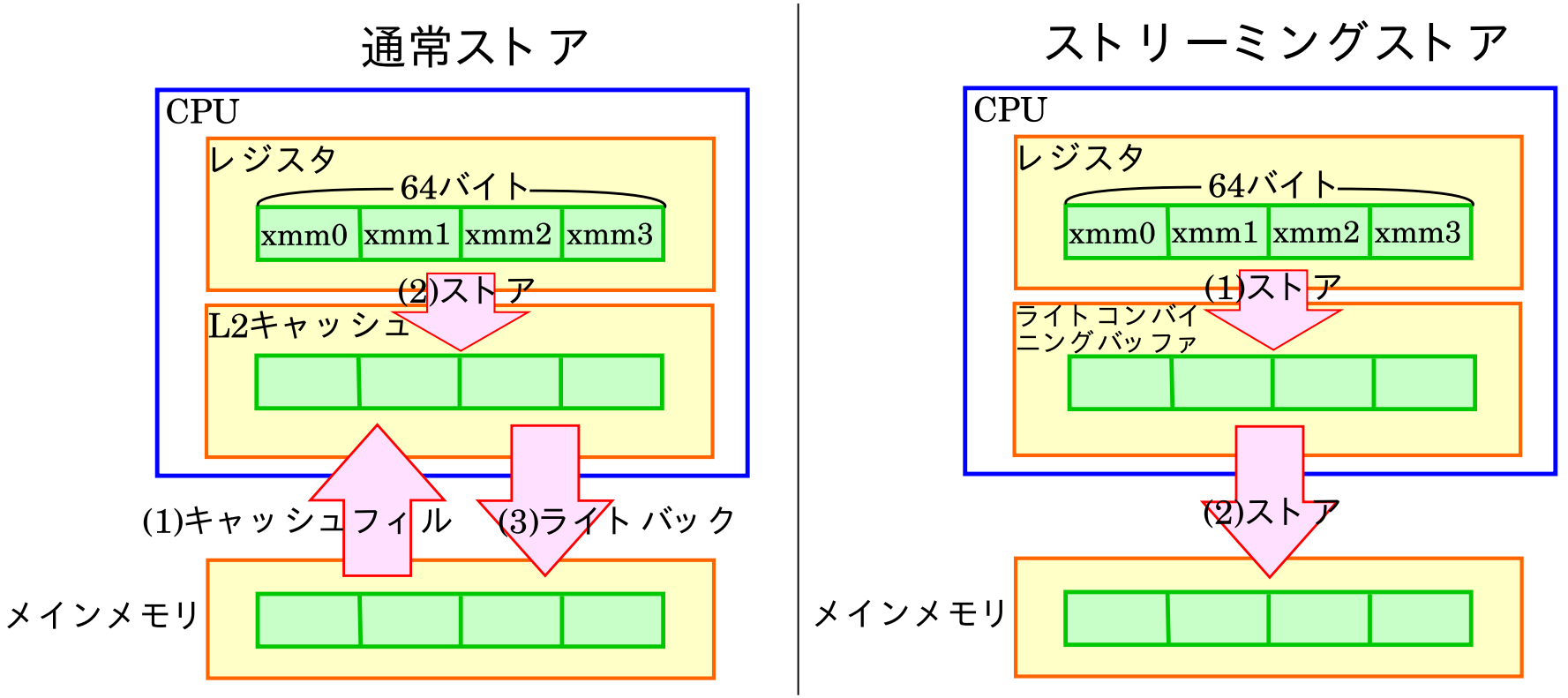


データ転送の高速化：プリフェッチ（結果）

- ハードウェアプリフェッチを無効化した上でソフトウェアプリフェッチの効果を検証した結果，
  - → 1 スレッド実行時には十分な効果があるが，多スレッドではバスが輻輳してバンド幅に律速されてしまうため効果なし

データ転送の高速化：ストリーミング ストア（説明）

- ライトコンバイニングバッファを経由して，レジスタから主記憶への，キャッシュを介さないストアを実現する仕組み
  - ➜ 通常ストアでは 2 回のバストランザクションが必要だが，ストリーミングストアでは１回で済む
  - ➜ キャッシュを介さないためキャッシュ汚染を防げる



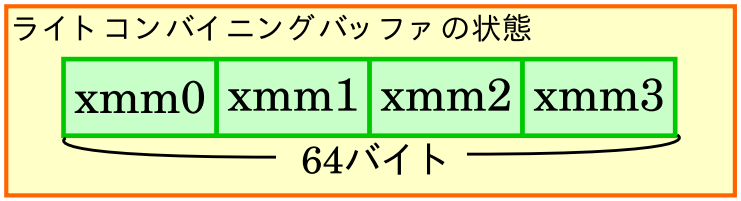
※キャッシュラインサイズを64バイトとする

データ転送の高速化：ストリーミング ストア（補足）

- ストリーミングストアは，キャッシュライン単位で時間的に連続的なストアを行った場合のみ効果を発揮
  - → 適度にループアンローリングした上で命令順序を上手に交換するなどの工夫が必要
  - → 連続的にできないならば通常ストアの方が高速

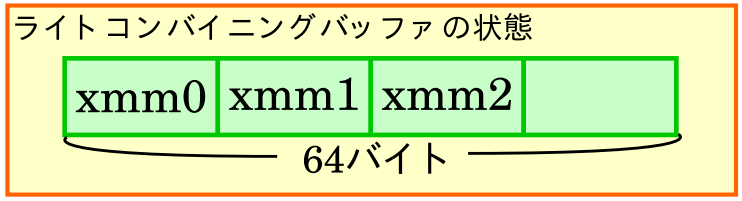
0.68 sec

```
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
}
```



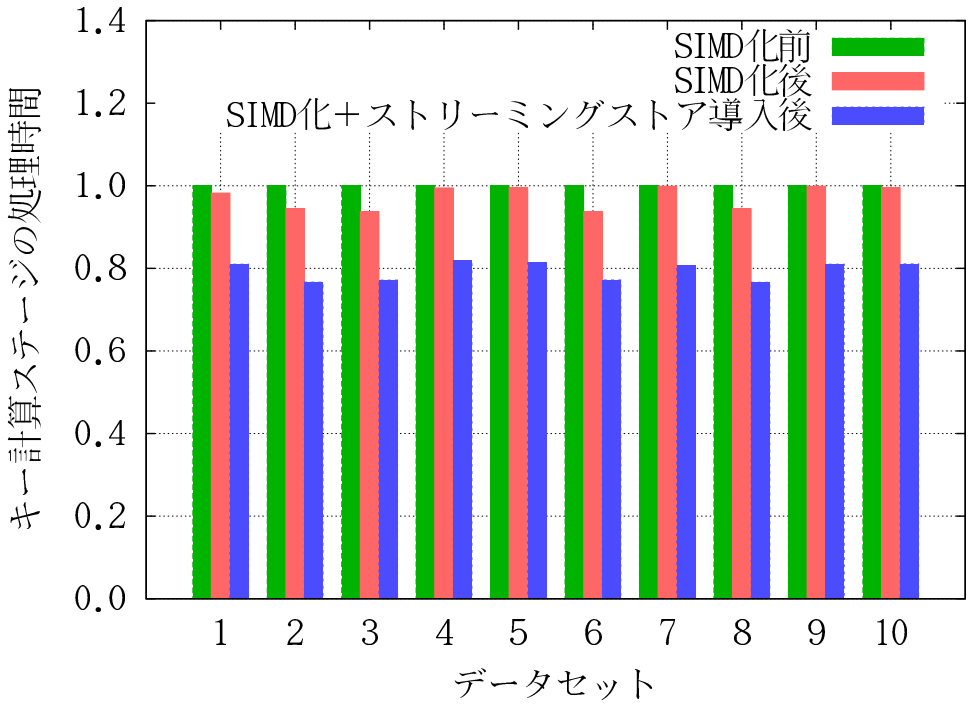
14.9 sec

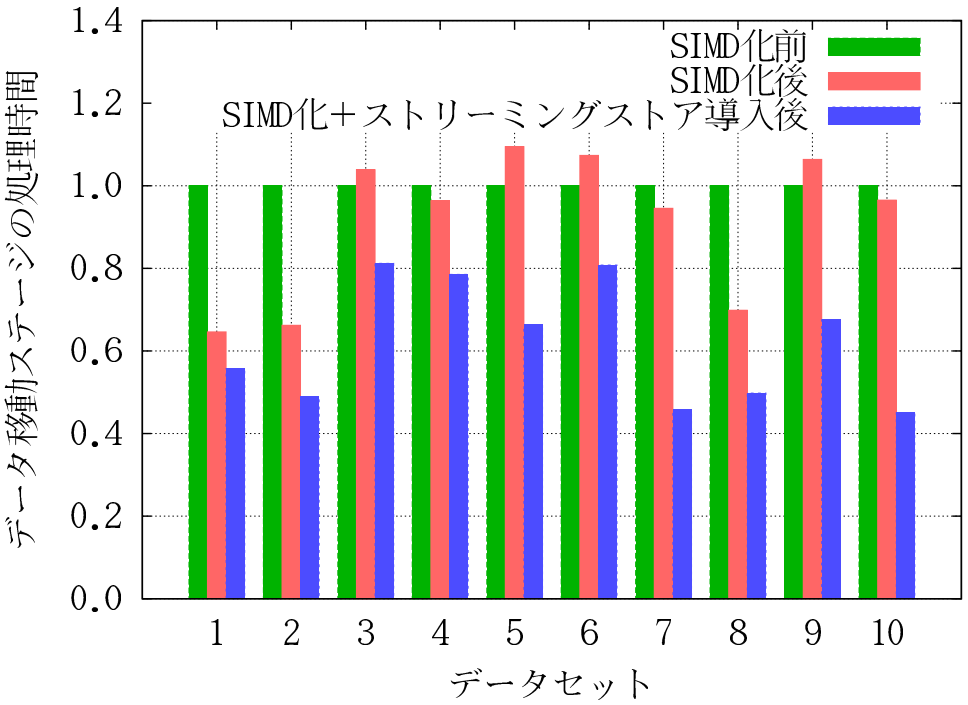
```
char buf[SZ],*p;
for(p=buf; p<buf+SZ; p+=64)
{
      movntps p , xmm0
      movntps p+16 , xmm1
      movntps p+32 , xmm2
}
```



※キャッシュラインサイズを64バイトとする
※movntps : 128ビットレジスタからメインメモリへのストリーミングストア命令

データ転送の高速化：ストリーミング ストア（結果）

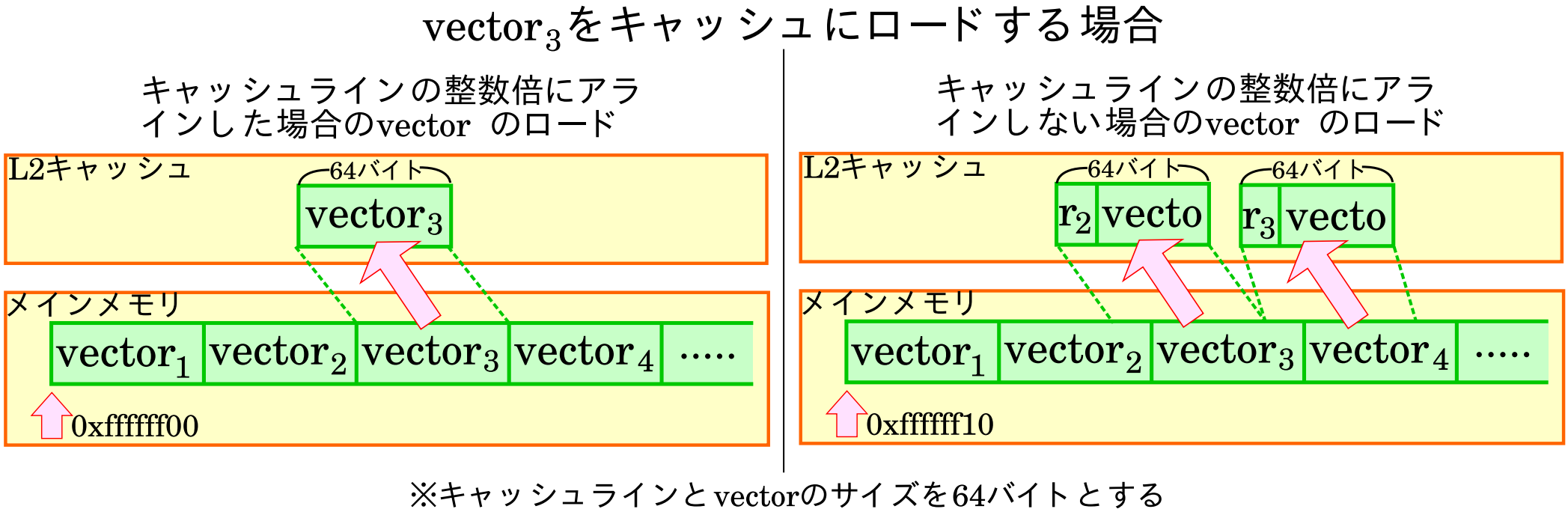




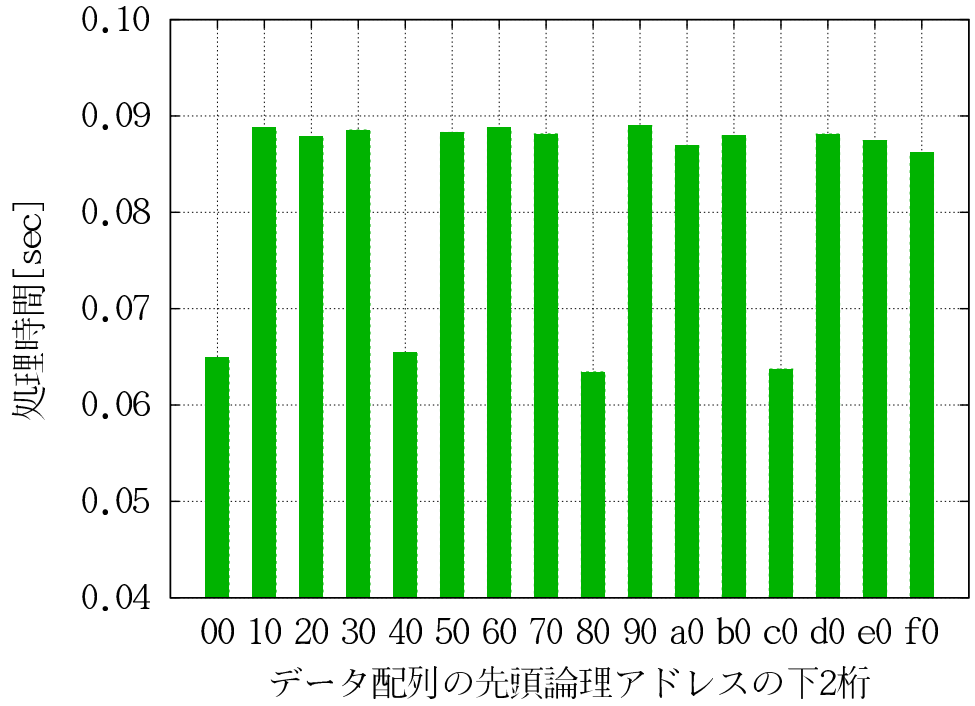
- キー計算ステージで 19〜23% 高速化
- データ移動ステージで 19〜55% 高速化

データ転送の高速化：データ アライン メント（説明）

- 入力データの先頭アドレスをキャッシュラインサイズの整数倍にアラインすることが重要
  - → アラインしない場合，1 個のベクトルが 2 個のキャッシュラインに分断されるためバストラフィックが増大



データ移動の高速化：データ アライン メント（結果）



- 入力データの先頭論理アドレスがキャッシュラインサイズの整数倍の場合のみ高速化
- このようなアラインは malloc 関数では保証されないため，明示的なアラインが必要

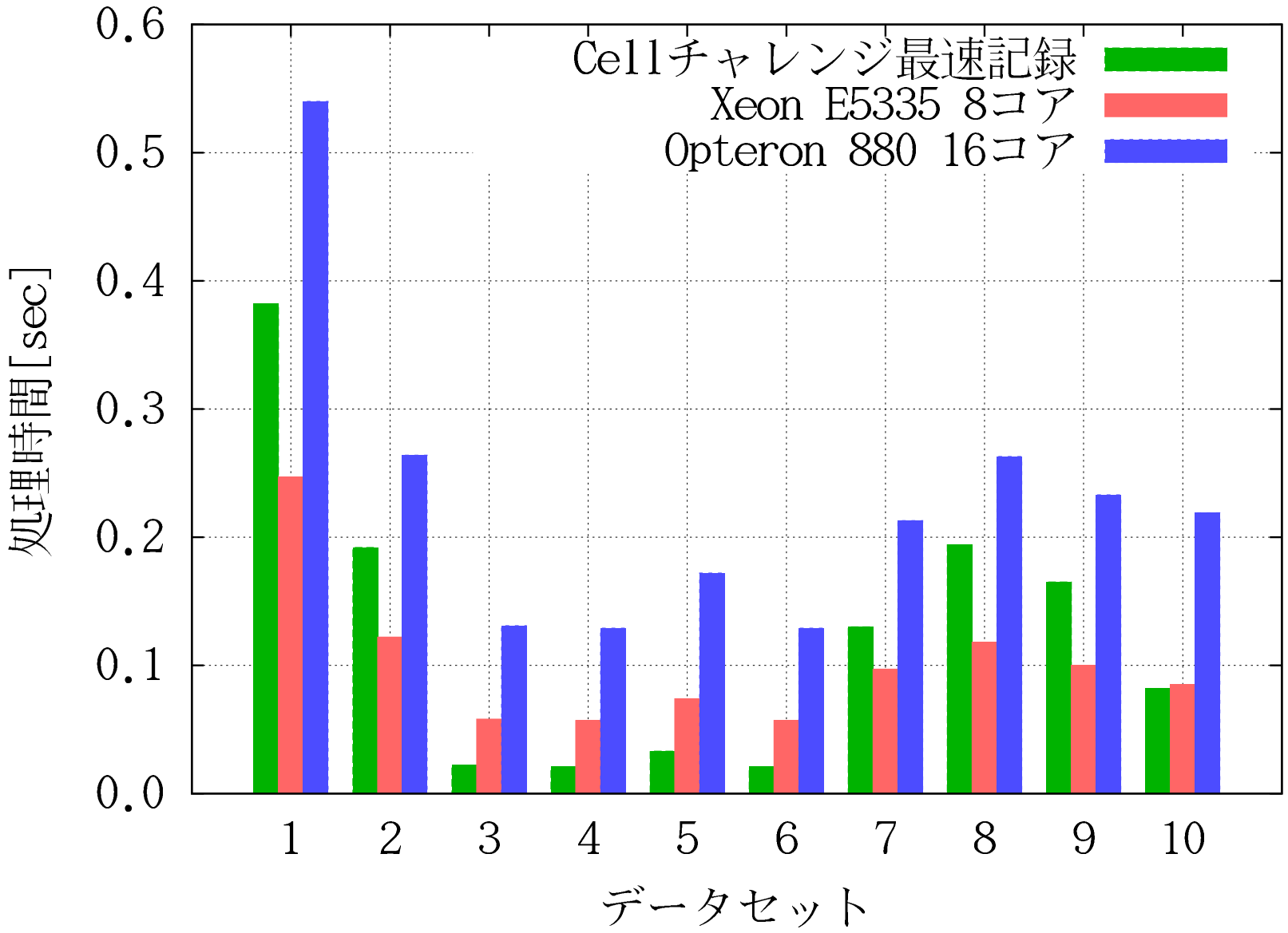
5. 結果と分析

- Xeon と Cell の結果を比較
- Xeon と Opteron の結果を比較

Cell vs Xeon vs Opteron（比較対象）

- 以下の 3 つを比較：
  - → Cell チャレンジ決勝の最速記録
  - → 本実験のコードを Xeon E5335（2.0 GHz,4 コア）を 2 基搭載した 8 コアマシン上で 8 スレッドで実行した記録
  - → 本実験のコードを Opteron 880（2.4 GHz,2 コア）を 8 基搭載した 16 コアマシン上で 16 スレッドで実行した記録

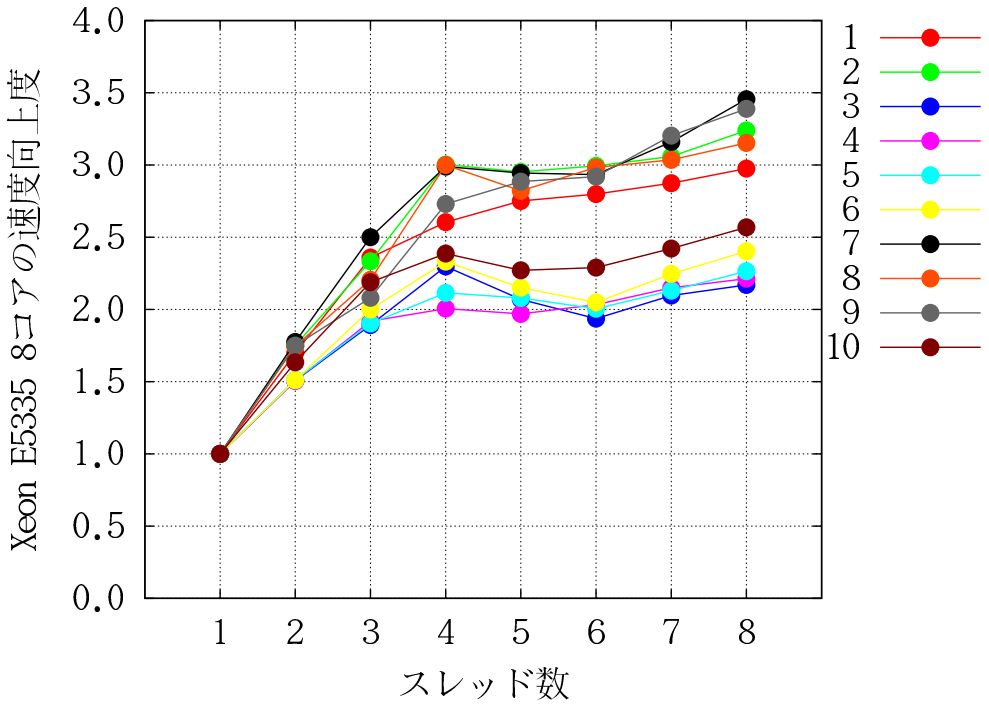
Cell vs Xeon vs Opteron（処理時間）

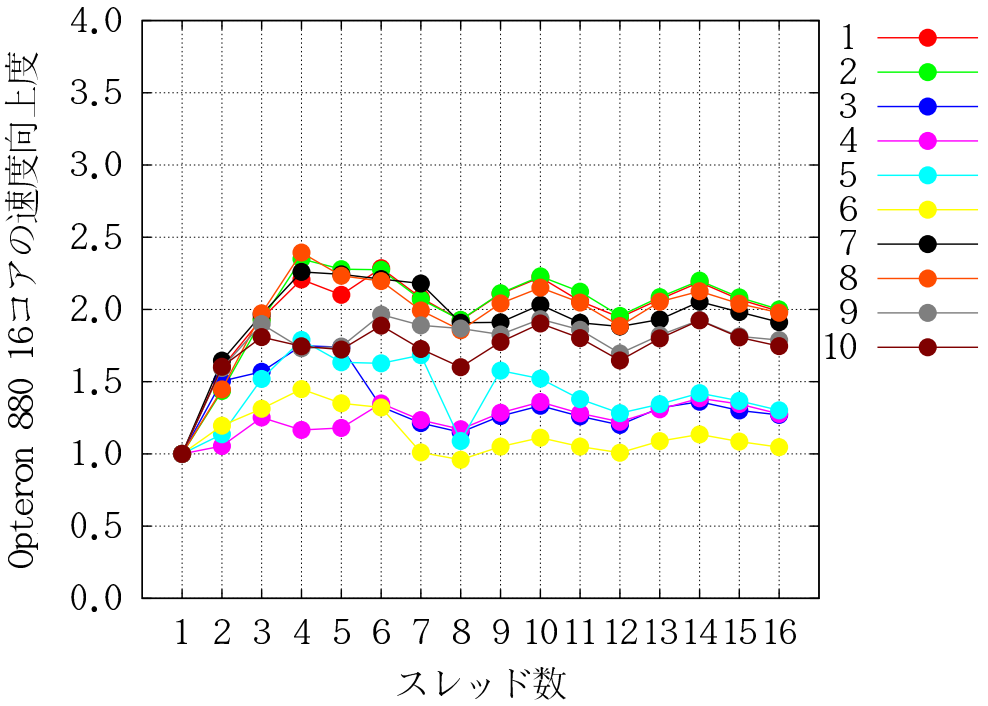


Xeon vs Cell（結果と分析）

- 結果：
  - → ベクトルの成分数の小さいデータセット（1,2,7,8,9）では，Xeon が Cell を 25〜39% 上回った
  - → ベクトルの成分数の大きいデータセット（3,4,5,6）では，Xeon は Cell より最大 2.7 倍遅かった
  - → 全データセットを通じて，Xeon の最長記録は最短記録の 4.3 倍だが，Cell の最長記録は最短記録の 18.2 倍
- 分析：
  - → Xeon で Cell と同程度の記録が出せた
  - →「Cell では，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が，成分数が小さいと DMA 転送の効果が出にくい」という傾向が現れた

Xeon vs Opteron（スケーラビリティ）





Xeon vs Opteron（結果と分析）

- **結果：**
  - → Opteron が Xeon の 2.2〜2.6 倍の処理時間を要した
  - → Opteron の方がスケーラビリティが悪い
- **分析：**
  - → Xeon：システムバス方式の SMP
  - → Opteron：各プロセッサにローカルメモリを設ける NUMA を採用した，単純なシステムバス方式よりも「進んだ」アーキテクチャ
  - → それなのに Opteron の方が性能が悪かった理由：
    Opteron ではローカルメモリへのアクセスは高速だが，本実験ではリモートメモリへのアクセス遅延が大きく響いたため

6. メモリアクセス最適化支援ツール

- Tulip の紹介
- Tulip の最適値探索アルゴリズム
- Tulip の利用例
- Tulip による最適化効果の検証

Tulip の仕様（説明）

- Tulip：
  - → 以上で述べた高速化手法のノウハウをもとに開発した，汎用プロセッサ向けメモリアクセス最適化支援ツール
  - → SIMD 化したコードがメモリバンド幅にほぼ律速されるようなプログラムに関して，メモリアクセスを最適化する方法を提案

Tulip の仕様（入力と出力）

- 入力：
  - → CPU のコア数
  - → メモリアクセスパターン
- 出力：ターゲットマシンで実効バンド幅を最大化するための
  - → スレッド本数
  - → 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
  - → チューニングの経過データ

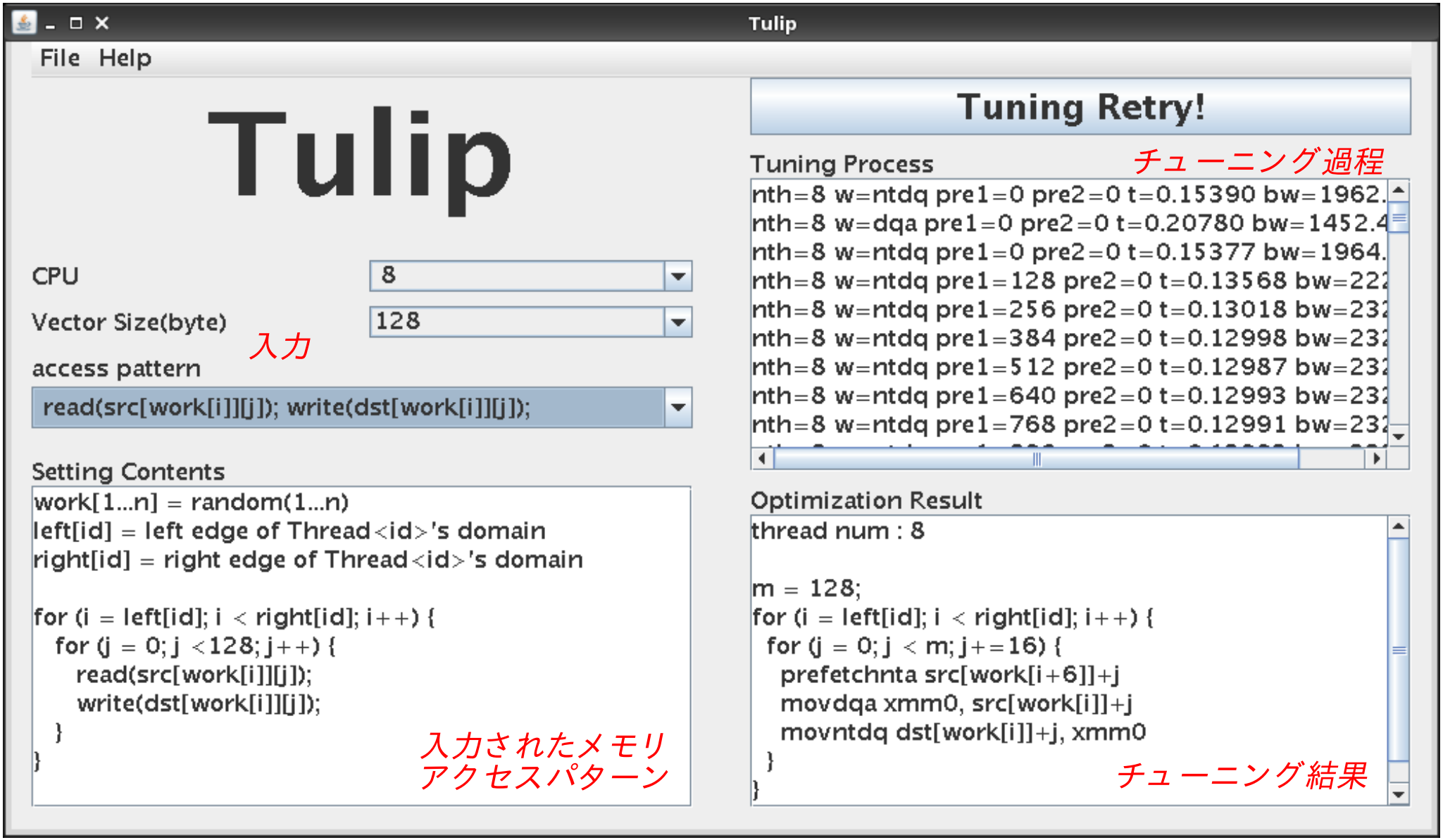
Tulip が対応するメモリアクセスパターン

- 2 次元配列からリードして別の 2 次元配列へライトするモデル

```
for ( i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
        read ( src [f (i)][ j ]) ;
        write ( dst [g (i)][ j ]) ;
    }
}
```

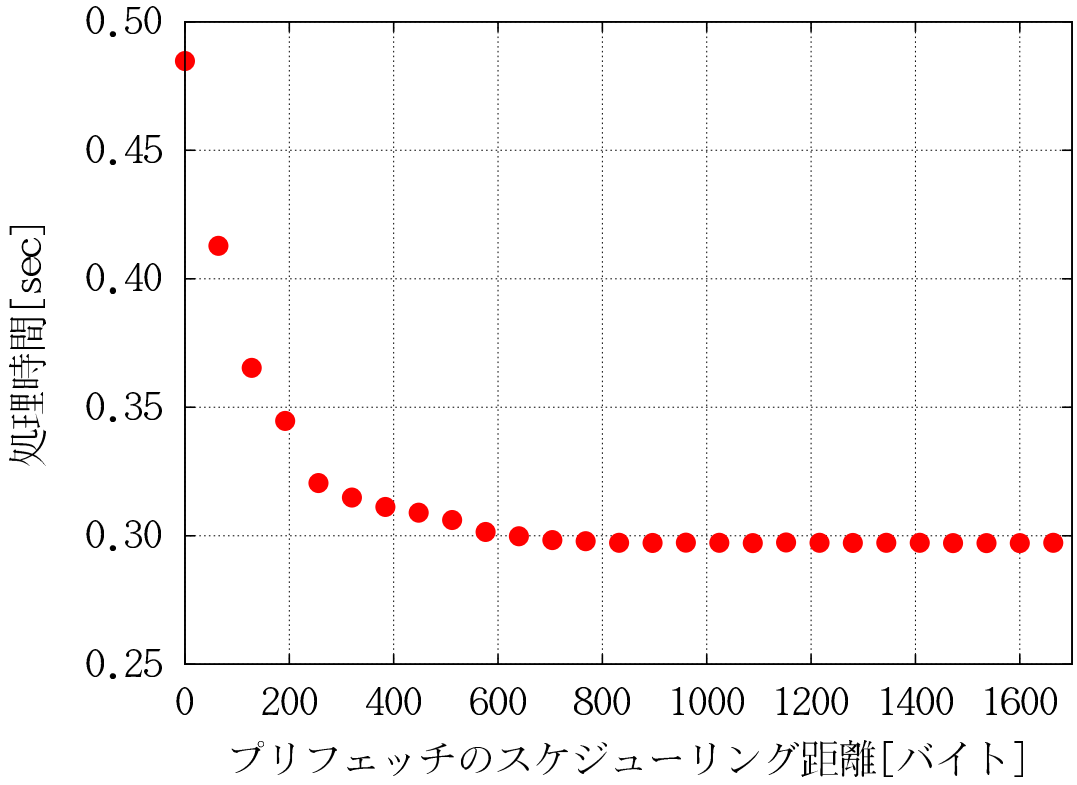
設定項目：(1) n はいくつか？
(2) f (i) = i か，それとも f (i) = random (i)か？
(3) g (i) = i か，それとも g(i) = random (i)か？

Tulip の外観


Tulip
File Help
Tulip
Tuning Retry!
Tuning Process
チューニング過程
nth=8 w=ntdq pre1=0 pre2=0 t=0.15390 bw=1962.
nth=8 w=dqa pre1=0 pre2=0 t=0.20780 bw=1452.4
nth=8 w=ntdq pre1=0 pre2=0 t=0.15377 bw=1964.
nth=8 w=ntdq pre1=128 pre2=0 t=0.13568 bw=222
nth=8 w=ntdq pre1=256 pre2=0 t=0.13018 bw=232
nth=8 w=ntdq pre1=384 pre2=0 t=0.12998 bw=232
nth=8 w=ntdq pre1=512 pre2=0 t=0.12987 bw=232
nth=8 w=ntdq pre1=640 pre2=0 t=0.12993 bw=232
nth=8 w=ntdq pre1=768 pre2=0 t=0.12991 bw=232
CPU
8
Vector Size(byte)
128
入力
access pattern
read(src[work[i]][j]); write(dst[work[i]][j]);
Setting Contents
work[1...n] = random(1...n)
left[id] = left edge of Thread<id>'s domain
right[id] = right edge of Thread<id>'s domain
for (i = left[id]; i < right[id]; i++) {
for (j = 0; j <128; j++) {
read(src[work[i]][j]);
write(dst[work[i]][j]);
}
}
入力されたメモリ
アクセスパターン
Optimization Result
thread num : 8
m = 128;
for (i = left[id]; i < right[id]; i++) {
for (j = 0; j < m; j+=16) {
prefetchnta src[work[i+6]]+j
movdqa xmm0, src[work[i]]+j
movntdq dst[work[i]]+j, xmm0
}
}
チューニング結果

Tulip の最適値探索：プリフェッチの性質

- 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には飽和特性あり
  - → この特性は様々なマシンや多少ランダムなアクセスに関しても成立



Tulip の最適値探索：アルゴリズム

- アイディア：
  - → コア数以下の各スレッドの本数に対して，ストリーミングストアを利用する or 利用しないの各場合について，プリフェッチのスケジューリングが飽和する際の処理時間を記録
  - → 処理時間が最小になるときの，[スレッドの本数，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] の組合せを出力
- 工夫：
  - → 探索する順番を工夫
  - → 最適値に達しないと予測される探索は早期に枝刈り

Tulip の利用例：高速化の余地があるかの判定

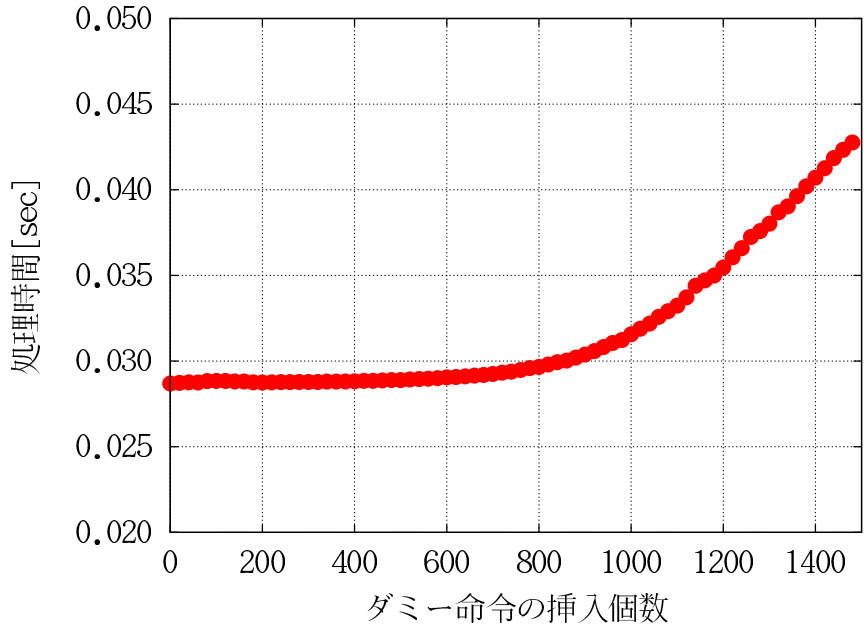
- Tulip の最適化を利用して，バンド幅にほぼ律速されるプログラムにまだ高速化の余地があるか否かを判定可能
- 手順：
  (1) プログラムに Tulip による最適化結果を反映
  (2) 最内殻の for ループ内にダミー命令（「意味のない」命令）を挿入
  (3) ダミー命令の挿入個数と処理時間の関係をプロット

```
for( i=0;  i<m;  i++) {
    for( j=0;  j<n;  j++) {
        …… 処理 ……
        cmp eax, ebx
        ⋮
        cmp eax, ebx
    }
}
```

ダミー命令を
挿入していく

Tulip の利用例：高速化の余地があるかの判定

- 例：データセット 4 のデータ移動ステージに関するプロット



- この処理はバンド幅に律速されている
  - → よって，SIMD などによる演算処理の高速化は無意味と判断可能
- 本実験ではこの評価法を用いて，Xeon E5335 上のソーティングのこれ以上の高速化は困難だと判断

Tulip の効果検証：検証する汎用プロセッサ

(1) Xeon E5335（2.0 GHz,4 コア）× 2

(2) Xeon 5140（2.33 GHz,4 コア）

(3) Core2Duo E6400（2.13 GHz,2 コア）

(4) Pentium M（1.7 GHz,1 コア）

(5) Celeron D（2.93 GHz,1 コア）

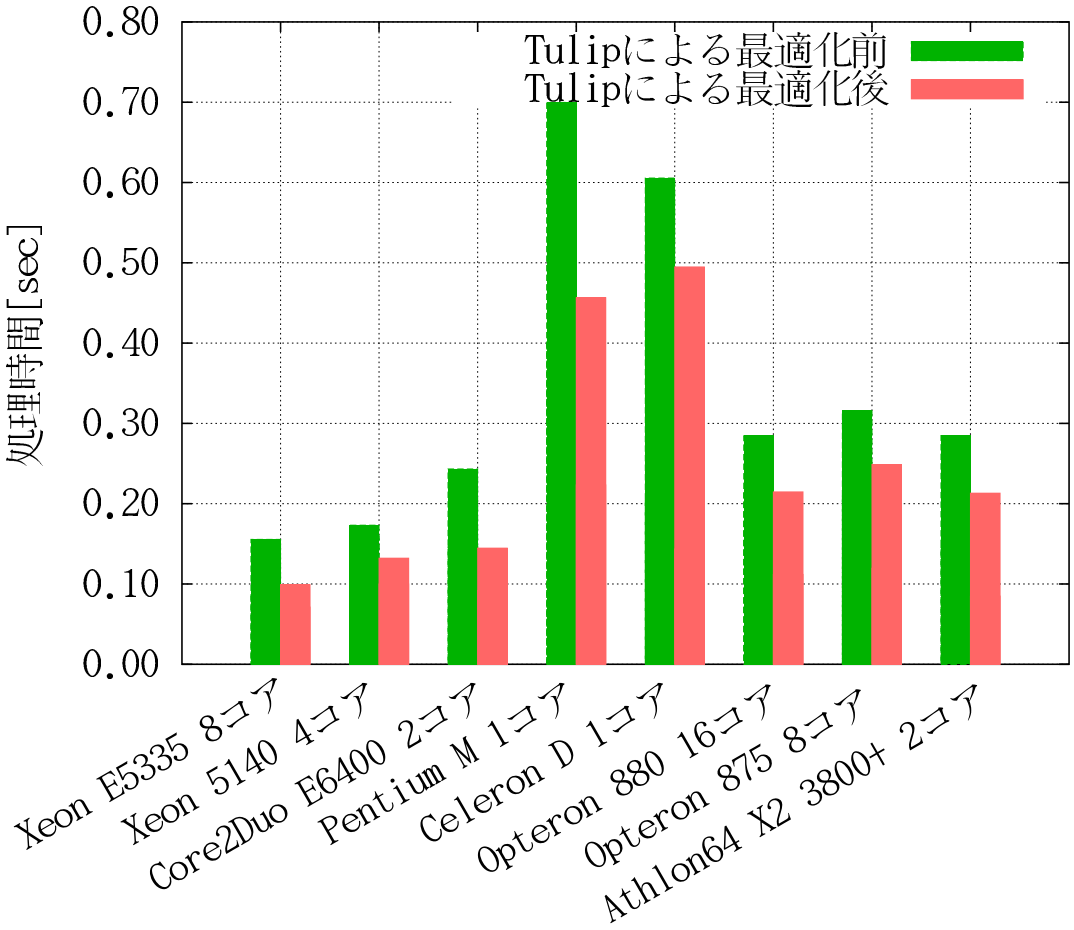
(6) Opteron 880（2.4 GHz,2 コア）× 8

(7) Opteron 875（2.2 GHz,2 コア）× 4

(8) Athlon64 X2 3800+（2.0 GHz,2 コア）

Tulip の効果検証：ソーティング

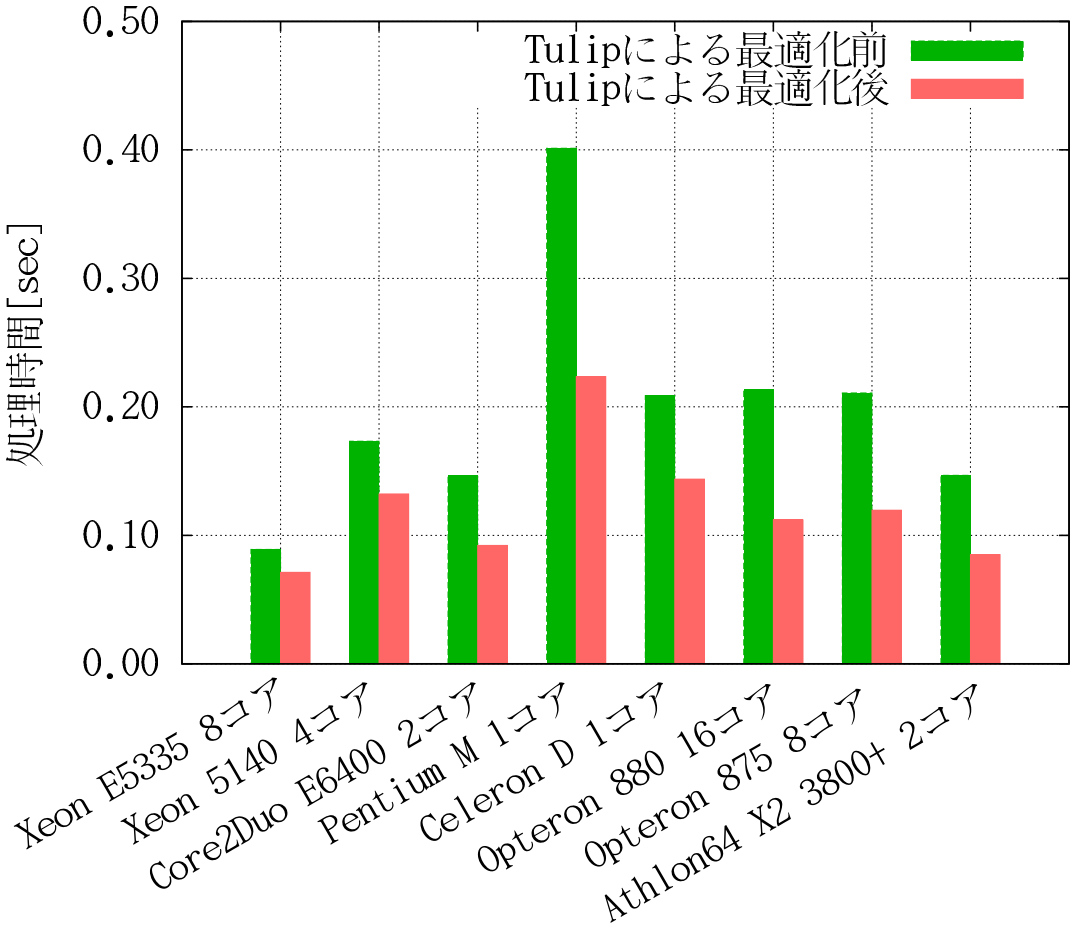
- 処理：データセット 7 に関するソーティング



- Tulip の最適化結果に従うことで 18〜40% 高速化

Tulip の効果検証：ストリーミングの簡易モデル

- **処理：320 MB の float 型配列 src[] の各要素の値を 2 倍にして配列 dst[] に格納する処理**



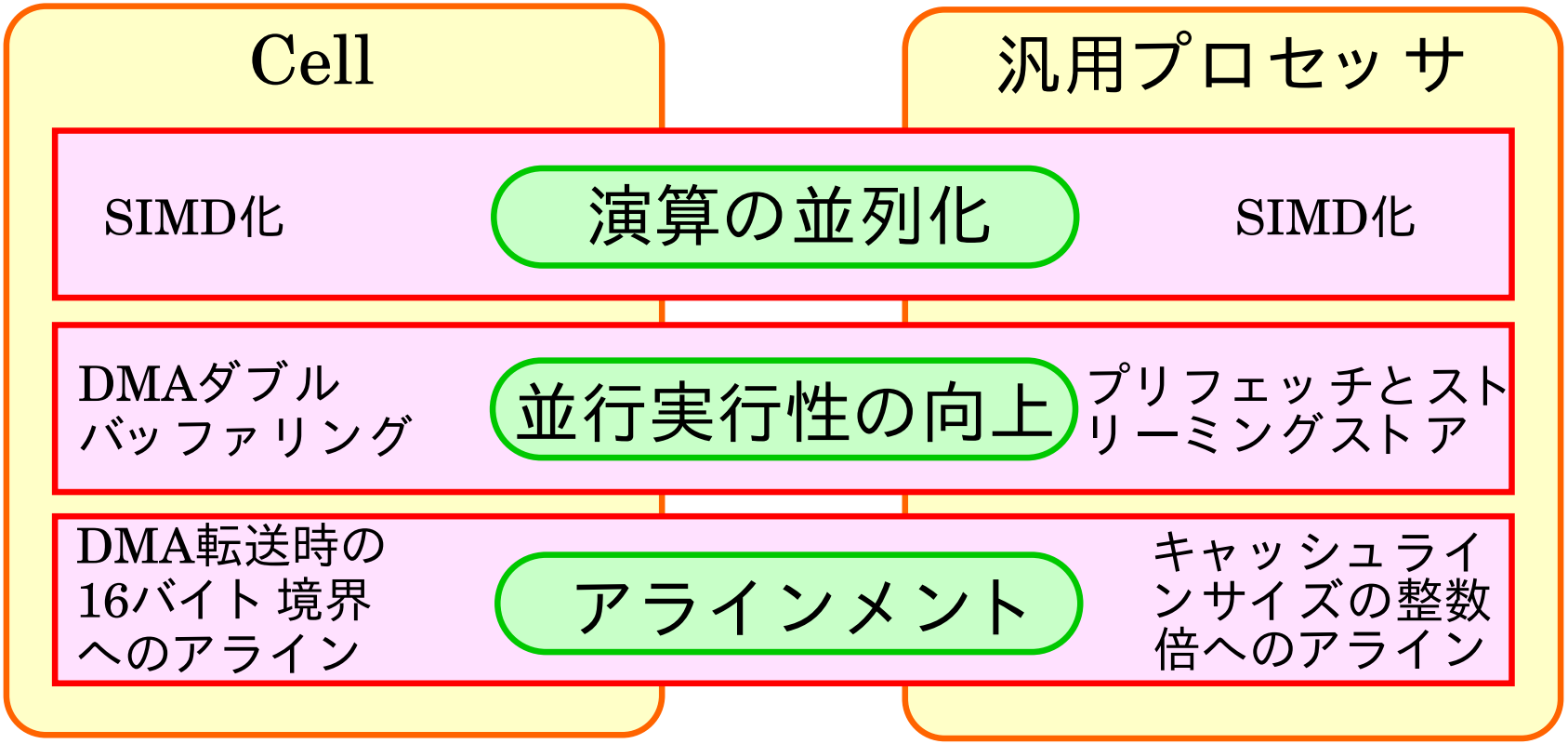
- **Tulip の最適化結果に従うことで 20〜48% 高速化**

7. 結論

- Cell と汎用マルチプロセッサの類似点・相違点

Cell と汎用マルチプロセッサの類似点

- Cell でも汎用プロセッサでも高速化のための方向性は同じ
- 本実験のソーティングに関しては汎用プロセッサで Cell と同程度の記録が出せた



Cell と汎用マルチプロセッサの相違点

- 違うのは，「高速化のための難しさが，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」