FPGAを用いた格子ガスオートマトンの高速計算

小堀友義[†] 丸山勉[††] 星野力[††]

格子ガスオートマトン法はセルラオートマトン法の一種で，主に流体力学のシミュレーションを行う際に用いられる．格子ガスオートマトン法において，流体は離散的な疑似粒子の集合として表され，単純な遷移則により格子面上の各格子の状態をいっせいに更新することで格子の次の世代の状態を得ることができる．原理的に高い並列性を持つゆえ，これまで多くの並列計算システムの研究対象とされてきた．本論文では，メモリバンド幅が制限されているような小規模システムでも高速化が可能な格子ガスオートマトン法の高速計算方式を提案する．本高速計算方式を，Xilinx社製Field Programmable Gate ArrayであるVirtex（XCV1000）を1チップ搭載し，外部メモリとして総容量8 MByteのメモリを持つPCIボード（ADC RC1000）上に実装し性能評価を行った．その結果，格子ガスオートマトン法の一種であるFHP-IIIモデル 2048×1024 格子において，マイクロプロセッサ（Pentium-III 700 MHz）に比べ，約140倍という大規模並列計算システムと同程度の速度向上が得られた．

High Speed Computation of Lattice Gas Automata with FPGA

Tomoyoshi Kobori,[†] Tsutomu Maruyama[††] and Tsutomu Hoshino[††]

Lattice gas automata are a class of cellular automata, and are used for simulating fluid dynamics. In the cellular automata, a single update rule is applied simultaneously to each cell on the lattice. Therefore, many approaches with parallel systems have been researched. In this paper, we propose a computation method of cellular automata for small systems with limited memory bandwidth. We implemented the method on a FPGA board (ADC RC1000 with one Virtex XCV1000). The speed gain for a lattice gas FHP-III model with 2048×1024 lattice is 140 times compared with Pentium-III 700Mhz.

1. はじめに

本論文では，Field Programmable Gate Array（FPGA）を用いた格子ガスオートマトンの高速計算について述べる．格子ガスオートマトンとはセルラオートマトン法の一種で流体力学等のシミュレーションに用いられる．格子ガスオートマトンのモデルでは，短いbit列データに対する整数演算のみで構成された簡単な遷移則がすべての格子にいっせいに適応され，各格子の状態が更新される．格子ガスオートマトンモデルは原理的に高い並列性を有しており，多くの分野において研究対象とされ，これまでに多種にわたる超並列計算システムや，FPGAを用いた並列システムによる高速化のアプローチがなされてきた[2)~8)]．

FPGAは書き換え可能なハードウェアであり，各問題に対する専用回路を容易に実現できることから，様々な問題の高速計算に広く用いられている．近年，半導体技術の進歩によりFPGAのゲート量は急激に増加し，1Mゲートを超えるデバイスも登場している．それゆえ，最新のFPGA 1チップを用い，格子ガスオートマトンモデルにおいて各格子の次の世代の状態を100格子以上並列に計算することが可能である．しかしながら，FPGAの内部メモリの容量はいまだに実用的な規模のシミュレーションに必要な格子数すべてを保持するほどの容量を持ち合わせていない（一般的に 1000×1000 格子以上必要）．このため，外部メモリを用いることで格子データの保持を実現する必要があるが，その場合にはFPGAの入出力性能が並列計算の際にボトルネックとなる．

本論文では，入出力性能が制限されているような，小規模システムを用いた格子ガスオートマトンモデルの高速計算方式を提案する．この方式では，まずはじ

† 筑波大学大学院工学研究科
Doctoral Program in Engineering, University of Tsukuba
†† 筑波大学機能工学系
Institute of Engineering Mechanics and Systems, University of Tsukuba

めに全格子の状態データが書き込まれた外部メモリから k 個の格子データ（k は外部メモリへの入出力幅によって決まる）を clock ごとに読み出すことから始まる．そして，FPGA 内に読み込まれたデータに対し，n 段のパイプラインで構成された回路により n 回連続して遷移則が適用され，n 世代後の格子の状態が計算される（ゆえに，パイプラインが全段稼働する際には $k \times n$ 格子が並列に計算される）．しかし，この方式では k 個の格子データの両外側のデータが入力されないため，n 回繰り返して遷移則を適用すると両側から n 格子分のデータは正しい値をとることができない．したがって，$(k-2n)$ 格子のみ正しい結果を得ることができる．

この計算方式において間違った計算結果を得る格子の比率をなるべく減らすには，一度 FPGA に読み込んだ格子データを効率良く再利用することが必要である．それゆえ，本高速計算システムでは内部メモリバンド幅が広い FPGA が必要となる．今回用いた Xilinx 社製の FPGA は，内部に Distributed RAM と呼ばれるバンド幅の広いメモリのモジュールを構成することができる．本論文では，Virtex XCV1000 が 1 チップ搭載されている PCI ボード（Alpha Data 社製 ADC RC1000）を対象として回路設計，性能評価を行った．格子ガスオートマトンモデルの一種である FHP-III モデルにおいて格子数を 2048×1024 格子としたとき，マイクロプロセッサ（Pentium III 700 MHz）に比べ約 140 倍の高速化が得られた．

2. 格子ガスオートマトン

格子ガスオートマトン法は，セルラオートマトン法を流体解析に適応しようとしたもので，流体を粒子の集まりとして離散的に扱う．このため，複雑な壁界面形状の動的変化の扱いが容易であり，多相流等の複雑な流体解析に適している．また，粒子間の局所的な相互作用から次の粒子の状態を計算するもので原理的に並列処理に非常に適している．

2.1 FHP モデル

FHP モデルとは 1986 年に Frisch らによって発表された格子ガスオートマトン法の流体解析モデルの発展型である[1]．FHP モデルでは 6 方格子を格子状に**図 1** のように並べることで格子面を形成する．図の太線は各格子の境界を表し，黒い点は粒子を意味している．1 つの格子にはまず式 (1) で表される 6 つの方向を速度方向として持つ粒子が存在できる（下式における i は図 1 左を参照．ただし，1 つの格子内では 1 つの速度方向に対して，1 つの粒子のみ存在することが許されている）．さらに 1 つの静止している粒子（図 1 左の 6 方格子の中央に位置する 0）も存在することができる（この粒子を考慮に入れないモデルも存在する）．

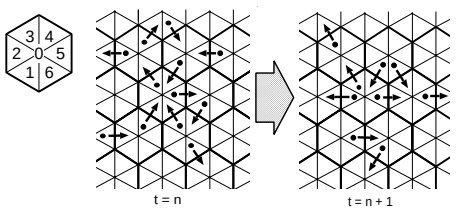

図 1 FHP モデルにおける 6 方格子構造
Fig. 1 Hexagonal grids for FHP model.

$$e_i(x,y) = \cos\left(\frac{5\pi}{3} - \frac{\pi i}{3}\right)x + \sin\left(\frac{5\pi}{3} - \frac{\pi i}{3}\right)y \quad (i = 1, 2, ..., 6) \quad (1)$$

FHP モデルではまず，格子面上の格子にランダムに粒子を配置し，

(1) 格子上に存在する粒子は単位ステップごとにその速度方向に従い，最近接の格子まで移動する．
(2) 各格子でいっせいに別の粒子と衝突，散乱し，速度方向が変わる．

以上の処理を繰り返し行うことにより，格子面上に存在する粒子の状態から密度，速度を導き，流体のシミュレーションを行う．

1986 年に発表された FHP-I モデルは，2 体の正面衝突と対称な 3 体の衝突の合計 5 通りの衝突のみを考慮したモデルであった．このモデルは衝突則こそ簡単なものであるが，体積粘性がゼロになる等の問題点が生じ，現実的なシミュレーションモデルとしてはあまり有効なものではない．そこで FHP-II モデルでは 6 方向の速度に加え，静止している粒子（速度が 0）を加わり，体積粘性がゼロになる問題点が解決された．また，FHP-II モデルは 2〜4 体の衝突まで考慮に入れたモデルとなっており，その衝突の総数は 22 通り（対称なパターンは同じものと見なす）となる．本研究のターゲットとしている FHP-III モデルは，前述した FHP-II モデルを発展させたもので，すべての衝突を考慮に入れたモデルである．その衝突パターンは最も複雑で 76 通り（対称なものは同じものと見なす）にもなる．

図 2 に FHP-III モデルの遷移則の一部を示す．図 2 a)〜g) の黒点は粒子を表し，矢印は粒子の速度方向を，小さな円は静止している粒子を表している．図 2 a)〜g) の矢印左側は衝突前の各粒子の状態を，右側は衝突後の各粒子の状態を表している．図 2 a)，e)，

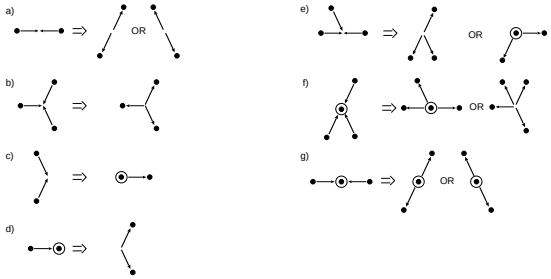

図 2 FHP-III モデルにおける遷移則
Fig. 2 Set of collision rules for the FHP-III.

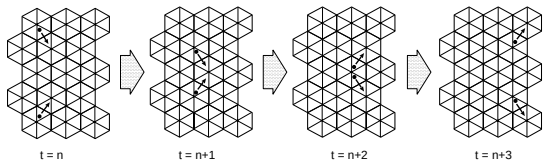

図 3 FHP モデルの振舞い（2 粒子の衝突）
Fig. 3 Collision of two particles.

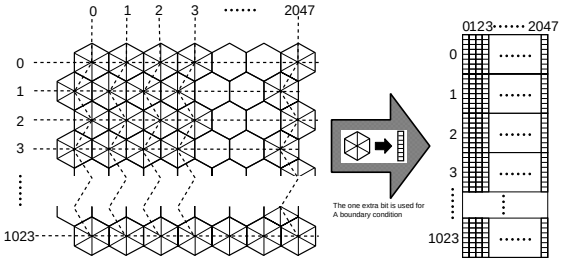

図 4 6 方格子面から 2 次元配列への変換
Fig. 4 Mapping from hexagonal grids to two dimension arrays.

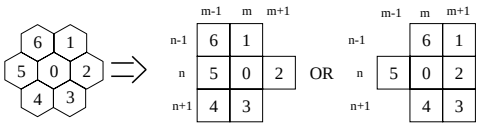

図 5 2 次元配列における近隣の格子との関係
Fig. 5 Neighborhood cells on two dimension arrays.

f)，g) のように粒子間の衝突には，衝突後の粒子の状態が 2 通り存在する場合があり，2 つの状態は 2 分の 1 の確率で起こりうる．このような遷移則を用いることにより質量保存則と運動量保存則が成り立つ．

上記の遷移則を用い，移動のプロセスを繰り返すと**図 3** に示すような任意の格子面上での，2 粒子の衝突における振舞いを見ることができる．2 つの粒子は遷移則に従い左から右へと移動をしていく．途中，2 つの粒子は衝突し（図 3 中 t = n + 2），互いに速度方向を変え再び移動していく．

2.2 FHP-III モデルの計算方法

FHP-III モデルは，格子面は多数の正六角形を格子状に敷き詰めた構成となっていることから，実際に計算をする場合にはメモリに格子データを格納するために格子面を 2 次元配列に変換する必要がある．**図 4** に格子面を 2 次元配列に変換する方法を示す．まず，格子面における各格子上の粒子の有無を以下のように表す（格子の位置は格子面では xy 座標系で表す）．

$$Cell_i(x,y) = \begin{cases} 1 & \text{粒子が存在する} \\ 0 & \text{存在しない} \end{cases} \quad (i = 0, 1, 2, ..., 6) \quad (2)$$

これにより各格子は速度方向それぞれに対する粒子の有無を表す 6 bit，そして静止粒子を表す 1 bit の合計 7 bit で構成される．そこにこの格子が境界面であるかどうかを表す 1 bit を加え，合計で 8 bit により各格子を表す．そして図 4 のように格子面の各列に対して番号を与え，各行に上から順に番号を割り振ることにより，図のような 2 次元配列への変換を行う．この変換を行うことで，格子面上の格子の状態データを効率良くメモリに格納することができる．

2 次元配列に変換した場合に近接する各格子間の関係は**図 5** のように変わる．近接する 6 つの格子の位置は各格子の座標によって異なり，格子が偶数行の格子の場合には図 5 左の，奇数行の格子の場合には図 5 右の位置関係で表される．各格子の次の世代の状態を計算するには以下のような処理が行われる．まず，近隣に境界面があるかどうかを隣接した 6 つの格子の最上位 bit により判定し，境界面が存在しない場合は図 5 で示した位置の格子から必要なデータを 1 bit ずつ取得する．図 5 中の四角形内の数字はこのときに読み出される bit の位置を表している．境界面が隣接していた場合は，計算対象となっている格子中の境界面方向へ移動する粒子は境界面において反射するものと見なし，その粒子に対応した格子データを計算の際に用いる．このようにして，取得した 7 bit の値と，2 種類ある衝突後の状態のうち，どちらの状態を適用するかを決定する 1 bit（乱数を生成することにより得られる）の 8 bit を用いて計算対象となる格子の状態を更新する．すべての格子について次の世代の状態を計算し，得られた結果が時刻 t + 1 の粒子の状態となる．

2.3 関 連 研 究

FHP モデルは変換に用いる遷移則の単純さや変換時における各格子間の依存関係の少なさにより，現在まで数々の並列計算機やワークステーションクラスタ等の研究の対象となってきた．

まず，汎用の高並列システムを用いたものは[4),6),7)]，

複数の汎用プロセッサを用いることにより，単一プロセッサによる処理に対する高速化を狙ったものである．最大プロセッサ数分の高速化が期待されるがシステム規模は非常に大きなものとなる．

これに対し，専用システムを用いた高速化も試みられている[2),5),8)]．これらのシステムでは主に，メモリと SRAM を用いたテーブルとで構成されたユニットを格子状に配置し，それら各ユニットに格子面上の格子を分割して割り当て，それぞれのユニットで割り当てられた格子の次の世代の状態を求めるという方法がとられている．この方式では，周辺の格子データの読み出し，その処理等をすべて並列化することが可能であり，より高い性能を実現することができるが，ユニット数（同時に処理される格子の数）に比例した入出力データ幅が必要となる．また，高性能を実現するためには各問題ごとに専用化された演算ユニットが必要となるという問題もある．この問題点を解決するために FPGA を用いた研究もあるが[2),8)]．当時の FPGA の回路規模はまだ小さく（1996 年当時で約 100k ゲート相当）実用的な規模の格子面を対象とする場合，十分な並列性を実現することができなかったため，単に FPGA を高速な RAM コントローラとして用いるにとどまっている[8)]．また，専用計算器自体も本論文で提案するようなパイプライン処理を実現していない．したがって，すでに述べたように，n 倍の並列処理（n 格子の並列処理）を実現するために n 格子分のメモリバンド幅を必要としている．これは，当時の回路規模では LSI 1 チップ上に多数の演算ユニットを実現することが困難であり，システムとして多数の LSI を用いることが必要であったためである．この場合，多数の LSI を用いたパイプライン処理よりは，格子面を分割し，それぞれを並列に処理する単純な並列処理方式の方が回路が簡潔であり，より効率的である．

これに対し本研究では，近年のより大規模な FPGA（今回使用したのは 1M ゲート相当の FPGA）を用いて，内部メモリを効率的に利用することにより複数世代のパイプライン処理を実現した．近年の大規模な FPGA では多数の演算ユニットを 1 チップ上に実現できるものの入出力ピン数はほとんど増加していないため，チップ上に実現した演算ユニットをできるだけ効率的に動作させることができる計算方式が必要となる．このため，本研究では

(1) パイプライン処理の導入，
(2) 限られたメモリバンド幅により生じる正しく計算ができない部分を再計算する，
(3) FPGA の内部メモリを有効に活用し，計算途中状態を適切に記憶させることにより正しく計算できない部分を減らす，

という計算方式を提案した．これによりメモリに対する入出力データ幅が制限されているような小規模システムでも高い並列性を実現し，速度向上を得ることができた．

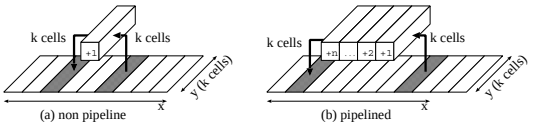

図 6 ハードウェアにおける基本計算 1
Fig. 6 Basic strategy of the computation method 1.

3. 高速計算方式の概要

本章では，格子ガスオートマトンの高速計算方式の基本的な方針について述べる．

本研究で提案する高速計算方式では以下に示すような FPGA の入出力データ幅と格子面の大きさとの関係により計算方式が異なる．

(1) FPGA の入出力データ幅 >= 格子面の幅
(2) FPGA の入出力データ幅 < 格子面の幅

3.1 FPGA の入出力データ幅 >= 格子面の幅の場合

図 6 は上記 (1) の関係，つまり格子面の幅が FPGA の入出力データ幅と比べて等しいか，狭い場合の計算方式を示したものである．この場合，格子面の幅が FPGA の入出力データ幅を超えないために，FPGA には 1 clock ごとに，格子面の 1 列（k 格子）ずつがメモリから読み込まれる．

まず図 6 (a) では，k 格子の次の世代の状態を同時に計算するための演算ユニットが FPGA 上に実現されているとする．そして，まず 3 列分の格子データを読み込んだ時点で 3 列中の 2 列目に位置する k 個の格子について次の世代の状態が計算される（格子面を 2 次元配列上に配置したため，各格子の次の世代の状態を求めるには図 5 で示したように近隣の 6 格子と対象となっている格子の合計 7 格子の格子データが必要となるが，これらの格子は計算の対象となる格子の上下左右の 3 行 3 列中に含まれている）．また，この場合，境界面では各格子上の粒子は反射するものとして処理されるため，第 0 行，および第 $k-1$ 行に位置する各格子についても次の世代の状態を正しく求めることができる．その後は，1 clock ごとに 1 列（k 格子）ずつメモリから FPGA に読み込まれ，その格子データと前の列の計算に利用した格子データ 2 列分を再利用することによって次の格子の状態を次々と計算する

ことができる．したがって，図 6 (a) の場合，FPGA 内には 3 列分のデータが保持されている．

さらに図 6 (b) に示すように，FPGA 上に $k \times n$ 個の演算ユニットが実現されているとする．このとき，いったんメモリから読み出されたデータに対して n 回連続して格子の状態を更新するための計算を行うことにより，n 世代先の格子の状態を求めることができる．このように n 世代続けて計算を行うためには，演算ユニットを k 個ずつ n 段のパイプライン状に構成し（図 6 (b) 中の+n…+2 +1 の部分），FPGA 内部に $3n$ 列分（各世代ごとに 3 列ずつ）の格子データを保持可能なメモリを用意する必要がある．n 列分の格子が同時に計算されるため，回路規模，メモリ容量とも，図 6 (a) の n 倍となる．これによりメモリアクセスの回数を増やさずに，n 世代先の格子の状態を計算することができる．パイプライン処理を行うため，初めの $3n$ clock はいくつかのステージが稼働していないものの，パイプラインの全ステージが稼働状態となると 1 度に $n \times k$ 格子分の格子の状態を計算することが可能となる．実用的な規模の問題においては格子面の規模は十分大きなものであるため，パイプラインの全ステージが稼働するために $3n$ clock かかったとしても，そのオーバヘッドは全体の処理の数パーセントに満たない．

3.2 FPGA の入出力データ幅 < 格子面の幅の場合

本節では上記 (2) の関係，つまり格子面の幅が FPGA の入出力データ幅より広い場合について述べる．この場合には，外部メモリからのデータ入力には制限があるため，FPGA が内部に持つメモリの容量によって計算方式が異なる．

まず，**図 7** に FPGA の内部メモリが格子面の幅に十分対応できるほどの容量を持つ場合の計算方式を示す．FPGA には k 個の演算ユニットが実現されているものとする．このとき，格子面の幅を $m \times k$ 格子とすると，FPGA には，1 度に k 格子分しか外部メモリから読み込むことができない．そこで，1 列分（$m \times k$ 格子）の各格子の次の世代の状態を計算するのに，前節で述べた計算方式を m 回縦方向に繰り返すことで 1 列分の格子の状態の計算を実現することができる．まず図 7 (1) では，最初の k 格子分の格子データが読み込まれ，次いで次の k 個の格子データが読み込まれる（図 7 (2)）．それぞれにおいて FPGA 中に 3 列分のデータが溜まると，k 個の演算ユニットにより 3 列中の 2 列目に位置する k 個の格子に対し次の世代の状態が計算される．このとき，図 7 (1) において計算に用いられた格子の状態が上端の境界条件として図 7 (2) の計算に引き渡される．同様に境界条件における格子データを次々と引き渡すことにより 1 列分の処理を終了する．

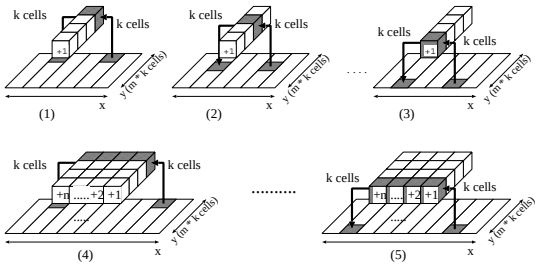

図 7 ハードウェアにおける基本計算 2
Fig. 7 Basic strategy of the computation method 2.

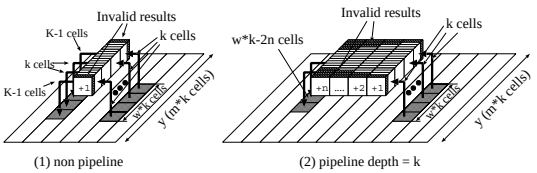

図 8 ハードウェアにおける基本計算 3
Fig. 8 Basic strategy of the computation method 3.

この場合も前節と同様にパイプライン状に配置された複数の回路を用いて 1 度に数世代先の格子の状態求めることができる（図 7 (4)，(5)）．FPGA 内で n 世代繰り返して計算するためには，$3n$ 列分の格子データを保持する必要がある．さらに，各 k 格子分の両端の境界条件が n 世代繰り返し計算する間は正しく入力されないため，n 世代分の境界条件つまり $2n$ 格子分を保持するためのメモリが必要となる．この場合も前節と同様にパイプラインの全ステージが稼働状態にあるとき，$k \times n$ 格子を 1 度に計算することができる．

次に，FPGA の内部メモリの容量が格子面の幅よりも小さい場合の計算方式を示す．これは最も複雑な計算方式で，本研究で FPGA に実装する際にはこの方式が適応される．**図 8** (1) に示すように格子面の幅が $m \times k$ 格子分，FPGA のデータ入力幅が k 格子分であり，FPGA の内部メモリが $3 \times (w \times k)$（w は $m \gg w$ を満たす任意の実数）格子分の格子データのみ内部に保持することができるとする．FPGA にはこれまでと同様に k 個の演算ユニットが実現されている．このとき，FPGA は k 格子ずつ w 回に分けて $w \times k$ 格子分の格子データを読み取り，次いで，同様にして次の列の $w \times k$ 格子のデータを読み込む．これを繰り返すことにより，$w \times k$ 個の格子に対して，k 格子ずつ k 個の演算ユニットにより，次の世代の

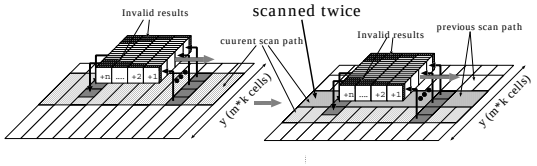

図 9 重複の必要なハードウェアにおける格子スキャン
Fig. 9 Over-wrapped scanning path of the lattice.

状態が順次計算されていくものとする．したがって，w ステップで $w \times k$ 格子の処理を終了する．このとき，FPGA のメモリの容量が $w \times k$ 格子分しかないため，両端の各格子の上下の格子データが読み込まれず，図 8 (1) 中に示すように両端の 1 格子は正しい次の世代の状態を得ることができない（図 8 (1) 中の黒い部分）．このため，$w \times k - 2$ 格子分の計算結果のみが有効となる．

FPGA が $w \times k$ 格子分のデータを $3 \times n$ 列分保持することが可能であれば，前節と同様に n 世代までの計算を連続して行うことができる（図 8 (2)）．しかし，この場合にも，$w \times k$ 格子の前後の格子データが読み込まれないために 1 世代計算するごとに両端から 1 格子ずつ格子の状態が正しい値を得ることができない．よって n 世代後，つまり n 回連続して計算を行った後には両端から n 格子分のデータの値が正しい値を示さなくなる．したがって，正しい結果を得ることができる格子数 p は

$$p = w \times k - 2n \qquad (3)$$

となり，全格子面の計算を行うためには**図 9** に示すように，次の $w \times k$ 行を計算する際に FPGA への入力位置を $2n$ 格子分ずつずらして（この $2n$ 格子に関しては 2 回読み込まれる）処理を進めていく必要がある．

この場合もパイプラインの全ステージが稼働状態であるときには，$k \times n$ 格子を 1 度に計算することができる．しかし，この計算方式では 1 世代計算するごとに両端から 1 格子ずつの格子の状態が正しく与えられない．よって 1 度に計算される格子のうち有効な値を持つ格子の数 q（以降，並列度と呼ぶ）は，式 (3) の p に，パイプラインの深さである n を掛け，k 格子ずつの入力を行う回数 w で割ることにより（式 (3) の格子数の計算を終了するのに w ステップが必要），

$$q = p \times n/w = (k - 2n/w) \times n \qquad (4)$$

と表すことができる．したがって，w の値が大きいほど，つまり，k 格子分の入力を行う回数が多いほど，正しい状態を得ることができない格子数が減る．

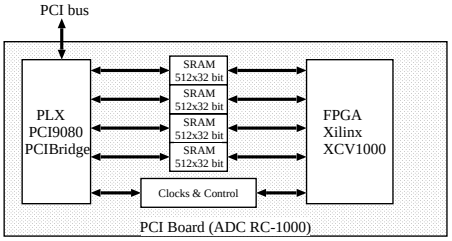

図 10 FPGA ボードの構成図
Fig. 10 Block diagram of RC1000 PCI board.

4. ハードウェアの構成

4.1 高速計算回路の構成

図 10 に本研究で性能評価に用いた PCI ボード（Alpha Data 社製 ADC RC-1000）の構成を示す．このボードは，1M ゲート相当の FPGA（Virtex XCV 1000）を 1 チップと総容量 8 MByte の SRAM を外部メモリとして搭載している．

本研究で使用する Virtex XCV1000 は，Look Up Table（LUT），Block RAM，インタコネクトにより構成される．LUT は 4 入力 1 出力の任意の論理演算を実現するための基本素子である．この LUT は深さ 16 bit × 幅 1 bit の SRAM から構成されており，Xilinx 社の FPGA ではこの LUT をそのまま 16×1 のメモリとして用いることができる．このように LUT をメモリとして用いる場合は，Distributed RAM と呼ばれる．この Distributed RAM を複数使用することによって，データ幅が広く，深さの浅いメモリを構成できる．今回，高速計算を実現するにあたり，Distributed RAM を用いて約 1 kbit のデータの読み/書きを同時に行った．

PCI ボード上の外部メモリは 2 MByte（データ幅 32 bit，アドレスデータ幅 19 bit で構成されている）の容量を持つメモリ 4 バンクから構成されており，各メモリは個別に FPGA からアクセスすることが可能である．本研究では格子ガスオートマトン法の一種である FHP-III モデルの 2048×1024 格子について性能比較を行った．これは FPGA の内部メモリでは保持することのできないほど規模の大きな格子面であるため，外部メモリを用いる必要がある．このボードに搭載しているメモリの総容量は 8 MByte であるため，4 バンクのうち 2 バンクを格子データの読み出し用として，残りの 2 バンクを書き込み用とすることで格子データの読み出しと書き込みを同時に行うことができる．これにより，64 bit の同時読み出し，書き込みが可能となり，格子データは図 4 に示したように 8 bit

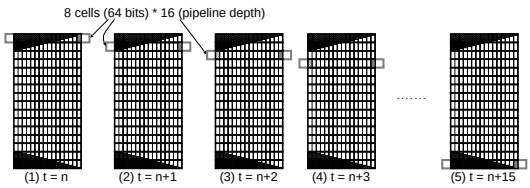

図 11 FPGA 回路による演算の流れ
Fig. 11 Processing by the circuit on XCV1000.

であるため，8 格子分の格子データの同時読み出し，書き込みが可能となる．また格子面全体の計算が終了したときには読み出し用のメモリと書き込み用のメモリを切り替えることでデータを移し変えることなしにそのまま次の計算を開始することができる．

本研究では本高速計算方式での速度向上を最大限に引き出すため，格子の状態を計算する回路のみを FPGA に実装する．したがって，得られた結果からの格子面上の粒子の分布等の解析，表示等はパーソナルコンピュータ上で行われる．そのため，PCI ボード上の外部メモリとパーソナルコンピュータとの通信を行う必要がある．そこで DMA （Direct Memory Access）転送を用いてパーソナルコンピュータと外部メモリ間の格子データの通信を行う．DMA 転送を用いることで，高速に格子データをパーソナルコンピュータに転送することができる．FPGA 上に実現する回路データの作成には，Xilinx 社製の Foundation と呼ばれるツールを用いた．

4.2 高速計算回路による処理の流れ

ここでは，PCI ボード上の Virtex XCV1000 に高速計算方式を実装した場合の処理の流れについて述べる．前節で述べたように性能比較をする際に用いる格子面の規模は FPGA の入出力データ幅，内部メモリの容量をはるかに超えるものであるため，高速計算方式は 3.2 節の FPGA の入力データ幅 < 格子面の幅の関係に加え，格子面の幅が FPGA の内部メモリ容量より大きい場合の計算方式に準ずる．

図 11 に Virtex XCV1000 上での処理の様子を示す．図 11 にある小さなマス目（各図には 16×16 個存在する）は 8 格子分の格子データ（計 64 bit）を表す．XCV1000 上の Distributed RAM は 16×1 bit の構成であるため，3.2 節式 (3)，(4) において $w = 16$ とすることができる．また，XCV1000 では幅 128 格子分（$w \times k = 16 \times 8$）深さ 48（$n \times 3 = 16 \times 3$）までの格子データを FPGA 上に格納することができる．すなわち，$w = 16$，$k = 8$，$n = 16$ となる．

図 11 (1) において 8 格子分の格子データ（図 11 (1) 中の右上端にある色の付いたマス目の部分）が外部メモリから読み込まれると，太線の長方形に囲まれた部分（16×8 格子分）の 128 格子の次の世代の状態が同時に計算され，16 世代先の格子の状態が太線の長方形左端から出力される．そして，図 11 (2) に示すように次の 8 格子（図 11 (2) 中の右端の色の付いた部分）が読み込まれ，図 11 (1) の 1 段下の行（図 11 (2) の太線で囲まれた部分）の格子についてそれぞれの次の世代の状態が計算される．このとき，太線で囲まれた格子データの上端にある格子に正しい境界条件を与えるために，図 11 (1) の計算の際に用いた格子データ中の下端にある格子データをレジスタに保持し，これを上端の格子の計算の際に用いている．また，下端の格子については Distributed RAM にあらかじめ保持してある次の行の格子データ（図 11 (3) 中の太線で囲まれた部分）の上端の格子データを用いている．以上の処理を繰り返すことにより，幅 128 行分の格子すべての 16 世代先の格子の状態を計算する．このとき，FPGA の内部メモリ容量に制限があるため，3.2 節で述べたように 128 格子の上下両端の格子に対する近隣の格子データを正しく与えることができない．そのため，1 世代分の計算で両端の 1 格子は間違った結果を得る．XCV1000 への実装においては 16 世代先の格子の状態を計算するため，両端の間違った計算結果が隣の格子に影響を与え続け，結果として両端からパイプラインの段数分の格子，すなわち 16 格子分の格子は間違った計算結果を出力することになる（図 11 の黒い部分）．

よって実際に計算して正しい状態が得られる 16 世代先の格子数 p は式 (3) $p = w \times k - 2n$ において $w = 16$，$n = 16$，$k = 8$ であるため，96 格子となる．また，並列度 q は，パイプラインの深さである n と，格子データ入出力幅である k，k 格子分の入力を行う回数 w を用いて式 (4) $q = (k - 2n/w) \times n$ より，$q = 96$ となる．

4.3 境界を考慮した格子面全体の処理

図 12 に格子面全体を処理するための方法を示す．

まず，格子面の上端から前節で述べた計算処理を格子面の左端から右端にかけて 128 格子ずつ行う（図 12 (1) 参照，第 3.2 節により連続して処理可能な格子数は $k \times w$ であり，今回の実装では $k = 8$，$w = 16$ であるため）．格子面の上端は境界が存在すると見なすため（境界で粒子は反射する），さらに上方の格子を計算の際には必要としない．このため，格子の状態が計算される 128 格子のうち，上端から 16 格子分は正しい計算結果を得ることができ，下端から 16 格子のみが間違った結果を得る．したがって，128 格子のう

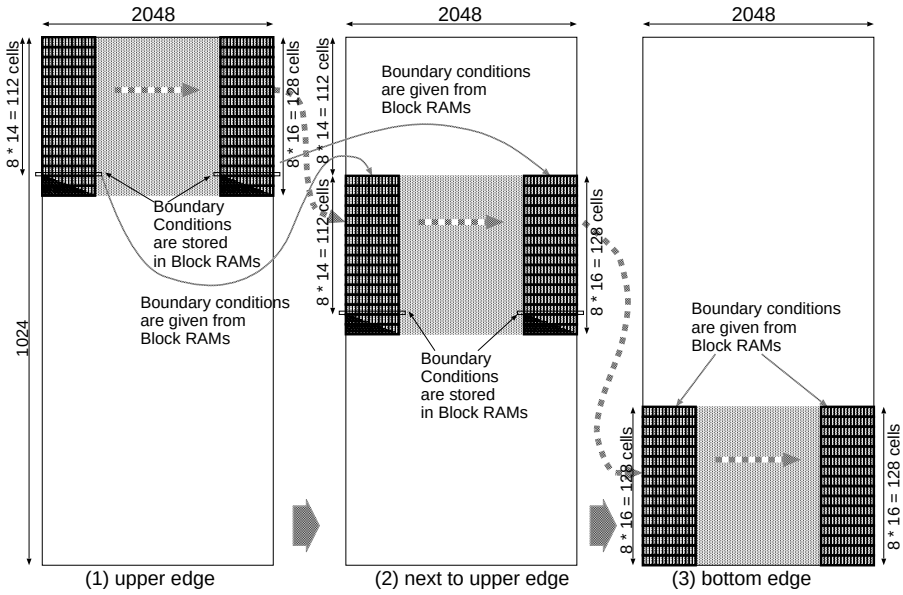

図 12 Block RAM を用いた全格子スキャン
Fig. 12 Processing of the whole lattice with optimized method.

ち 112 格子について正しい計算結果を出力できる．ここで図 12 (1) に示す部分の計算時間について考えることにする．格子面の大きさを $x \times y$ 格子とすると（x は格子面の横，y は格子面の縦を表す．図 12 では $x = 2048$，$y = 1024$），左端から右端まで 16 世代先の格子の状態を計算するには，$(w \times x) + \alpha$ clock 必要となる．ここで α clock はパイプラインの全ステージが稼働状態となるまでのオーバヘッドである．次に α の値を求める．格子面の左右端の外側は境界であるために（境界で粒子は反射する）2 列分の格子データを読み込むと計算を開始することができる．このために計算が開始されるまで $2 \times w$ clock を要する．また，保持されている格子データから次の格子の状態を計算するのに格子データの選択，遷移則の適用で合計 2 clock 必要となる．したがって，はじめの 8 格子分について 16 世代先の状態を計算するのに

$$\alpha = n \times ((2 \times w) + 2) \quad \text{clock} \qquad (5)$$

を要する．よって今回の実装では，$n = 16$，$w = 16$ であることより 544 clock となる．この後はパイプライン制御を行っているために 1 clock ごとに計算結果が出力される．したがって，図 12 (1) に示す部分の計算に要する clock 数は $16 \times 2048 + 544$ clock となる．

図 12 (1) では 0〜111 格子目までのみが正しい結果であったため，次には 112 格子目から正しい結果が得られるように計算を行う必要がある．図 11 に示した方式において，上下両端が境界面ではない場合には計算された格子のうち，上下からパイプライン段数分の 16 格子分（$n = 16$ により）の格子データは正しい値とならない．このため，$112 - 16 = 96$ 格子目から格子データの読み込みを開始する必要がある．しかし，図 11 において，ある 128 格子分の格子の計算の際に上端で必要となる境界条件のデータはそれ以前の 128 格子分の下端から 16 行の計算途中に 1 度用いられた値である．したがって，この値を保持することができれば図 11 における上端から 16 格子分の計算結果が間違った値となることを防ぐことができる．この境界条件として必要となるデータ量は 1 格子分の計算に対して 4 bit であり，2048×1024 格子の格子面を対象にした場合には，4×2048 bit，さらに 16 世代分連続して計算するために各世代ごとに境界条件のデータが必要となり，必要な格子データ量は $4 \times 2048 \times 16 = 131072$ bit となる．XCV1000 は Distributed RAM のほかに Block RAM と呼ばれるメモリを持っている．Block RAM の容量は合計 $32 \times 16 \times 256 = 131072$ bit であり，必要とされる境界における格子データを保持することができる．したがって，図 12 (1) で計算された境界における格子データを Block RAM 中に保持しておき，図 12 (2) の計算の際に用いることにより，上端の 16 行に対して正しい結果を得ることができる．よって正しい結果を得ることができる 16 世代先の格子数 p は p' に改善され，

$$p' = p + n = w \times k - n \qquad (6)$$

となり，並列度 q は式 (7) の q' に改善される．

$$q' = (k - n/w) \times n \tag{7}$$

したがって，今回の実装では $w = 16$，$n = 16$，$k = 8$ であるため，$p' = 112$ となり 112 格子分の正しい結果を得ることができ，その並列度 q' は $q' = 112$ となる．

以上の処理を格子面の下端（図 12 (3)）に到達するまで繰り返す．格子面の下端の 128 格子を計算する場合，前述した方法で計算をすることで 128 格子分の上方から 112 格子分の格子は正しい計算結果を得ることができる．さらに格子面の下端は境界であるため，下端の 16 格子に関しても正しい結果を得ることができ，128 格子すべての格子に対して正しい計算結果を出力することができる（図 12 (3)）．

格子面全体を処理するのに，1 回の計算で通常は 112 格子分，下端部分では 128 格子分の格子の状態を計算することができるために，今回の実装の際に対象とした 2048×1024 格子分の計算を行うには格子面の左端からの計算を 9 回（$112 \times 8 + 128 = 1024$）行う必要がある．ゆえに格子面全体について 16 世代先の状態を計算するのに必要な clock 数 β は w，x，α を用いて，

$$\beta = 9 \times w \times x + \alpha \tag{8}$$

と表すことができる．今回の実装では，$w = 16$，$x = 2048$，$\alpha = 544$ であるため $\beta = 295456$ clock となる．ゆえに格子面全体を計算するために必要な clock 数（295456 clock）に対して，パイプラインの全ステージが稼働状態になるまでの clock 数（544 clock）は，1 パーセントにも満たない．よって，性能評価においてこのパイプライン処理のためのオーバヘッドは無視できる．

4.4 FPGA 内の計算回路の詳細について

Virtex XCV1000 に実装した FHP-III モデルの計算回路の構成を**図 13** に示す．

この計算回路は主に 8×16 の配列状に並べられた演算ユニットと，境界の格子データを保存しておくメモリから構成されている．それぞれの演算ユニットは各格子の次の世代の状態を計算する．図 13 において右端から 1 clock ごとに 8 格子分の格子データが読み出され，最右端に位置する 8 個の演算ユニットにおいて各格子の次の世代の状態が計算される．これをパイプライン状に構成された 16 列の演算ユニットにより 16 回連続して計算することで 16 世代先の状態を求めることができる．このとき，各演算ユニットは Distributed RAM により 3 列分 $\times 16$ のデータを保持し，順次 1 列ずつ左側の演算ユニットに引き渡す．

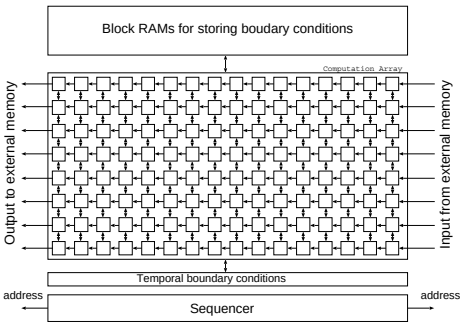

図 13 XCV1000 上に構成された計算回路
Fig. 13 Overview of the circuit.

図 14 に演算ユニットの構成を示す．各演算ユニットは主に，Distributed RAM を用いて構成される格子データを保持するバッファと，4 つのバッファから出力されたデータのうち，計算時に必要なデータを選択し変換回路へ供給するセレクタ，境界条件の判定を行った後，遷移則に従い各格子の次の世代の状態を計算する変換回路により構成されている．

各格子について次の世代の状態を求める場合，3 列分の格子データが必要となる．そのため，演算ユニットにバッファを 4 つ並列に配置し，図 14 右のようなタイミングで 4 つのバッファの読み/書きを制御する．そうすることで 3 列分の格子データをバッファから出力している間にも次の格子の列をバッファに書き込むことができる．したがって，パイプラインがつまることなく，1 clock ごとに次の格子の状態を計算し，左側の演算ユニットへ格子データを出力することができる．

図 15 に変換回路部分の構成を示す．変換回路は主に境界条件の判定に用いられるセレクタと遷移則を記したテーブル，乱数発生器により構成されている．図 15 における各近隣格子からの格子データ選択方法は偶数行に存在する格子についてのもので，奇数行に存在する格子についても隣接する格子の位置関係が異なるものの，同様な変換回路が用いられる（奇数行における選択方法は 2.2 節参照）．各格子の次の世代の状態の計算には，図 15 に示すようにセレクタにより選択された各近隣格子のデータを用いテーブルを参照することで行われる．FHP モデルは六方格子を格子として使用しているため，遷移則の対称性を活かしテーブルへの入力を 8 入力から 7 入力に減らすことでテーブルの規模を 1/2 にすることができる．遷移則を適用する際に必要な乱数は線形合同法を用いた乱数発生器（実際には漸化式 $C_{n+1} = C_n << 2 + C_n + 1$ を論理回路で実現している）により生成される．

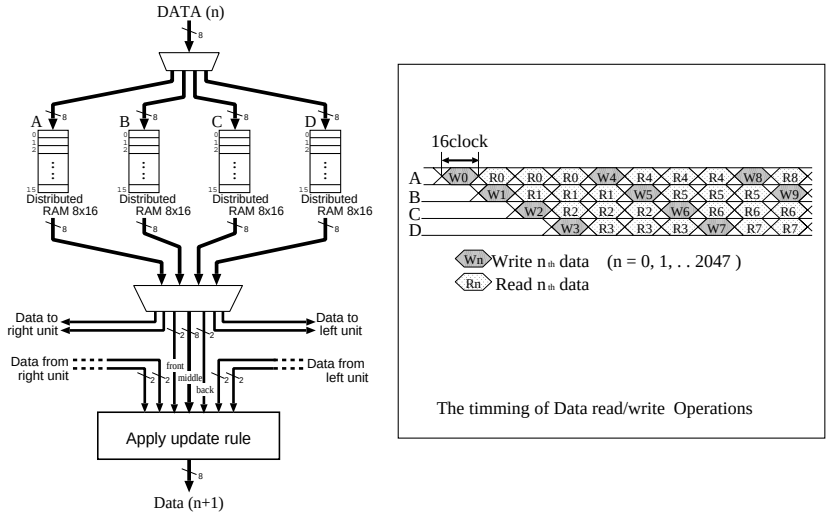

図 14 演算ユニットの回路構成
Fig. 14 The circuit for a cell on the grid.

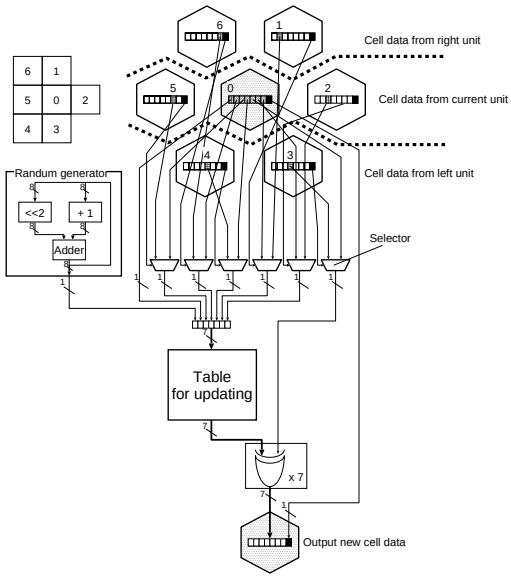

図 15 変換回路部分の構成
Fig. 15 The circuit for appling update rules.

4.5 FPGA のサイズと入出力データ幅における並列度の関係

表 1 に FPGA の入出力データ幅と本計算方式における並列度との関係を示す．式 (7) において w，k，n の値は Distributed RAM の深さが 16 であることにより $w = 16$ と固定されるため，FPGA の入出力データ幅 k とパイプラインの深さ n を $(k, n) = (4, 32)$, $(8, 16)$, $(16, 8)$ とし並列度 q' を求めた（FPGA の内部メモリの容量は一定であり，FPGA の内部メモリに格納する必要がある格子データ量は $w \times k \times 3n$ であるため，$k \times n$ の値を一定とした）．入出力データ幅が 32 bit と 64 bit とでは明らかに並列度において差が見られるが，128 bit と 64 bit とを比較してもさほど違いは見られない．**表 2** は FPGA の入出力データ幅を前述した 64 bit に固定した状態での FPGA の規模と本高速計算方式を用いた場合の並列度との関係を示したものである．並列度 q' は式 (7) において $k = 8$, $w = 16$ に固定し，パイプラインの深さ（n）について表 2 上段では今回の実装での値である $n = 16$ を，下段では FPGA 回路規模が 2 倍となったときを想定し，実現できるパイプラインの深さ $n = 32$（FPGA の回路規模が 2 倍になることで Distributed RAM 量も 2 倍となる）を用いて求めた．

表 1 入出力データ幅と並列度の関係
Table 1 Effective parallelism (I/O datawidth).

入出力データ幅 ($8 \times k$ bit)	32+32	64+64	128+128
パイプラインの深さ (n)	32	16	8
並列度 (q')	64	112	124

表 2 FPGA のサイズと並列度の関係
Table 2 Effective parallelism (size of FPGA).

FPGA size（ゲート数）	Distributed RAM 使用数	LUT 使用数	並列度 (q')
1.0×10^6	4×10^3	20×10^3	112
2.0×10^6	8×10^3	40×10^3	192

$n = 32$ の場合も $n = 16$ のときと同様にパイプラインのスタートアップに要するオーバヘッドは無視することができる．これはパイプラインのスタートアップにかかる clock 数は式 (5) において $n = 32$，$k = 8$,

$w = 16$ より，$\alpha = 1088$ clock となり，格子面全体を処理するのに要する clock 数は式 (8) より $\beta = 296000$ clock となるため，スタートアップに要する clock 数は格子面全体の処理に要する clock 数に比べ十分に小さいからである．

FPGA の回路規模は近年大幅な増加の傾向にある．今後回路規模が倍程度に増加しても本高速計算方式は表 2 に示すように Distributed RAM で保持する格子データを拡大することによって入出力データ幅 k を変えずにパイプラインの深さを $n = 32$ とすることでさらなる高速化を望むことができる．

さらに 8M ゲート相当まで回路規模が増大した場合，FPGA の内部メモリ量も回路規模に比例して増加するため，1 列分の格子データを保持することが可能となり（計算方式は 3.2 節の始めに述べた格子面の幅が FPGA の入出力データ幅より広く，FPGA の内部メモリが格子面の幅に十分対応できるほどの容量を持つ場合の計算方式となる），すべての計算結果が有効となるため 1024 という並列度を達成することができるようになる．

5. 結　　果

性能評価を行った結果を**表 3** に示す．表 3 には 16 世代分の計算時間を示す．実行速度比較は Pentium III 700 MHz を比較対象とし，総格子数 2048×1024 の大きさの FHP-III モデルについて行った．結果約 140 倍という速度向上を得た．ソフトウエアの実行は 1 世代先の格子の状態を求める計算を行う場合の実行時間を 16 倍したものを採用した．ソフトウェアでは 1 世代先の格子の状態を求める計算を行うには，376 msec 必要である．格子面の規模が 2048×1024 格子であることより 1 格子の計算を行うのに約 185 nsec かかっている．1 格子計算するのに 185 nsec 必要となる要因として，

- 各格子の次の世代の状態を計算するために 1 bit の乱数を生成する必要がある（2 章参照），
- 各格子の次の世代の状態を計算するためには逐次的に自分を含めた各近隣の格子が境界であるかを判定し，それに対応して近隣の格子から格子データを 1 bit ずつ取得し，これに上記の乱数 1 bit を加え，テーブルを参照し次の状態を求める必要がある，
- 格子面の規模が大きいため，キャッシュがミスヒットを起こす，

等があげられる．

計算回路の動作周波数は 22.9 MHz である．本計算回路を実現するのに 1M ゲート相当 FPGA である Virtex XCV1000 のうち，95 パーセントのゲートを使用した．表 3 中の 140 倍という値は 16 世代ごとの計算結果を表示するためにパーソナルコンピュータに全格子データを DMA 転送する時間を含んだ値である．全格子面を計算するのには FPGA では 295456 clock かかるため（4.3 節），16 世代先の格子の状態を計算をするのに約 12 msec の時間を費やす．また，DMA 転送でデータを読み/書きする場合約 30 msec の時間を要する．格子面に初期値として与える乱数は DMA 転送を用いてパーソナルコンピュータから PCI ボード上の外部メモリへと送るが，はじめに 1 度送るのみであるため，評価結果には含めないことにした（**図 16** 参照）．DMA 転送に要する時間が実行時間の約 $30/(30+13)$ という高い割合を示している．より高速な入出力を備えるシステムに本計算方式を適用することにより，さらに高い速度向上を得ることができると期待される．DMA 転送をまったく考慮しない場合には，466 倍という非常に高速な処理速度となっている．

表 3 比較結果
Table 3 Result of speed gain.

	実行速度 (msec)	速度比
ソフトウェア	376×16	1
FPGA システム（DMA 転送含）	42.9	140
FPGA システム（DMA 転送無）	12.9	466

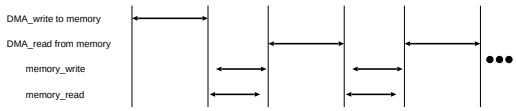

図 16 外部メモリの読み/書きと DMA 転送のタイミング
Fig. 16 The timming for memory read/write and DMA.

6. おわりに

本論文では，メモリバンド幅に制限のある小規模システムにおける，セルラオートマトンの高速計算について述べた．今回，格子ガスオートマトン法の一種である FHP-III モデル 2048×1024 格子について，FPGA ボード（ACD RC1000 Virtex XCV1000 を 1 チップ搭載）を用い，Pentium III 700 MHz に対して約 140 倍の速度向上が得られた．この計算方式で高速計算回路を構成した場合，データの制御部分と各格子の次の世代の状態を計算する部分が独立した回路構成となるため，各格子の状態を計算する部分のみを変更することにより，他のセルラオートマトン法にも適応することができる．また，DMA 転送の回数を増やすことで

多少の速度向上率は減少するものの，2048 × 1024 格子以上の格子面についても対応することができる．

現在の実装では 16 世代先の格子の状態が一括して計算されるため，16 世代ごとにしか結果を表示することができない．また，結果を表示するためにパーソナルコンピュータへデータ転送を行うための時間が全計算時間の半分以上を占めている．しかし，実際のシミュレーションを行う際には格子面を一定の領域に区切り，その領域内の粒子に対する密度や平均速度を用いて流体の振舞いの観測を行うため，16 世代ごとに必ず表示を行う必要はなく，DMA 転送によるオーバヘッドはより小さくなる．また，今後 FPGA 上で計算途中の格子データを表示に必要なだけのデータへ圧縮し，FPGA からパーソナルコンピュータへ転送する方式を開発することにより，さらなる処理時間の短縮が期待できる．

さらに今後の課題として，3 次元のセルラオートマトンにも対応できるようにアーキテクチャの検討を行っていきたい．

参 考 文 献

1) Frisch, U., d'Humires, D., Hasslacher, B., Lallemand, P. and Pomeau, Y.: Lattice gas hydrodynamics in two and three dimensions, *Complex Systems*, Vol.1, pp.649–707 (1987).
2) Shaw, P., Cockshott, P. and Barrie, P.: Implementation of Lattice Gasses Using FPGAs, *Journal of VLSI Signal Processing*, Vol.12, No.1, pp.51–66 (1996).
3) Adler, C., Boghosian, B.M., Flekkoy, E.G., Margolus, N. and Rothman, D.H.: Simulation Three Dimensional Hydrodynamics on a Cellular-Automata Machine, *Journal of Statistical Physics* (1995).
4) Moscinski, B.M.J. and Slota, R.: FHP lattice gas on networked workstations, Gruber, R. and Tomassini, M. (Eds.), *6th Joint EPS-APS International Conference on Physics Computing*, Lugano, Switzerland, pp.427–430 (1994).
5) Bubak, M., Gryniewicz, R., Moscinski, J. and Palubiak, T.: FHP lattice gas on T800 transputer board, *Proc. 1st Int. Conf. Parallel Processing and Applied Mathematics—PPAM'94*, Czestochowa, pp.28–34 (1994).
6) Bubak, M., Moscinski, J. and Slota, R.: Parallel program for 2-D FHP Lattice gas simulation on clusters of workstations, *Presentation at 10th Summer School on Computing Techniques in Physics, "High Performance Computing in Science,"* Skalsky dvur, Czech Republic (1995).
7) Slota, R. and Moscinski, J.: Lattice Gas Automata Simulation on HP/Convex Exemplar SPP1600, Sloot, P., Bubak, M. and Hertzberger, B.(Eds.), *Proc. Int. Conf. High Performance Computing and Networking*, Amsterdam, Lecture Note in Computer Science 1401, pp.963–965 (1998).
8) Margolus, N.: An FPGA architecture for DRAM-based systolic computations, *Proc. FCCM '97*, pp.2–11 (1997).

(平成 12 年 6 月 21 日受付)
(平成 13 年 4 月 6 日採録)

小堀 友義（学生会員）

1976 年生．1999 年筑波大学第三学群工学システム学類卒業，同年筑波大学大学院工学研究科入学．FPGA を用いたセルラオートマトンの高速計算システムの研究に従事．

丸山 勉（正会員）

1958 年生．1987 年東京大学大学院工学系研究科情報工学専門課程博士課程修了．同年日本電気（株）入社．並列オブジェクト指向言語，並列遺伝的アルゴリズム，並列マシン Cenju の開発/研究に従事．1997 年より筑波大学機能工学系助教授．書き換え可能なハードウェアを用いた計算の高速化に関する研究に従事．

星野 力（正会員）

1965 年京都大学大学院工学研究科博士課程電気工学専攻修了．工学博士．京都大学原子エネルギー研究所を経て，1980 年より 2001 年まで筑波大学機能工学系教授．原子力工学の研究を経て，1977 年より並列計算機 PACS シリーズの研究と開発，1989 年より人工生命，進化的計算に関する研究に従事．