卒業論文

ライフゲームにおけるパターン探索の並列化

氏　　名：酒井 俊介
学籍番号：2371020010-6
指導教員：山崎 勝弘 教授
提 出 日：2006 年 2 月 20 日

立命館大学 理工学部 情報学科

内容梗概

並列処理には大規模な問題でも計算時間を大幅に短縮できるというメリットがある。単一プロセッサの速度限界が近づく中で欠かせない技術の 1 つである。近年では、共有メモリ計算機の普及に伴い、並列プログラミングも分散メモリ環境から、共有メモリ環境へと移行しつつある。その共有メモリ用のプログラミングモデルとして現在注目を集めているのが OpenMP である。OpenMP は移植性が高く、プログラミングも比較的簡単なので、今後並列プログラミングの主流になると期待されている。また、高性能な PC が安価で手に入るようになり、Myrinet などの高速なネットワーク環境が普及してきた事から高性能な PC クラスタの構築が可能になった。

本論文では、並列処理の応用研究として本研究室で構築された SCore 型 PC クラスタ上でのライフゲームにおけるパターン探索の並列化について述べる。

ライフゲームにおけるパターン探索の並列化は、OpenMP を用いて行い、スレッド数 8 台で実行した。4×4 のセルに初期状態を発生させ、50×50、および 100×100 のセルの中心に配置し、スレッド数 8 台で実行した際、約 7.5 倍の速度向上が得られた。

目次

図目次

表目次

1.はじめに

並列処理研究の応用として、気象予測や、環境問題、流体計算、デジタル画像処理、遺伝子の解明、データベース処理などがある。その中でも特に気象予測や、環境問題、流体計算は並列処理の活用が広がるといわれている。ライフゲームはシミュレーションの基本といわれており、データの量も多く、並列処理に向いた性質を持つ。

並列計算機には 3 つのメモリモデルがあり、それぞれに異なった性質を持つ。複数の CPU がメモリを共有し､異なる CPU が同じメモリ空間にアクセス可能な共有メモリ並列計算機。メモリは各 CPU にローカルに接続され、異なる CPU のメモリに単独でのアクセスが不可能な分散メモリ並列計算機。また分散メモリの状態から異なる CPU のメモリ空間にアクセス可能にした分散共有メモリ並列計算機がある。共有メモリに即したライブラリとしては、移植性や並列性能という観点から OpenMP が主流となっている。

近年の劇的なマイクロプロセッサの高速化と低コスト化により、複数台の PC とネットワークを用いたシステム構築が広まっている。このシステムを PC クラスタと呼んでいる。現在、PC クラスタにはその冗長性を利用して高信頼システムの実現を目的とするものと、複数のコンピュータを用いて大規模システムの実現を目的とするものがある。近年、特に後者の PC クラスタの発展が目覚しく、その勢いは留まることを知らない。PC クラスタが他のスーパコンピュータと異なる点は、最低 2 台の PC があれば誰でも構築可能なことである。しかも、PC クラスタ用のフリーソフトウェアを利用すれば、ソフトウェアのコストを掛けずに構築することができる。しかし、安定した性能を実現するためには、様々な知識を必要とすることもまた事実である。

クラスタシステムの中でも注目されているのが新情報処理開発機構の中のリアルワールドコンピューティング(RWC)プロジェクトで、クラスタコンピューティングのために開発された SCore というミドルウェアを利用したクラスタシステムである。SCore は、ワークステーションや PC 等で稼動しているオペレーティングシステムで、Linux 上に構築したトータルソフトウェアであり、高性能通信を可能にする通信ライブラリを持っている。さらに Linux カーネル上に構築した SCore-D グローバルオペレーティングシステムは、クラスタシステムの構成要素であるコンピュータをすべて制御し、クラスタを単一並列コンピュータのようにすることが可能である。またソフトウェア分散共有メモリシステム(SCASH)により、分散メモリのクラスタ上で共有メモリのクラスタ上での共有メモリプログラミングを可能にしている。本研究室には Raptor という Score 型 PC クラスタがあり、SCore の SCASH により OpenMP が利用可能である。

本研究ではライフゲームに初期状態を与え、ｎ世代後までの変化を PC クラスタ上で並列実行した。ライフゲームは"0 人ゲーム"である。通常のゲームではプレイヤーの操作でその後の状態が変化していくが、ライフゲームでは初期状態のみでその後の状態が決定されるからである。碁盤のような格子があり、一つの格子はセルと呼ばれる。各セルは 8 つのセ

ルと接している。各セルには「生」と「死」の 2 つの状態があり、あるセルの次のステップ（世代）の状態は今の世代における周囲 8 つのセルの状態により決定される。本研究では 4×4、5×5 のセルでの全てのパターンを初期状態としてその後の変化を計算する。初期状態の数は 3×3 のセルでは 512 通り、4×4 では 65536 通り、5×5 では 33554432 通りになる。そして、それぞれに対して世代の変化の計算を行う。そのため計算量が爆発的に増加する。同時に計算時間も爆発的に増加する。プログラムの工夫により計算時間をある程度は小さくできるが、限界がある。しかし並列プログラミング、PC クラスタを用いることで計算時間が大幅に短縮することが可能である。

2.並列処理と OpenMP

2.1 並列プログラミング

2.1.1 並列アルゴリズム

一般的に並列アルゴリズムはプロセッサファーム(Processor Farms)、分割統治法(Divide and Conquer)、プロセスネットワーク（Process Networks)、繰り返し変換（Iterative Transformation）の 4 つに分類できる[1]。

(1)プロセッサファーム

全体の制御をマスタが行い、与えられた問題を複数の独立した計算に分割し、各スレーブまたはマスタがそれを担当する。それぞれがその与えられた計算を独立に行い、その結果をマスタに返す。そして最終的にマスタがそれをまとめ、その問題の解答を得る。

このアルゴリズムでは各スレーブまたはマスタは独立に計算を行うため、高い並列効果が得られる。

(2)分割統治法

与えられた問題についてなんらかの計算をする。そしてある条件のもとにそれを分割し、また、その計算を行う。そしてまた分割･･･と、再帰的に計算、分割を繰り返しながら解かれていく。その問題の解は分割された部分解を回収し、まとめることにより得られる。

(3)プロセスネットワーク

まずは与えられた問題について、その計算を複数のステージに分ける。入力データはあるステージ上で計算が行われると次のステージに移り、また、そこで計算が行われ次のステージに移る･･･と順番に流れていき、その各ステージは並列に実行される。

(4)繰り返し変換

与えられた問題をあるオブジェクトに分ける。各オブジェクトは複数の繰り返しの計算により値が変換され、求める値が得られる。その繰り返しはオブジェクトの値がある与えられた条件を満たすまで続けられ、また、前のステップで計算された値は自分自身または別のプロセッサ上でランダムに利用される。

2.1.2 並列計算機のメモリモデル

並列処理を行う上で、プログラミングに影響を与える要因としてメモリモデルがある[2]。主なモデルとして 3 つを挙げる。共有メモリモデル(SMP)は複数のプロセッサがメモリバス/スイッチ経由で主記憶に接続されている。共有分散メモリはプロセッサと主記憶から構成されるシステムが複数個互いに接続された形態で、プロセッサは主記憶の読み書きができる。分散メモリモデルはプロセッサと主記憶から構成されるシステムが複数個お互いに接

続された形態で、プロセッサは主記憶の読み書きができない。共有メモリモデルに即した並列プログラミングモデルとしては、本実験で使用した OpenMP が主流となっている。また分散メモリモデルに即した並列プログラミングライブラリとしては PVM(Parallel Virtual Machine)、および MPI(Message Passing Interface)が有名である。それぞれのモデルを図1、図 2、図 3 に示す。

(1)共有メモリモデル

複数のプロセッサがメモリバス/スイッチでメモリを共有し、異なるプロセッサが同じメモリ空間にアクセス可能であるため排他制御を必要とする場合がある。このアーキテクチャを有するシステムのことを SMP（Symmetrical Multi Processor）と呼ぶ。この形態の利点はメモリモデルが最も汎用でプログラムが組みやすいこと、及び並列処理だけでなく、スループットを重視するサーバマシンとしても適しているという点である。

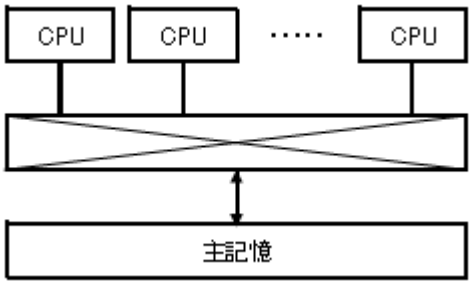


図 1：共有メモリモデル

(2)分散共有メモリモデル

プロセッサとメモリから構成されるシステムが、互いに接続された形態である。各プロセッサが全てのメモリとアクセス可能であるため、自由度が高く、大規模なシステムの構築が可能になる。しかし１つでもバリア同期の場所を間違えると、タイミング依存で非常にわかりにくいバグを作りこむことになるという欠点も持つ。

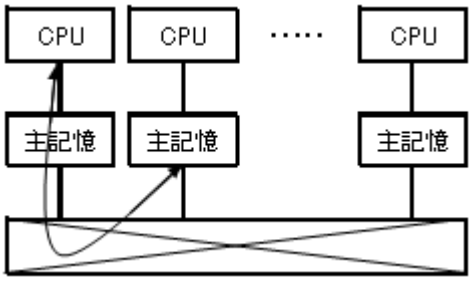


図 2：分散共有メモリ

(3)分散メモリモデル

メモリは各プロセッサにローカルに接続されているが、他のプロセッサが持つ情報にアクセスする場合、アクセス先のプロセッサの許可が必要なため、、プロセッサ間で明示的な情報交換を行わなければならない。利点は、大規模なシステム構築が可能であることと、デッドロックさえ気をつければ、タイミングに依存するバグは発生することが少ないことである。

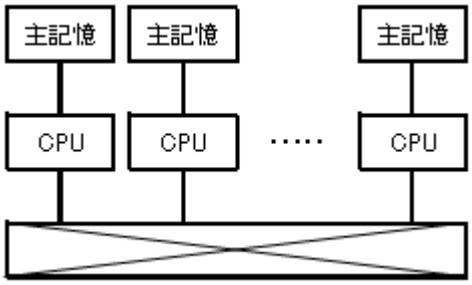


図 3：分散メモリモデル

2.2 PC クラスタ

2.2.1 PC クラスとは

PCクラスタとはPCを高速のネットワークで複数台接続し、全体としてのパフォーマンスを向上させるシステムである[3]。近年はPCの処理能力が高まり、また大幅に低価格化が進んだ。そのため非常にコストパフォーマンスに優れたシステムの構築が可能である。1990年前後に、数千から数万台のCPUを搭載する超並列計算機の開発が進む一方で、TCP/IPベースのネットワークで接続された複数台の計算機を仮想的に1台のマシンとして捉えて並列プログラミングを走らせるようなPVM(Parallel Virtual Machine)が開発された。PVMはそれぞれの計算機でメッセージ交換を行うメッセージ通信ライブラリであり、公開されたソフトウェアをインストールするだけで仮想的な並列処理環境が構築できた。これがPCクラスタの幕開けといえる。1995年前後になると、イーサネットスイッチやGigabit Etherなどの技術も普及し、比較的安価に高性能なネットワークの構築が可能となった。さらに、Myricom社のMyrinetなどのクラスタを指向した専用の高速ネットワークが登場した。これにより、専用の並列計算機並みのスループットを持つネットワークの構築が可能となった。現在では複数のプロセッサを搭載したSMP型のWSやPCが容易に入手できるようになり、これらをベースにしたSMPクラスタが主流になっている。

SCore型PCクラスタは、RWCP(Real World Compurting Project)が開発した、高性能通信ライブラリを持つSCoreと呼ばれるクラスタシステムソフトウェアを用いて構築された、専用並列マシンと同等の性能を有するクラスタシステムである。

2.2.2 本研究室の PC クラスタ

本研究室では、SCore型のRaptorというPCクラスタを使用している[4]。RaptorはCompute Hostとして8台、それらを管理するServer Hostとして1台設置する。EthernetでServerとクラスタ8台(Compute Host)を接続する。Server HostはXeon2.8GHz、メモリは2GBのSDRAMであり、Compute HostはPentiumIVプロセッサの3.2GHz、メモリは2GBのSDRAMである。Ethernetは最大帯域幅1Gbps、最大遅延60μsecであり、クラスタ同士を接続する。クラスタの構成図を図4に示す。

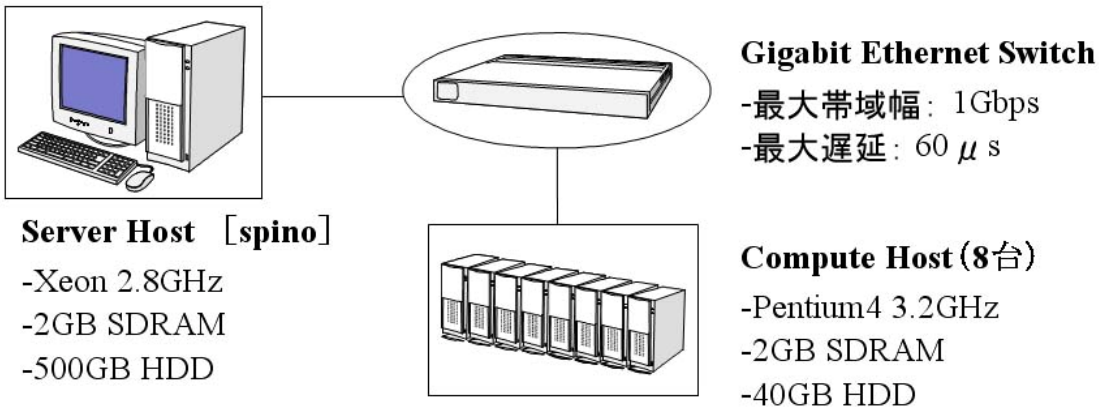


図 4：Raptor 構成図

2.3 OpenMP

2.3.1 OpenMP とは

OpenMPとは、共有メモリマルチプロセッサの並列プログラミングのためのプログラミングモデルであり、ベース言語(Fortran, C/C++)をコンパイラ指示文(directives, pragma)、ライブラリ、環境変数によって拡張したものである[5]。並列実行や同期をプログラマが明示する事により並列化を行う。また、指示文を無視する事で逐次実行が可能なので逐次版と並列版を同じソースで管理でき、段階的な並列化が可能である。なお、本研究室のPCクラスタに実装されているOpenMPは、RWCPが開発したOmni OpenMPと呼ばれるもので、FortranとCでの並列化が可能である。

2.3.2 OpenMP の実行モデル

OpenMPはfork-joinによる並列実行モデルを用いている[6]。OpenMPで記述されたプログラムは、マスタスレッドと呼ばれる単一のスレッドで実行を開始する。マスタスレッドは並列構文が現れるまで逐次リージョンを実行する。そして、並列指示文に遭遇すると複数

のスレッドからなるチームを作成し、そのチームのマスタになる。ワークシェアリング構文に対応するブロックは、全スレッドによって実行されなければならない。全てのスレッドにより並列に実行される部分を並列リージョンという。また、ワークシェアリング構文にno wait指示節が指定されていなければ、ワークシェアリング構文の最後で暗黙のバリア同期をチーム内の全スレッドに対して実行し、その後の処理はマスタスレッドのみが実行を続ける。1つのプログラムにおいて、いくつでも並列構文を使う事ができる。したがって、プログラムは実行中のforkとjoinを繰り返す事になる。図5にfork-joinモデルの実行の流れを示す。

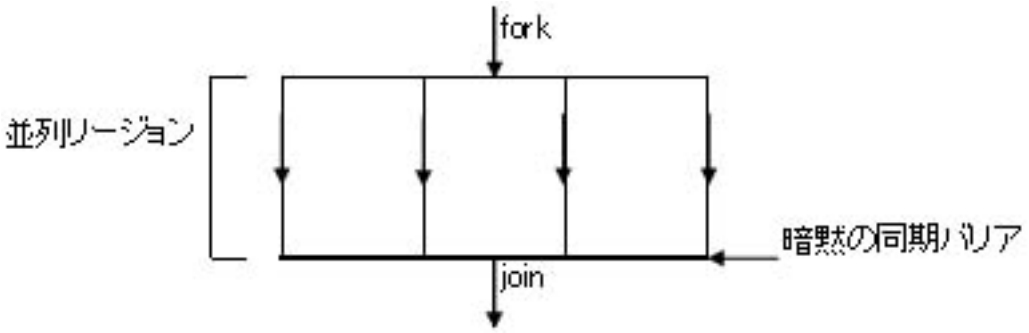


図 5：fork-join モデル

3.ライフゲームにおけるパターン探索の並列化

3.1 ライフゲームとは

3.1.1 ライフゲームの歴史

ライフゲームは 1970 年にイギリスの数学者ジョン・ホートン・コンウェイ(John Horton Conway)によって考案された生命の誕生、進化、淘汰などのプロセスを再現したゲームである[7]。生物集団においては、過疎でも過密でも個体の生存に適さないという個体群生態学的な側面を背景に持つ。セル・オートマトンの最もよく知られた例でもある。ライフゲームは、1970 年 10 月の『サイエンティフィック・アメリカン』誌のマーチン・ガードナーのコラム上で紹介されたところ多くの反響を呼んだ。興味深いことにライフゲームは万能チューリングマシンであることが証明されている。これは、ライフゲームは計算機で実行可能な全てのアルゴリズムを作ることができるということを表している。

この『サイエンティフィック・アメリカン』誌の出版後すぐに、グライダパターンと R-ペントミノパターンが発見された。これらの興味深いパターンの発見やコンピュータの普及によってライフゲームは大流行した。夜間あるいは未使用のコンピュータ上でライフゲームのプログラムが動かされることとなり、興味深いパターンが多数発見された。

3.1.2 ライフゲームのパターン

ライフゲームでは世代を経ることで最終的に死滅する図形が多い。生き延びる場合の変化は 4 パターン（固定型、振動型、移動型、繁殖型）に分類することができる。

- 固定型は世代が進んでも同じ場所で形が変わらないものを指す
- 振動型はある周期で同じ図形に戻るものを指す
- 移動型は一定のパターンを繰り返しながら移動していくものを指し、グライダーと呼ばれるものが有名である
- 繁殖型はマス目が無限であれば無限に増え続けるパターンである

図 6、図 7、図 8 に固定型、振動型、移動型の例を示す。繁殖型については変化が複雑なのでここでは示さない。

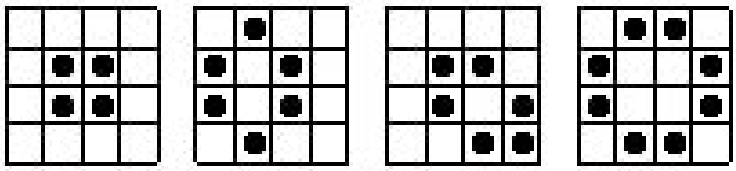


図 6：固定型

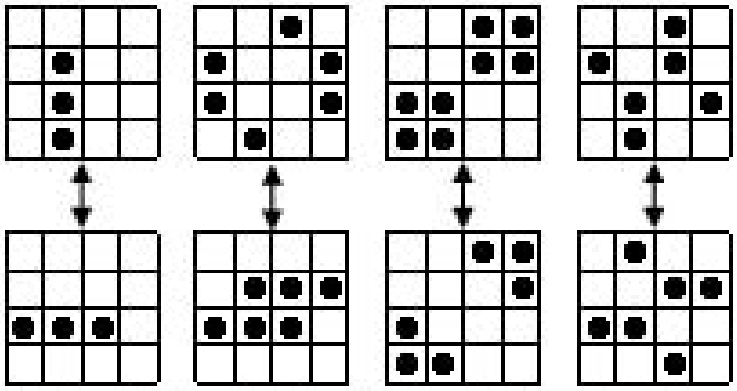


図 7：振動型

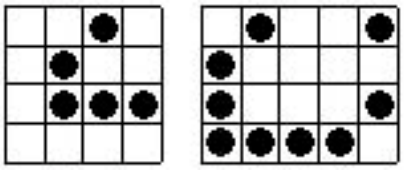


図 8：移動型

コンウェイは「無限にセルの数が増えつづけるパターンはありうるか」という問題に懸賞金をかけた。この問題は、1970 年 11 月にゴスパー(Bill Gosper)により解かれた。それは 30 世代毎にグライダーを打ち出す「グライダー・ガン」と呼ばれるパターンであった。繁殖型には他にも、本体が通過した後に破片を残していく「汽車ポッポ」(puffers)や、宇宙船が集まって移動しながらグライダーを発射していく「宇宙編隊」と呼ばれるものを含め様々なパターンが見つかっている。

3.2 パターン探索

4×4 のセルに対して総当りの初期状態を作成する。そのすべてに対して n 世代後の生存数別のパターン数を検出するプログラムの速度向上を比較する。生存数の大きいものやパターン数の大きいものに注目することにより、美しい振動型など興味深い変化をするパターンの発見につながる。

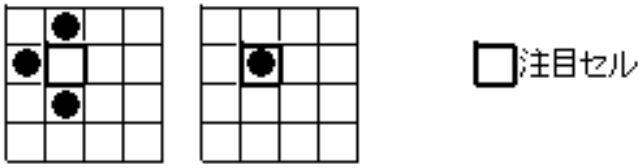
3.3 アルゴリズム

3.3.1 ライフゲームのアルゴリズム

セルの生死は次のルールに従う。基本的な考えは「過疎状態でも過密状態でも生き残ることはできない」というものである。

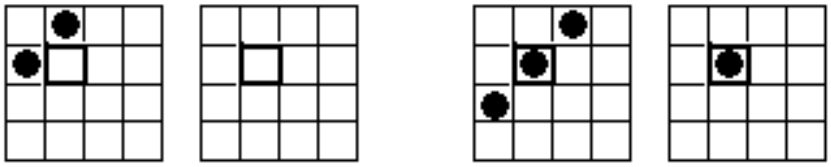
・誕生:周囲に 3 つの生きているセルがあれば次の世代では生きる(誕生する)

・維持:周囲に 2 つの生きているセルがあれば次の世代は前世代の状態を維持する
・死亡:上以外の場合には次の世代では死ぬ
図 9、図 10、図 11 に誕生、維持、死亡の様子を示す。
図 12 に世代の進行と状態の変化、図 13 に注目セルの移動順序を示す。



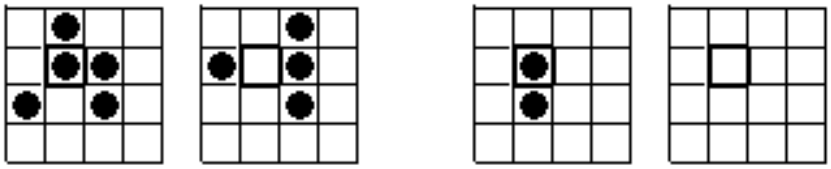
0 世代目　　1 世代目

図 9：誕生



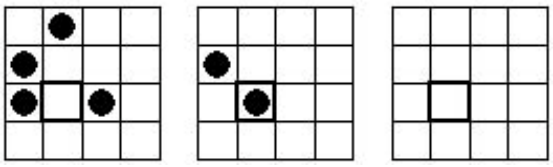
0 世代目　　1 世代目　　0 世代目　　1 世代目

図 10：維持



0 世代目　　1 世代目　　0 世代目　　1 世代目

図 11：死亡



0 世代目　　1 世代目　　2 世代目

図 12：世代の進行と状態の変化

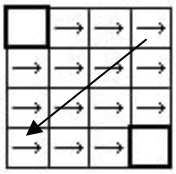


図 13：注目セルの移動順序

3.3.2 パターン探索のアルゴリズム

4×4 のセルに初期状態を作成する場合、全部で 65536 通りになる。0 番から 65535 番生の状態を 1、死の状態を 0 と考え、初期状態を作成する。図 14 に総当りの作成手順を示す。

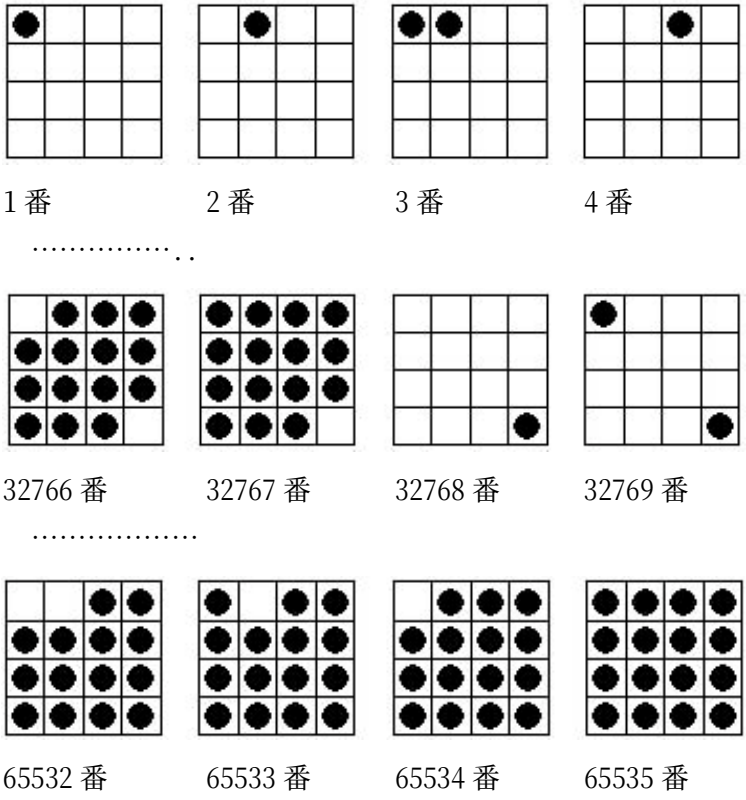


図 14：総当りの作成手順

3.4 並列化手法

4×4 のセルから初期状態を発生させると、総パターン数は 65536 通りである。CPU8 台で実行する場合、図のように 0 から 65536 番目までのデータ DATA0 をブロック分割で 8 つに区切り、それぞれを CPU に割り当て並列化している。割り当てられた番号からそれぞれの CPU が初期状態を生成し指定した n 世代後の状態まで処理を進行する。本実験ではそこから生存数を DATA1 として抽出した。その他に、変化が固定するパターンや同じ変化を繰り返すループと呼ばれるパターンの抽出も可能である。また何か条件を与えて実際の状態を抽

出することも可能である。図 15 にブロック分割の様子を示す。

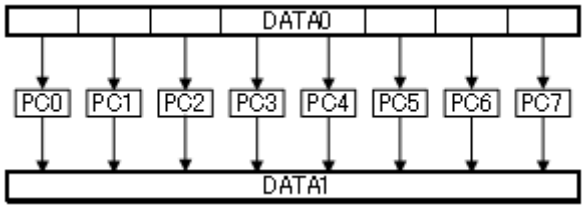


図 15：ブロック分割

4.実験

4.1 実験条件

4×4 のセルに初期状態を発生させ、50×50、および 100×100 のセルの中心に配置した。それぞれ 25 世代、50 世代、100 世代、200 世代まで進行させ、生存数のパターン数を検出した。Raptor クラスタにおいて、それぞれ 1 台実行、2 台実行、4 台実行、8 台実行を行い、速度向上を比較した。

4.2 実行結果

50×50 のセルにおける実行時間、速度向上比をそれぞれ表 1、表 2 示す。速度向上比のグラフを図 16 に示す。50×50 のセルでの 25 世代後、50 世代後、100 世代後、200 世代後における生命数別のパターン数のグラフをそれぞれ図 17、図 18、図 19、図 20 に示す。

100×100 のセルにおける実行時間、速度向上比をそれぞれ表 3、表 4 示す。速度向上比のグラフを図 21 に示す。100×100 のセルでの 25 世代後、50 世代後、100 世代後、200 世代後における生命数別のパターン数のグラフをそれぞれ図 22、図 23、図 24、図 25 に示す。

表 1：50×50 のセルにおける実行時間

50×50

世代数	1 スレッド	2 スレッド	4 スレッド	8 スレッド
25	72.67	37.17	19.36	9.95
50	137.49	70.49	36.63	18.38
100	267.84	136.50	71.55	35.81
200	529.75	270.11	140.76	71.08

(単位：秒)

表 2：50×50 のセルにおける速度向上比

50×50

世代数	1 スレッド	2 スレッド	4 スレッド	8 スレッド
25	1.00	1.95	3.75	7.31
50	1.00	1.95	3.75	7.48
100	1.00	1.96	3.74	7.48
200	1.00	1.96	3.76	7.45

(単位：倍)

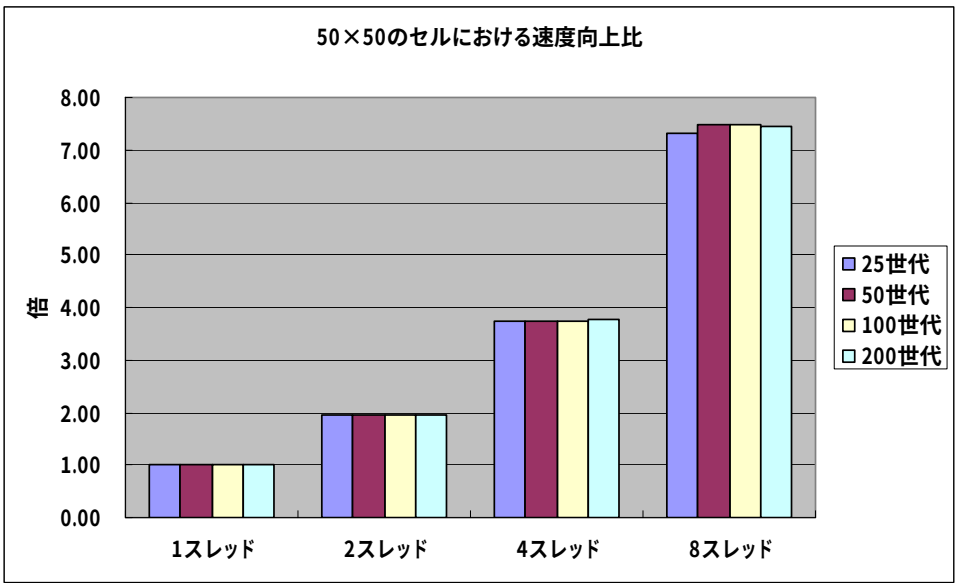


図 16：50×50 のセルにおける速度向上比

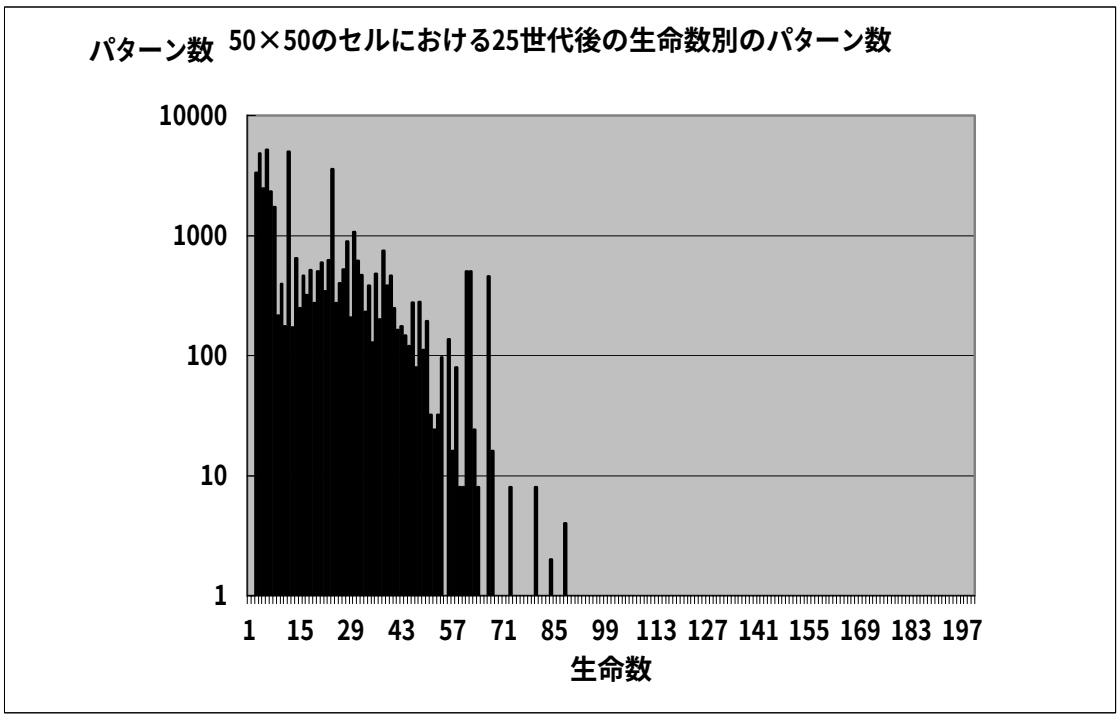


図 17：50×50 のセルでの 25 世代後における生命数別のパターン数

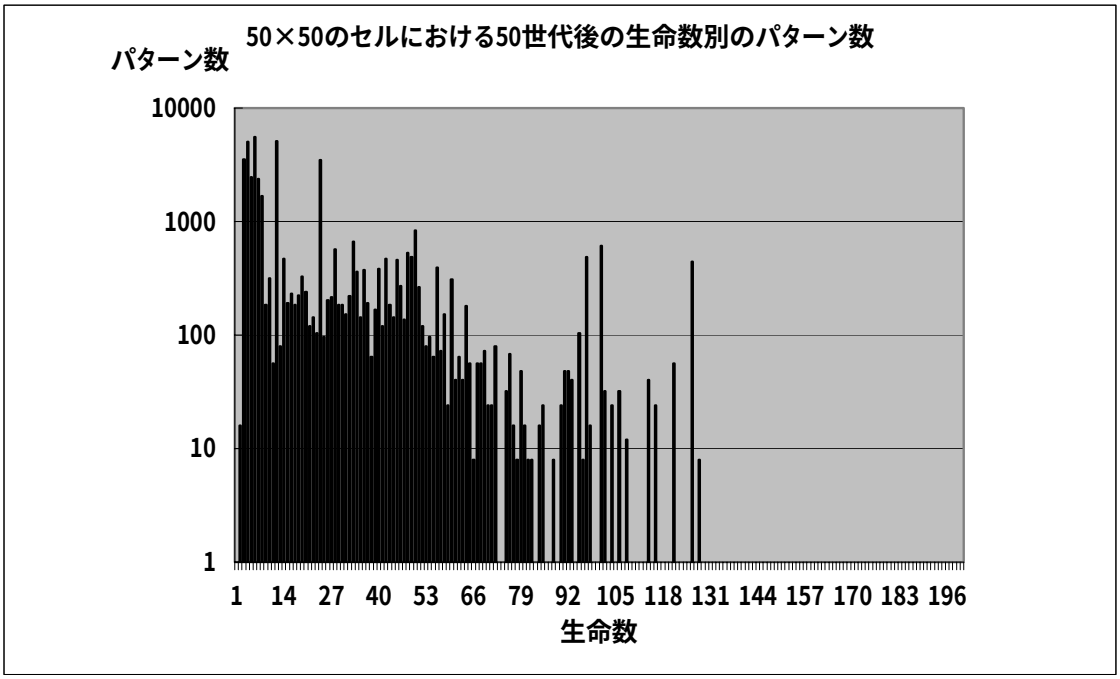


図 18：50×50 のセルでの 50 世代後における生命数別のパターン数

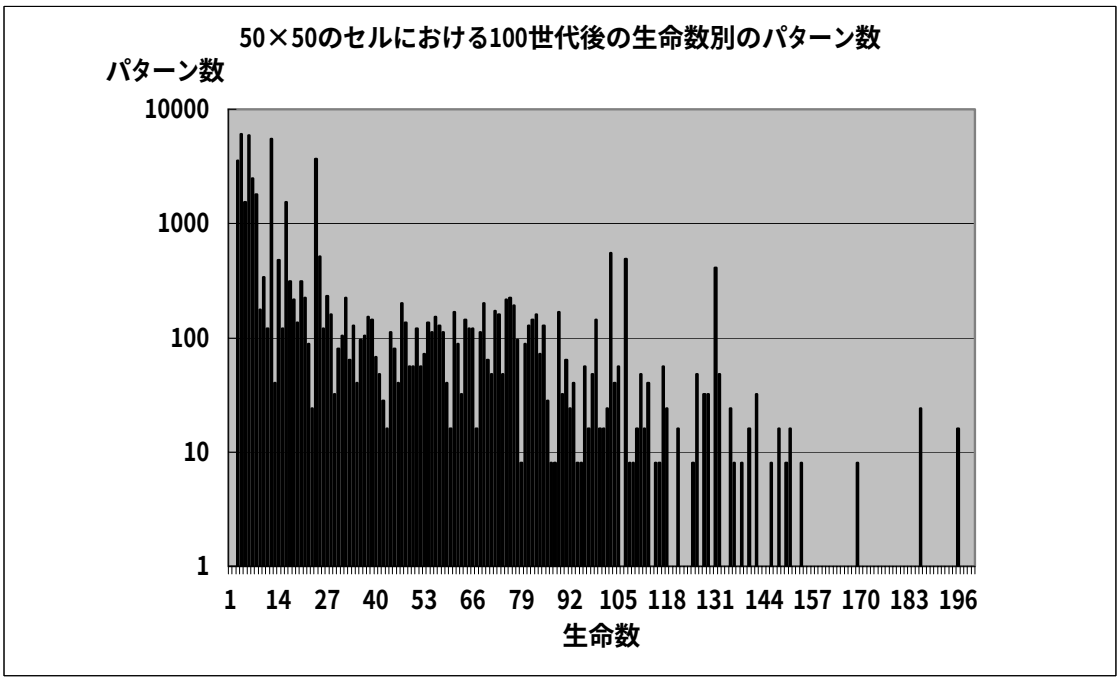


図 19：50×50 のセルでの 100 世代後における生命数別のパターン数

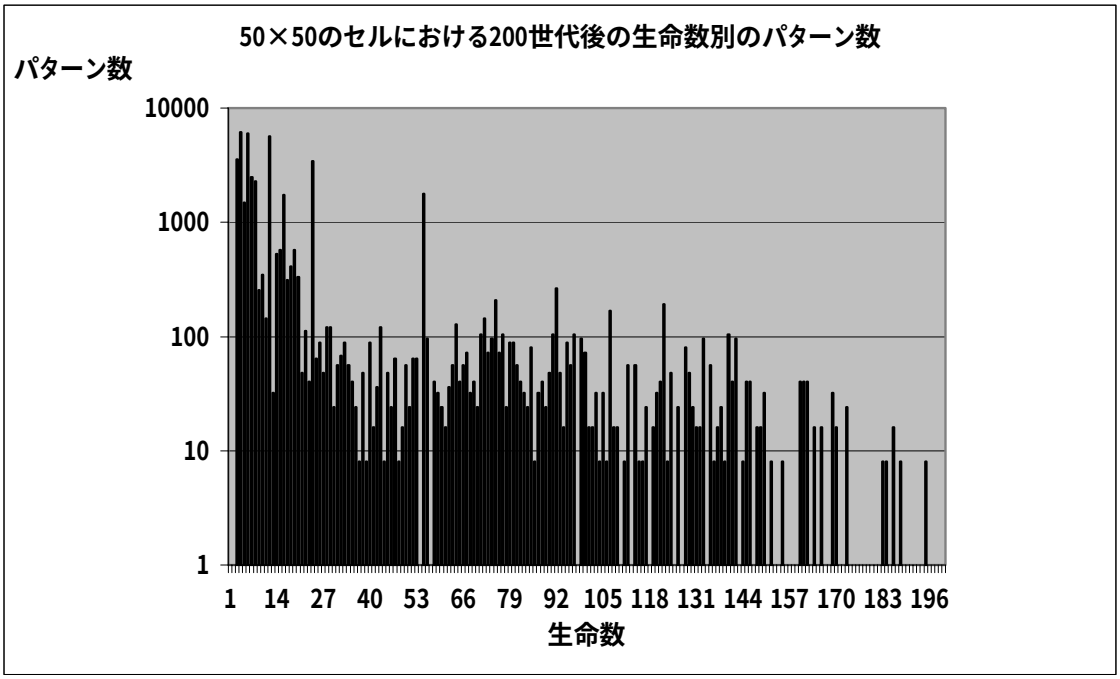


図 20：50×50 のセルでの 200 世代後における生命数別のパターン数

表 3：100×100 のセルにおける実行時間

100×100

世代数	1 スレッド	2 スレッド	4 スレッド	8 スレッド
25	276.77	141.70	73.64	36.68
50	519.49	266.07	138.60	69.17
100	1003.40	513.20	267.03	133.75
200	1974.02	1009.71	525.77	263.47

(単位：秒)

表 4：100×100 のセルにおける速度向上比

100×100

世代数	1 スレッド	2 スレッド	4 スレッド	8 スレッド
25	1.00	1.95	3.76	7.55
50	1.00	1.95	3.75	7.51
100	1.00	1.96	3.76	7.50
200	1.00	1.96	3.75	7.49

(単位：倍)

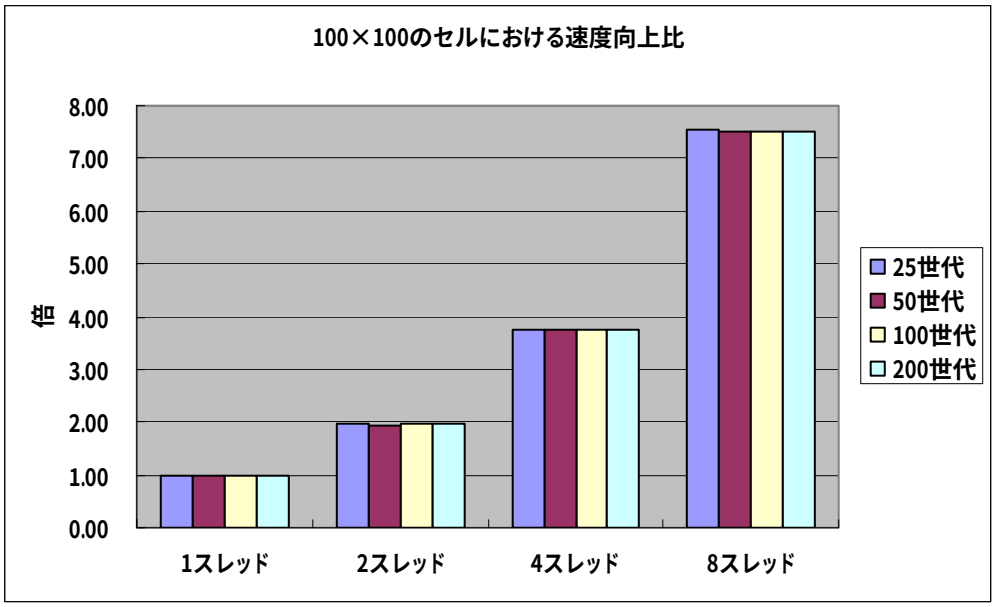


図 21：100×100 のセルにおける速度向上比

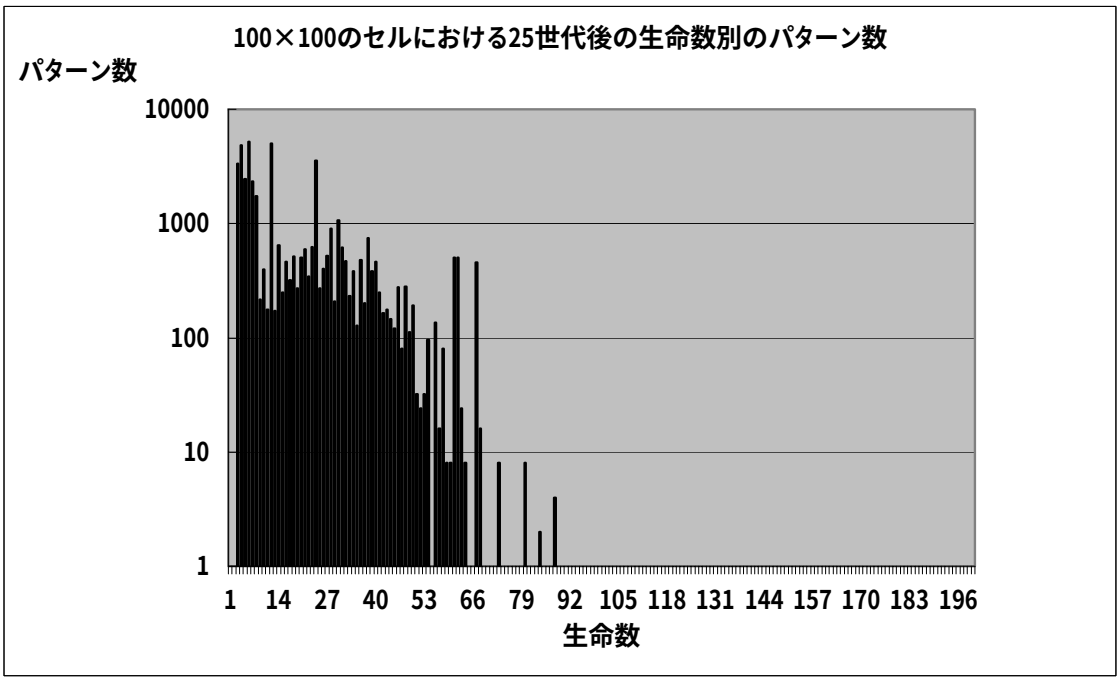


図 22：100×100 のセルでの 25 世代後における生命数別のパターン数

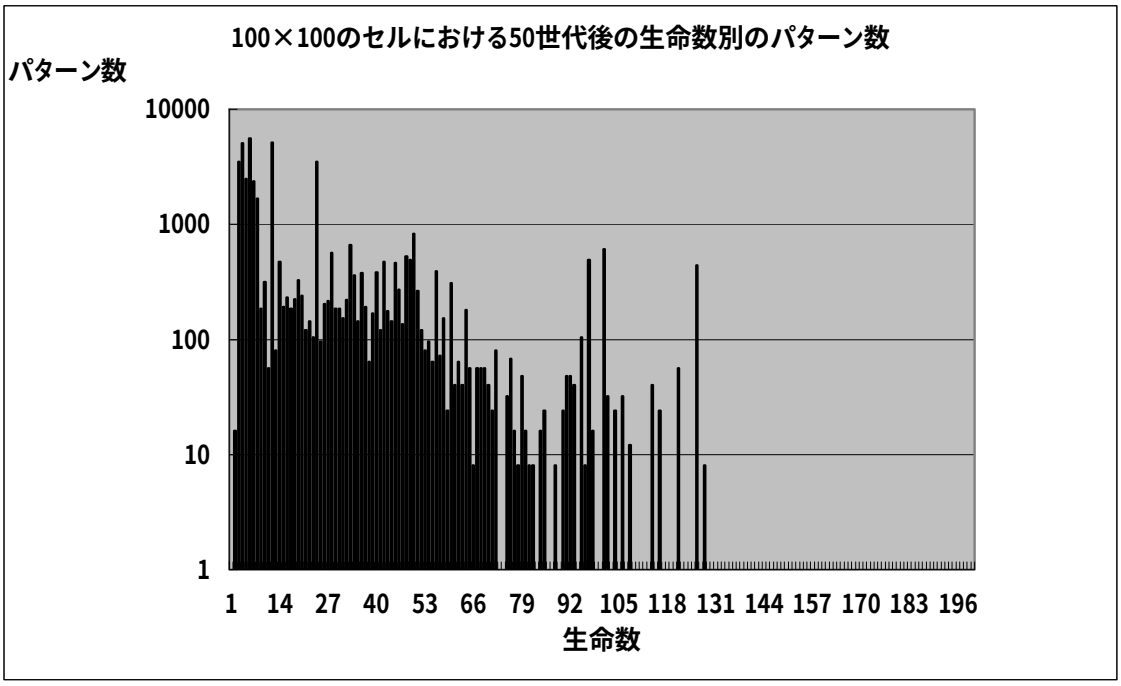


図 23：100×100 のセルでの 50 世代後における生命数別のパターン数

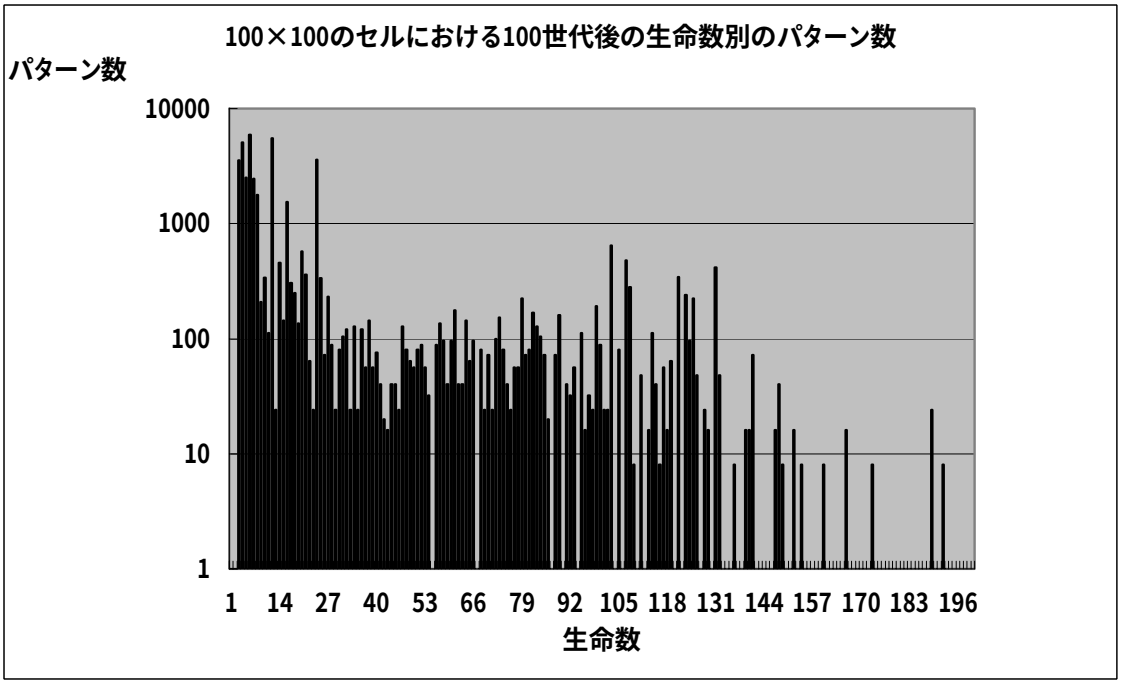


図 24：100×100 のセルでの 100 世代後における生命数別のパターン数

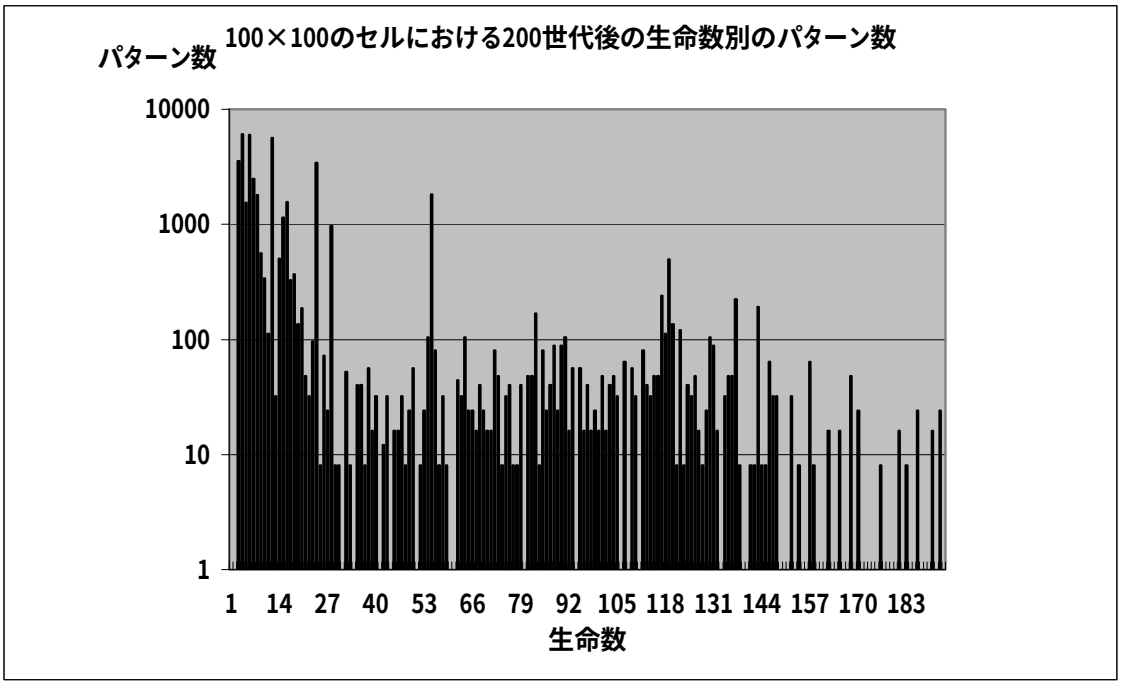


図 25：100×100 のセルでの 200 世代後における生命数別のパターン数

実行時間は、1 スレッド実行と比べて 2 スレッド実行で約 1.95 倍、4 スレッド実行で約 3.75 倍、8 スレッド実行で約 7.5 倍の速度向上を確認した。8 スレッド実行に注目する。すると 100×100 のセルで実行した時が 50×50 のセルで実行した時よりも僅かに速度向上比が大きいことがわかる。

4.3 考察

50×50 と 100×100 の実行時間を比べてみると、100×100 の実行時間は 50×50 の約 3.75 倍になっている。セルの面積は 100×100 が 50×50 の 4 倍である。このことから初期状態を発生させる処理よりも世代を進行させる処理に多くの時間が使われていることがわかる。

通常、データサイズが大きくなれば、実際の処理にかかる時間が、データの分割やプロセッサ間の通信等にかかる時間に対して大きくなるので、並列化効果が高まる。しかし本実験では 100×100 のセルで実行した時に 50×50 のセルで実行した時よりも僅かに並列効果が高まることが確認できただけである。50×50 のセルでも十分にデータサイズが大きいために、はっきりとした並列化効果の高まりを確認できなかったのではないかと考える。

試験として、状態が固定したパターン、ループしているパターンの世代進行を終了する処理を付加し、プログラムを並列実行した。この処理は特徴的な変化をする初期状態の抽出に有効である。試験では25世代、50世代において実験よりも実行時間が大きくなった。しかし世代を大きくし100世代、200世代では実験より実行時間が小さくなることを確認し

た。また世代数を大きくしても実行時間が実験結果のように比例的に大きくならないことを確認した。このことから実験では無駄な処理を実行終了まで行っていることがわかる。またそのことから安定した実行時間になっているとも考えられる。50×50のセル、200世代の試験において、実行時間が約52秒、固定するパターンが約25000件、1回のループが約12000件、と大きく、2回のループが約100件と小さいことを確認した。この結果は高速な処理を行うことの参考になる。

5.おわりに

本研究では、本研究室のSCore型PCクラスタ上でOpenMPによってライフゲームにおけるパターン探索を並列化し、実行した。その結果、100×100のセルにおいて、最高の約7.5倍の並列化効果を得られた。

今後の課題としては実際に特徴的な変化をするパターンを抽出したい。また,より特徴的なパターンが発生すると考えられる、5×5、6×6のセルで初期状態を発生させたい。ただし初期状態の件数は爆発的に大きくなる。そのため高速な処理を行うプログラムの作成が必要である。SCoreのバージョンがアップされるとともに、OpenMPのコンパイラや、SCASHの性能も良くなっていけば、PCクラスタ上でのOpenMPによる並列プログラミングは今後ますます実用化されると思われる。

謝辞

本研究の機会を与えてくださり、数々の助言をいただきました山崎勝弘教授に心より感謝いたします。また本研究にあたり、親切な指導や貴重なご意見をいただきました池上広済様、梅原直人様、加藤寛暁様、松崎裕樹様をはじめ本研究室の皆様に心より感謝いたします。

参考文献

[1] 山崎勝弘：事例ベース並列プログラミングの研究、平成 9 年度から平成 10 年度科学研究費補助金、基盤研究(C)(2)研究報告書、1999.

[2]湯浅太一、安村通晃、中田登志之：bit別冊　はじめての並列プログラミング、共立出版、1998.

[3]池上 広済：PCクラスタ上でのOpenMPによるJPEGエンコーダの並列、立命館大学理工学部情報学科卒業論文、2003.

[4]林雅樹、池上広済：PCクラスタの使用法とOpenMP並列プログラミング、資料、2005.

[5]佐藤三久：OpenMP、JSPP99 OpenMP チュートリアル資料、1999.

[6]大村浩文：PC クラスタの動作テストと OpenMP 並列プログラミング、立命館大学理工部情報学科卒業論文、2000.

[7]フリー百科事典『ウィキペディア（Wikipedia)』http://ja.wikipedia.org/

[8]バリーウイルキンソン、マイケルアレン(著)、飯塚肇、緑川博子(訳):並列プログラミング入門―ネットワーク結合 UNIX マシンによる並列処理、丸善、2000.

[9]ライフゲーム保存会日本関西支部
http://www.mars.dti.ne.jp/~khnum/life/index.html

[10]さわぼう宅 http://park7.wakwak.com/~sawaboh/program/lifegame/index2.html