MMXテクノロジによるCAシュミレータの高速化

MMXテクノロジによるCAシミュレータの高速化

赤嶺 有平*，遠藤 聡志**，山田 孝治**

Acceleration of Cellular Automata Simulator Using MMX Technology

Yuhei AKAMINE*, Satoshi ENDO** and Koji YAMADA**

Abstract

Cellular Automata (CA) are inherently suited for parallel processing, and have been characterized as easy to parallelize. Therefore their simulation has prospects of speeding up using SIMD (Single Instruction stream, Multiple Data stream). Furthermore a low-cost CPU has been had SIMD such as MMX technology. MMX technology is a kind of SIMD what permits one instruction cycle to act on multiple data pieces and is SIMD what may be one of the most famous technologies. In this paper, we propose a method of high-speed CA simulation using MMX technology without dedicated purpose hardware. The results of simulations represent that our method is better with 7 times than scalar arithmetic.

Key Words: MMX, SIMD, CA

1. はじめに

1950年代に，S. Ulamと J. von Neumannによって提案されたセルラオートマトン法（Cellular Automata, 以下CAと記す）は，簡単なセル間の局所的相互作用から，複雑な現象を再現できる手法である．その後の研究により，生態（種の増減，種の住み分け，捕食関係など），反応・拡散現象（生物の紋様形成，化学反応など），フラクタル自然現象（結晶成長，凝集など），災害（森林火災，地震など），交通（高速道路の車の流れ）といった，様々な分野に適応できることが分っている [加藤 98].

CAは本質的に高度なデータ並列性を持ち，並列処理に向いているため，CA専用計算機の開発 [Margolus 93] や，専用記述言語からCAシミュレータを自動生成するソフトウエアの開発 [Beck]，また，ネットワークで相互接続された複数の計算機で，CAを高速にシミュレートするソフトウエアの開発が行われている [平林 98]．これらのシステムは，それぞれ実装レベルの違いはあるが，SIMD[1] の概念を用いて高速化を行っている．SIMDは，一つの命令の流れが複数のデータの流れを処理する並列アーキテクチャの一種である [Patterson 96]．CAは全てのセルの更新処理を同時に行う必要があるため，SIMDを用いることで効率的に処理できる．

従来，SIMDをハードウエアで扱うためには，スーパーコンピュータや専用プロセッサが必要であったため，コストが高くなる傾向にあった．ところが，近年の半導体技術の向上により，パーソナルコンピュータ用の廉価なCPUにもSIMD型並列演算命令の実装が一般的になってきた．したがって，これらの技術を用いることで，ローコストで高速なCAシミュレータの開発が可能である．

MMXテクノロジは，MMXテクノロジPentiumプロセッサ，Celeronプロセッサ，PentiumIIプロセッサ，PentiumIIIプロセッサに搭載されたSIMD型演算命令セットである [インテル 99a][Int 00]．これらのプロセッサは一般的なパーソナルコンピュータに採用されているため，普及率が非常に高くかつローコストである．また，プロセッサを特定することにより，そのプロセッサに特有の最適化も可能である．MMXテクノロジによる並列化と，このような最適化技術を組み合わせて，ハードウエアのもつポテンシャルを最大限に引き出すことができる．

本研究の目的は，汎用プロセッサ用のSIMD技術であるMMXテクノロジを用いて，高速かつ低コストなCAシミュレータの開発を行うことである．本論文においては，例題としてライフゲームのシミュレータをMMXテクノロジを用いて作成する際の高速化手法を述べる．

2. MMXテクノロジとSIMD

2.1 概要

MMXテクノロジは，マルチメディアの処理を高速化するために開発された技術である．マルチメディアの処理では，大量のデータに対する単純な操作の繰り返しがその処理時間の大半を占めることが多いため，複数のデータをまとめて処理できれば効率的である．MMXテクノロジは，SIMDの概念を用いることで演算を並列化し，処理の高速化を図っている [小鷲 97].

受理：2000年6月5日
*大学院理工学研究科情報工学専攻
(Master Couse in Complex Intelligent Systems Engineering, Graduate School of Science and Engineering)
**工学部情報工学科
(Dept. of Information Engineering, Fac. of Eng.)
[1] Single Instruction stream, Multiple Data stream

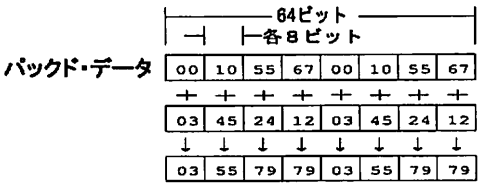

Fig. 1. MMX のパックド・データ演算の例

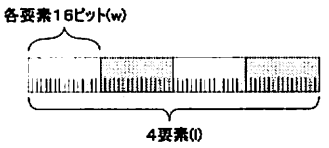

Fig. 2. 4 × 16 パックド・データ

SIMD とは，1 つの命令の流れが複数のデータの流れを並行して処理する機構である [Patterson 96]．MMX テクノロジは，64 ビットの演算器を命令単位で分割して，8 組の 8 ビット演算器，4 組の 16 ビット演算器，2 組の 32 ビット演算器，1 組の 64 ビット演算器として動作させることが出来る [小鷲 97]．そのため，たとえば 8 組の 8 ビット演算器として動作させた場合，逐次演算に対して 8 倍の性能向上の可能性がある．

2.2 パックド・データ演算

パックド・データ演算は，MMX テクノロジの中核となる機能であり，SIMD 機構に基づく演算である．パックド・データ演算では，64 ビットの演算器を分割して複数の演算器の集合として動作させる．演算を行う際は，複数の 8 ビット，16 ビット，あるいは 32 ビットのデータをパックド・データレジスタと呼ばれる 64 ビットのレジスタにパック化 (packing) する．パックド・データ演算は，このパック化されたデータに演算を施すことを意味する (1)．

定義 *1:* パックド・データ $\boldsymbol{v}$ が，l 個の w ビットデータの集合であるとき，パックド・データ v の i 番目の要素とは，iw ビット目から $(i+1)w-1$ ビット目までの部分的なビット列で表される値である．

パックド・データ演算は，パックド・データ $\boldsymbol{u}, \boldsymbol{v}$ の各要素 u_i と v_i に対して独立した演算を施す．本論文では，l 個の w ビットデータからなるパックド・データを，$l \times w$ パックド・データと表現する．例えば，4 つの 16 ビットのデータをパック化したデータは 4×16 パックド・データである (2)．

2.3 比較演算

比較演算は，パックド・データ演算において条件判断を並列に行う命令である．

定義 *2:* $l \times w$ パックド・データ $\boldsymbol{a}, \boldsymbol{b}$ に対して条件を “等しい” とする比較演算を適用したとき，結果 $\boldsymbol{c}$ は以下のように定義される (3)．

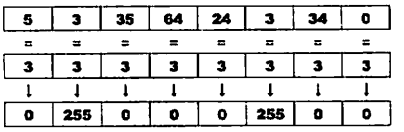

Fig. 3. 比較演算の例

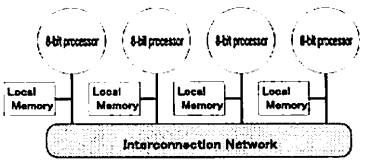

Fig. 4. 分散メモリ型 SIMD マシン

$$c_i = \begin{cases} 2^w & (a_i = b_i \text{ の時}) \\ 0 & (a_i \neq b_i \text{ の時}) \end{cases} \quad (1)$$

ただし，$i = \{0, 1, \cdots, l-1\}$

条件としては，“等しい (=)”，“より大きい (>)” の 2 つを指定できる．

本論文では，式 1 を

$$\boldsymbol{c} = \mathrm{comp}\,(\boldsymbol{a} = \boldsymbol{b})$$

と表記する．カッコ内の等号は条件を表す．

2.4 データ分割法

2.2 節において，パックド・データ演算では各データをパックド・データレジスタにパック化する必要があることを述べた．データ分割法 [天野 89] は，分散メモリ型並列計算機の概念だが，MMX テクノロジのパック化に対しても有効である．

パックド・データ演算は，単一の演算命令がパックド・データの各要素に作用する．そのため，8×8 パックド・データ演算を行う時，MMX テクノロジの演算器は，8 個の 8 ビットプロセッサエレメント (以下 PE と記す) からなる分散メモリ型 SIMD マシン [Patterson 96] とみなせる (4)．各 PE は，8 ビットのレジスタと演算器を持ち，並列に動作する．

MMX テクノロジにおいて，パックド・データレジスタとメモリ間の 8 バイト境界をまたぐアクセスは，大きなペナルティ (プロセッサ・ストール) を伴う [インテル 99c] ため，メモリアクセスの際は位置合わせ (alignment) を行う (5)．8 個の 8 ビット PE からなる SIMD マシンの観点からは，メモリ空間を 8 つに分割し，それぞれのメモリ空間を各 PE のローカルメモリとみなすことが出来る (6)．メモリアドレスは，全ての PE に同時に指定され，アクセスも同時に起こる．パックド・データ演算は，PE のローカル演算であり，パックド・データレジスタ全体に対するシフト演算は，PE 間の通信である．

PE は，リモートメモリへのアクセスには，余分なコストがかかる．従って，演算に必要なデータをローカルメモリに配置することが処理を効率化する．

2.5 データ構造の最適化

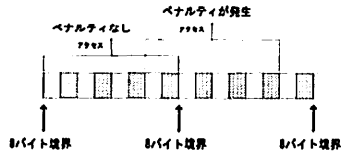

Fig. 5. データ・アラインメント

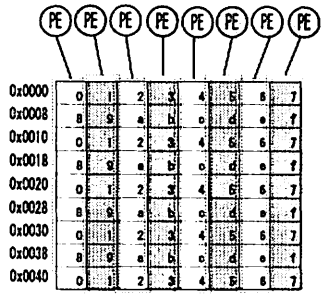

Fig. 6. ローカルメモリ

分散メモリ型SIMDマシンでは，出来るだけPE間通信が少なくなるようにデータを分割して各PEに割り当てる [天野 89]．パックド・データ演算においては，データのメモリ上の配置を並び替えることにより処理を効率化する．例えば，$n \times 8$ 個の要素を持つ配列 a[i] を 8×8 パックド・データ演算で隣接する要素間の演算を効率よく行うために配列 b[i] に並び替える操作は，以下の様に表現できる (7)．

```
for(i=0;i<8;i++) {
  for(j=0;j<n;j++) {
    b[j*8+i] = a[i*n+j];
  }
}
```

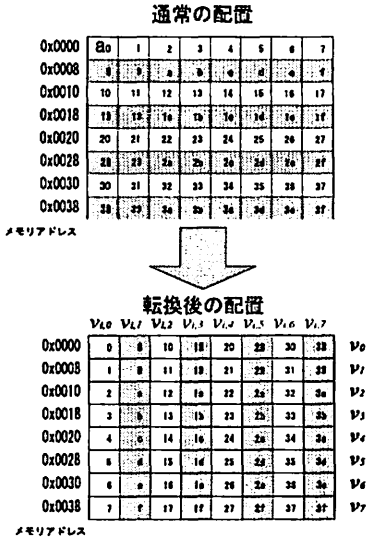

Fig. 7. 隣接する要素間の演算を効率よく行う配列（v_i はパックド・データ）

この配置法は，配列の隣接する要素が同じローカルメモリに存在するため，プロセッサ間の通信，つまり，パックド・データ演算におけるシフト演算を不要にし，演算を大幅に効率化している．

3. 提案手法

提案手法は，CAシミュレータにおけるセル状態の更新処理をMMXテクノロジを用いて並列化し，プロセッサ特有の最適化を行うことにより，シミューションを高速化するものである．対象とするプロセッサは，MMXテクノロジを搭載したPentiumプロセッサである．

提案手法が対象とするCAは，ライフゲームに代表される半合計的なCAである．この種のCAは，セルの新しい状態を隣接するセルの状態と自身の状態とのルールセットに基づいて決定する [Ladd 95]．本論文では，半合計的なCAのセル状態を更新する処理における，近傍のセルの状態値の合計を求める部分を“状態和の算出”，また求めた状態和の値にルールを適用して新しい状態を決定する処理を“次状態の決定”と表現する．

提案手法では，パックド・データ演算を用いて状態和の算出と次状態を決定を行うが，この処理を効率的に行うためにデータ構造を最適化する．に示したように，パック化の際の配置を工夫する事で，余分なデータの移動をなくすことができる．この配置の転換は最初に一度行えばよいので，時間的なコストは無視できる．

本論文では，説明のためライフゲーム・シミュレータに対する具体的な最適化手法を述べる．ライフゲームは，状態和の算出，次状態の決定などの処理が一般的であり，セル間のルールが単純[2]なことから，例題として適当であると考えた．

提案手法の処理の概要を以下に示す．

step.1　パックド・データ演算を用いて近傍セルの状態和の算出を並列化する．

step.2　比較演算を用いて次状態の決定を並列化する．

3.1 状態和算出の並列化

まず，近傍のセルの状態値を合計する処理をパックド・データ演算で並列化する．

3.1.1 状態和算出の定義

ライフゲームのセル空間は2次元なので，$M \times N$ の2次元格子空間を考える．各セルは，0及び1の2つの状態をとりうる．格子空間上の横方向を x 軸，縦方向を y 軸として，セル座標を位置ベクトルであらわす．ただし，左上のセル座標を原点 $(0,0)$ として，右方向と下方向を正とする．$\boldsymbol{x} = (x, y)$ のとき，$s(\boldsymbol{x})$ は，左から $x+1$ 番目，上から $y+1$ 番目のセルの状態値を表す (8)．

ライフゲームにおける状態和の算出は，以下の様に記述される．

$$S(\boldsymbol{x}) = \sum_{i=0}^{7} s(\boldsymbol{x} + \boldsymbol{c}_i) \qquad (2)$$

[2]数ステップ程度の機械語で記述可能．

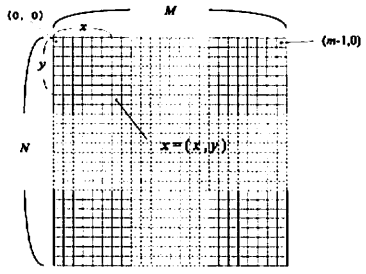

Fig. 8. $M \times N$ の 2 次元格子空間

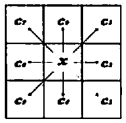

Fig. 9. 近傍 8 セル

ただし，$S(\boldsymbol{x})$ は格子空間上の位置 $\boldsymbol{x}$ における近傍 8 セルの状態和であり，$\boldsymbol{c}_i$ は，“i”方向の，各要素が “0”，“1”，“-1”のいずれかである位置ベクトルである．“i”方向とは，y 軸の負の方向を基準として，時計回りに $(i*45)$ 度回転した方向のことである (9)．

3.1.2 処理の並列化

式 2 の処理を複数のセルについて並列に行うため，格子空間を分割して各 PE [3] に処理を割り当てる．状態値は 1 ビット，状態和は 3 ビットのビット幅で表せるので，パックド・データで最も要素のビット幅が小さく，最も要素数の多い 8×8 パックド・データを用いる．この場合，8 つのデータを並列処理できるため，10 に示すとおり格子空間を 8 分割する．

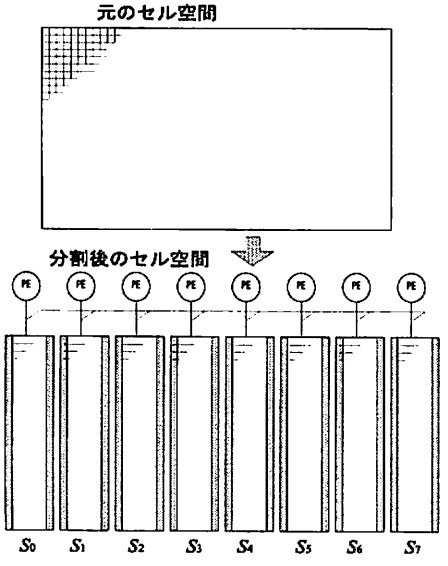

Fig. 10. 格子空間の分割（グレー部は PE 間通信が必要な部分）

[3] において，パックド・データ演算を分散メモリ型 SIMD マシンと置き換えて説明したが，本節においても円滑に説明を進めるために，パックド・データ演算を分散メモリ型 SIMD マシンとみなして話を進める．

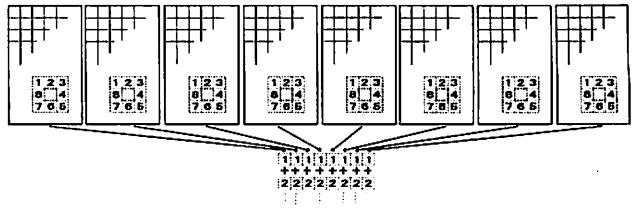

Fig. 11. 状態和算出の並列化

$$
\begin{array}{cccc}
s_0(0,0) & s_1(0,0) & \cdots & s_7(0,0) \quad \longleftarrow (a) \\
s_0(1,0) & s_1(1,0) & \cdots & s_7(1,0) \\
\vdots & \vdots & \ddots & \vdots \\
s_0(m,0) & s_1(m,0) & \cdots & s_7(m,0) \quad \longleftarrow (b) \\
s_0(0,1) & s_1(0,1) & \cdots & s_7(0,1) \quad \longleftarrow (c) \\
\vdots & \vdots & \ddots & \vdots \\
s_0(m,n) & s_1(m,n) & \cdots & s_7(m,n)
\end{array}
$$

ただし，

m, n : 各部分空間の最も大きい x, y 座標

Fig. 12. 状態値のメモリ上の配置

分割した格子空間のセルの状態値は，それぞれの PE のローカルメモリに分散して配置する．このとき，各ローカルメモリの状態値の並びを同じにする．各 PE のメモリアクセス時のアドレッシングは同時に行われるため，ある方向の隣接するセルの状態値のアドレスは，全てのローカルメモリにおいて，同じ方向の隣接するセルの状態値を指し示すことになる．したがって，あるセルに着目して状態和を算出する処理を行うと，ほかの 7 つの部分空間上のセルの状態和も同時に算出される (11)．

3.1.3 データ並び替え

SIMD マシンにおけるローカルメモリへの配置は，パックド・データ演算ではデータの並び方を工夫することによって行う．12 は，8 つの各部分空間を $s_0, s_1, \cdots, s_7$ として，$s_i(x, y)$ はそれぞれの，空間の左上のセルを原点とするローカル座標 x, y 上の状態値を表すとしたときの，状態値のメモリ上の配置を示している．図中の各行はパックド・データである．各列が PE のローカルメモリに相当する．

3.1.4 境界処理

一般に CA の格子空間は，トーラス構造であることが多い．そのため，図のように格子空間を 1 セル分大きくとって，境界のセルの状態値を反対側の余分にとった空間にコピーする (13)．こうすることで，境界セルに対して特別な処理を行う必要はなくなる．

さらに，この操作を行うことで各部分空間は連続的なものとして扱える．例えば，$s_0(m,0)$ の右隣の状態値は $s_1(0,0)$ でなくてはいけないが，12 の並びでは $s_0(m,0)$ の

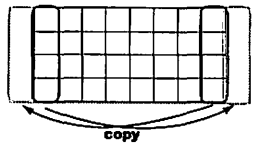

Fig. 13. 境界のセルの処理

右隣の状態値は $s_0(0,1)$ である（図中の $(b),(c)$）．

この問題を解決するために境界の処理は，単純なコピー操作だけではなくパックド・データのシフト演算も行う．各部分空間の状態値を保存するメモリ領域は1セル分大きくとってあるので，図中の (a) の前と $(b),(c)$ の間に1パックド・データ分の" 空き "がある．(a) のパックド・データを8ビット左に回転して $(b)(c)$ 間（右境界）の空きにコピーし，(b) のパックド・データを8ビット右に回転して (a) の前（左境界）の空きにコピーする．

これらのコピー操作は，セル状態が更新するたびに行う必要がある．しかしながら，この操作は，2回のコピーと1回の回転のみなので，全体の処理に占める割合は少ない．

3.2 次状態の決定

次に，近傍セルの状態和をもとに新しい状態を決定する処理の並列化を行う．新しい状態を決定する際の条件判断に，MMX テクノロジの比較演算を用いる．比較演算を用いた場合，通常のスカラ演算の条件分岐のように branch 命令を伴わないため，制御ハザード [中澤 95] が発生しない．制御ハザードが発生した場合[4]，PentiumIII プロセッサは 10〜15clock cycle 分ストール（停止）する [インテル 99c]．

3.2.1 次状態の決定の定義

本論文で扱う半合計的な CA では，新しい状態は，処理対象となるセル自身の状態値とその近傍8セルの状態和の組み合わせによって決定する．ライフゲームにおいては，以下のルールに従って新しい状態を決定する [加藤 98]．

- 生きているセルは，近傍セルの状態和が2または3の時のみ，生き続ける．それ以外は，死ぬ．
- 死んでいるセルは，近傍セルの状態和が3の時のみ，生き返る．それ以外は，変化しない．

これは，次のように言いかえることができる．

- 近傍セルの状態和が3の時は生きる．
- 近傍セルの状態和が2の時は変化しない．
- 上記以外の時は死ぬ．

これを，式で記述すると，

$$C_{t+1} = \begin{cases} 1 & (S_t = 3 \text{ のとき}) \\ C_t & (S_t = 2 \text{ のとき}) \\ 0 & (\text{それら以外のとき}) \end{cases} \quad (3)$$

[4] Pentium プロセッサには分岐予測機構があるため，必ずしも制御ハザードが発生するとは限らない．

ただし，C_t：時刻 t におけるセルの状態値
S_t：近傍のセルの状態和

3.2.2 比較演算の適用

状態和は，step.1 の処理が終了した時点で状態値のデータ構造とまったく同じような並びでパック化されている．S_t, C_t を時刻 t における状態和，状態値のパックド・データとすると，比較演算による次状態の決定は以下の様になる（14）．

$$c_2 = \mathrm{comp}(S_t = \mathbf{all3s}) \,\&\, \mathbf{all1s} \quad (4)$$

$$c_1 = \mathrm{comp}(S_t = \mathbf{all2s}) \,\&\, C_t \quad (5)$$

$$C_{t+1} = c_1 \mid c_2 \quad (6)$$

ただし，"&"，"|"はビット単位の論理積，論理和を表し，c_1, c_2 は 8×8 パックド・データである．また，**all(n)s** は，全ての要素が n である 8×8 パックド・データである．

式4は，式3の最初の条件に対応する式である．comp は，条件を満たす要素をその要素の最大値に設定する．最大値は全てのビットが1である値なので，必要な値と論理積を取ることで求る値になる．この式では，比較演算の結果と **all1s** との論理積を取ることで必要な値を求める．

式5は，式3の2番目の条件に対応する．ここでは，求めたい値は C_t なので，C_t との論理積をとる．

最後に，式6で式4と式5の結果の論理和をとることで処理が完了する．式3の各条件は，排他的であるため式4と式5は，少なくとも一方の結果が0になる．従って，論理和をとることは，条件の成り立つ方の式の結果をとることに等しい．

step.1 と step.2 の並列化のよって8つのセルの更新処理が同時に行われるため，逐次演算と比較して8分の1の処理時間ですむ可能性がある．

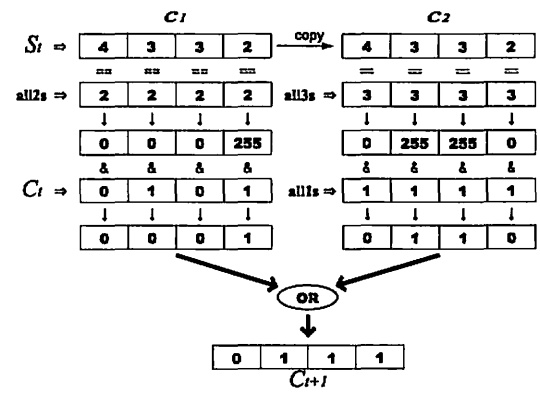

Fig. 14. 比較演算を用いた次状態の決定

4. 実験

4.1 実験環境

提案した手法の有効性を確認するために，評価用に作成したライフ・ゲームシミュレータを用いて実験をおこなった．実験は，提案手法の各ステップを段階的に適用した3

つのバージョンとスカラ演算バージョンの比較を行った．また，処理時間を厳密に計測するために，時間をプロセッサの clock cycle 数よって計測した．さらに，ビデオメモリへの転送処理によるオーバヘッドを除去するため，CA の更新部分の clock cycle 数のみを計測した．

実験環境は windows98 だが，同 OS は，マルチタスク OS なのでタスク・スイッチングによる結果のばらつきがでる．そのため，セル状態の更新処理を 100 回おこなうのに必要な clock cycle を計測して，その平均値をとった．実験に用いたプロセッサは，MMX テクノロジ Pentium プロセッサと PentiumIII プロセッサである．clock cycle の計測には，MMX テクノロジ Pentium プロセッサの Time Stamp Counter を用いた [小鷲 97]．

実験に用いたライフ・ゲームの格子空間は，256 × 256 である．この大きさは，実験用のシステムにおいて，状態値の全データがプロセッサ（CPU）の 1 次キャッシュには収まらないが，2 次キャッシュには入りきる大きさである．データの大きさが 2 次キャッシュのサイズを超えると，メモリアクセス速度のボトルネックが急激に増大[5] し，実験結果がシステムのバス速度に依存してしまうので，前述したサイズが適当である．実験は，以下のようなシミュレータをそれぞれ作成，比較した．

1. 評価基準となるシミュレータ（C 言語によるスカラ演算バージョン）
2. step.1 を適用したもの（状態和の算出のみに提案手法を適用）
3. step.1 に加えて step.2 を適用したもの（状態和の算出および次状態の決定に提案手法を適用）

C 言語によるスカラ演算バージョンは，VisualC++バージョン 6.0 のコンパイラで Pentium プロセッサを対象とした速度に対する最適化を行うオプションをつけてコンパイルした．このオプションをつけた場合，コンパイラは MMX テクノロジを用いないで，superscalar 実行のための pairing などの，速度を高速にするための最適化を行う．

4.2 実験結果

前述した，4 種類のシミュレータを MMX テクノロジ Pentium プロセッサ，PentiumIII プロセッサの各プロセッサ上で動かして，その実行 clock cycle を測定し，それぞれの値を C 言語によるスカラ演算バージョンのものと比較を行った．実験結果をグラフに示す（15）．

実験結果から，MMX テクノロジ Pentium プロセッサ上で走らせた場合，C 言語によるスカラ演算バージョンを基準として，状態和の算出のみに提案手法を適用したものが約 3 倍，状態和の算出および次状態の決定に提案手法を適用したものが約 5.5 倍の速度で処理が終了することがわかった．また，PentiumIII プロセッサ上で走らせた場合は，それぞれ，約 3 倍，約 7.5 倍の速度で処理できることがわかった．

5. 考察

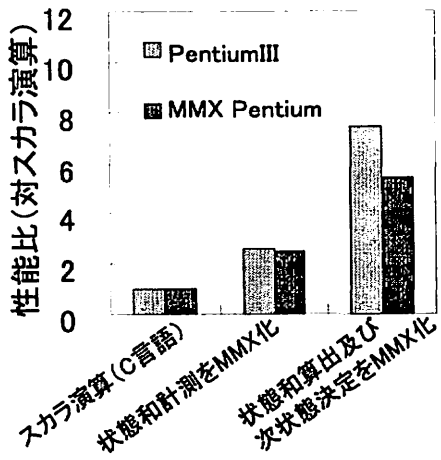

Fig. 15. 実験結果

MMX テクノロジのパックド・データ演算は，8×8 パックド・データを用いた場合，最大で 8 倍の高速化の可能性がある．しかしながら，実際にはループ制御やその他のスカラ演算が必要であるため理想値には達しない．

提案手法では，このような並列化を阻害する要因をデータ構造とコーディングの工夫によって排除している．提案手法の step.1 では，データ構造の最適化により余分なデータ移動を低減することで，演算の並列度を高めている．step.2 では，比較演算を用いることで制御ハザードを無くし，結果として並列度が上がっている．

実験結果は，提案手法がより複雑な CA モデルにも適用可能であることを示している．例えば，step.1 のみを適用した場合，評価基準である C 言語バージョンに比べて明らかに高速である．これは，状態和の算出にはパックド・データ演算が適用できるが，次状態の算出には適用が難しい場合，つまり，次状態の決定のルールが複雑すぎて並列化できない場合などでも，ある程度は効果があることを示している．

6. おわりに

本論文では，MMX テクノロジのパックド・データ演算，比較演算による並列演算により，ローコストで高速なセルラ・オートマトン・シミュレータを作成する手法を提案した．

提案した手法を用いてライフゲームのシミュレータを作成し，スカラ演算によるものとの比較を行ったところ，良好な結果を得た．

提案手法は，理論的にはライフゲームに限らず一般的な CA に対して適用可能である．今後，様々な CA モデルに対して提案手法を適用し，その効果を検証する必要がある．

[5] 一般的な PC アーキテクチャのシステムでは，2 次キャッシュのミスヒット時のペナルティは，数十から百数 clock cycle である．

[Beck] Beck, M.: The Cellular Automata Simulation System: http://www.cs.runet.edu/~dana/ca/cellular.html.

[Beck 94] Beck, M. and Castellanos, A.: Vector Processing on Scalar Architecture (1994).

[Int 00] Intel Celeron Processor - Datasheet,Intel Celeron Processor up to 533MHz(Order Number:243658-010) (2000).

[Ladd 95] Ladd, S. R.: C++シミュレーションズ&セルラ・オートマトン日本語版, 株式会社ディー・アート (1995), 朝沼　美雪：訳.

[Margolus 93] Margolus, N.: CAM-8:a computer architecture based on celluar automata, Physiscs of computation seminar (1993).

[Patterson 96] Patterson, D. A. and Hennessy, J. L.: *Computer Architecture A Quantitative Approach*, Morgan Kaufmann Publishers,Inc., second edition (1996).

[インテル 99a] インテル株式会社：インテル・アーキテクチャソフトウェア・ディベロッパーズ・マニュアル下巻：システム・プログラミング・ガイド (資料番号 243192J) (1999).

[インテル 99b] インテル株式会社：インテル・アーキテクチャソフトウェア・ディベロッパーズ・マニュアル上巻：基本アーキテクチャ(資料番号 243190J) (1999).

[インテル 99c] インテル株式会社：インテル・アーキテクチャ最適化 (資料番号 730795J-001) (1999).

[加藤 98] 加藤, 光成, 築山：セルオートマトン法, 森北出版 (1998).

[小鷲 97] 小鷲英一：MMX テクノロジ最適化テクニック, アスキー出版 (1997).

[中澤 95] 中澤喜三郎：計算機アーキテクチャと構成方式, 朝倉書店 (1995).

[天野 89] 天野, 高橋, 富田, 渡辺, 渡辺：並列処理機構, 丸善株式会社 (1989).

[平林 98] 平林, 石塚, 横田, 伊藤, 小前, 渦原, 竹岡, 安村 通晃：セルラオートマトン・シミュレータ用並列コンパイラの開発, 情報処理振興事業協会 「創造的ソフトウェア育成事業」最終成果発表会 (1998).