Cell B.E. 上における PXCA モデルの実装

政光寛人

平成 21 年 2 月 24 日

目次

図 目 次

表 目 次

第1章　序章

1.1　はじめに

現代生物学において、遺伝子などに関するゲノムレベルでの解析、そしてそれに基づく知識の進歩は著しい。ヒトゲノムの解析作業は 2003 年に完了しており、その他多くの動物のゲノム塩基配列に関しても解析が行われ続けている。遺伝子の配列情報を基にリボ核酸 (RNA) を転写、その情報を元にタンパク質を合成することにより細胞構造となるが、こうして出来上がった細胞が図 1.1 に示すようにいかに他の遺伝子により生成された細胞と協調して多細胞からなる組織となっていくのか、また、どのように細胞同士が化学的、および力学的に作用しあい細胞間における秩序を作り上げていくのか、そういった点についてはまだまだ不明な点が多く、解明は進んでいないのが現状である。これは多細胞構築物は各要素が互いに影響を与え合う複雑系であり、従来の手段による解析が難しいのが一因である。この複雑系の振る舞いをとらえる手段として有効なのが、コンピュータを用いた実時間シミュレーションである。各細胞についてモデル化し、コンピュータ上で観測、およびユーザが介入してシミュレーションに影響を与えることが可能ならば、細胞の組織化の過程を解明する一助になりうる。

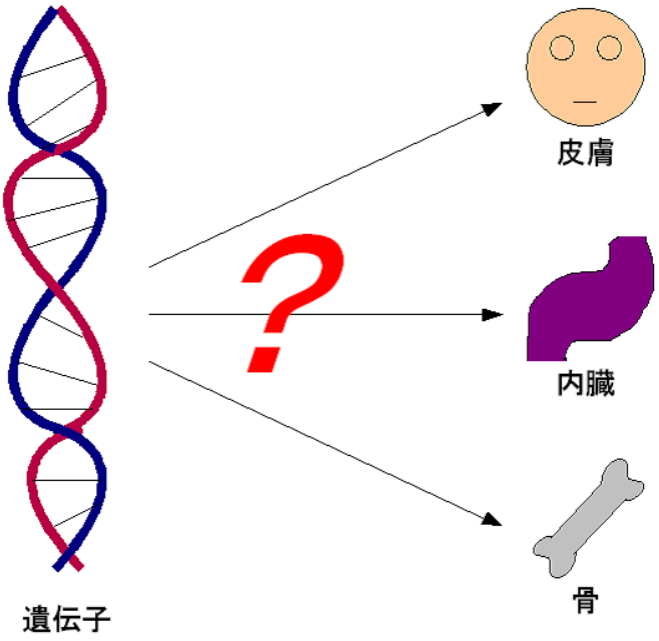

図 1.1: 遺伝子が多細胞組織化していく過程の概念

1.2　細胞モデル

細胞は生物の最も基本的な最小単位であり、あらゆる生物は細胞の集合として成り立っている。一つ一つの細胞の行う振る舞いは多様であるが、主な振る舞いとして、

- 細胞分裂
- 代謝
- 移動
- 組織化
- 死滅

などといったものがあり、それぞれが独自の生命活動を行っている。これを数学的に各動作を定義し、計算で細胞の動作をシミュレーションすることを可能にするのが細胞モデルである。計算機によるシミュレーションにおいては、図 1.2 のように各細胞をクラス及び構造体として定義し、細胞のふるまいを関数として規定、それぞれの細胞クラスについて並列処理を行うことで効率的なシミュレーションを可能にする。

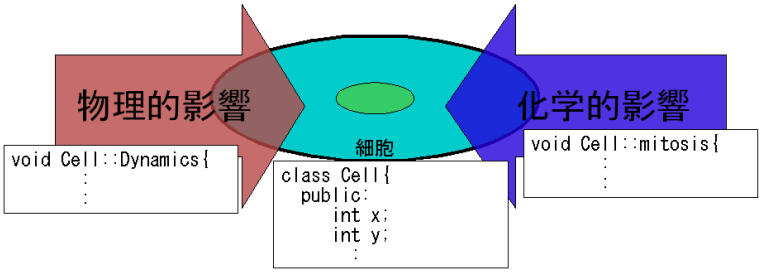

図 1.2: 細胞モデルの概念

1.3 PXCA モデル

PXCA(Pixel Cluster Automata) モデルは有明工業高等専門学校の松野哲也准教授が提唱する細胞指向モデルである [1]。PXCA モデルにおいては細胞は図 1.3 のようにピクセルの集合体として表され、各ピクセルの状態は周囲の細胞の様子によって決定されるセルオートマトンに近いものとなっている。各ピクセルは 2 つの同種細胞が接続していることを示す接続因子、異なる細胞が触れていることを示す接着因子、いずれかの細胞に属している状態、あるいはこれらのどれにも当てはまらない空白状態に決定される。このピクセルの状態を決定する計算式によって、さまざまな細胞の特性を実装することができる。しかし実際の細胞をシミュレートするに際し、考えられる特性を実装していくと計算量が膨大になってしまう。従って、PXCA モデルを用いて実用的なシミュレーションを構築するためには高速で大規模な計算が可能な環境が必要となる。

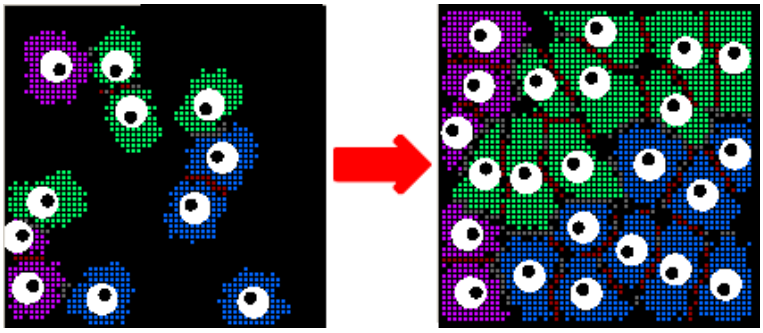

図 1.3: PXCA モデルによる細胞シミュレーションの様子

セルオートマトン

セルオートマトンとは格子状に存在するセルの状態をある規則に従って決定するモデルである。セルはいくつかの内部状態を持ち、図 1.4 のように時刻 t におけるセルの状態やその周辺セルの状態などを元に決められた規則に従って時刻 $(t+1)$ における新たな状態を決定する。セルオートマトンとして最も有名なものに、1970 年にジョン・コーウェンによって考案されたライフゲームがある。ライフゲームにおいて、二次元平面に網目状に広がったセルはそれぞれ「生」と「死」の 2 状態を持ち、以下のルールによって次の時刻の状態が決定される。

- 死んでいるセルの周辺の 8 つのセルのうち 3 つが生きていれば、そのセルは生きる (誕生)
- 生きているセルの周辺の 8 つのセルのうち 2 つもしくは 3 つが生きていれば、そのセルは生きる (維持)
- 上の 2 つのルールに当てはまらないセルはすべて死ぬ (死亡)

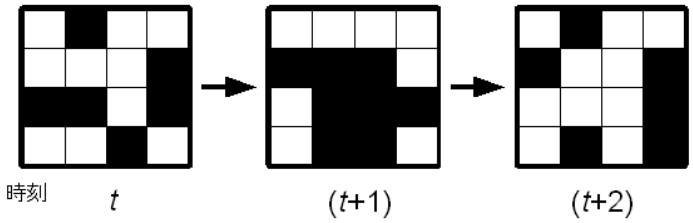

図 1.4: ライフゲームの進行の様子

1.4 並列コンピューティング

並列コンピューティングとは、一つの処理を複数のプロセッサに分割して同時に処理させる事を指す。多数のプロセッサに同時に処理をさせる事で、高速な処理が可能となる。近年では従来より多数のプロセッサを搭載していたスーパーコンピュータのみならず、パーソナルコンピュータに搭載する CPU においても複数のプロセッサを搭載したものが多く登場しており、並列コンピューティングの重要性はより高まってきている。

各プロセッサに分配されたタスクに実行順序などの制約があると、それらを並列実行することが難しくなる。したがって、単純に並列化すれば速度が上がるという訳でもなく、逆に並列化処理のオーバーヘッドにより遅くなることさえある。よって、並列コンピューティングを行うためには、並列化の種類に応じて処理方法などを慎重に考える必要がある。主に並列化手法として以下のようなものがある。

- 分散メモリ並列
- 共有メモリ並列

1.4.1 分散メモリ並列

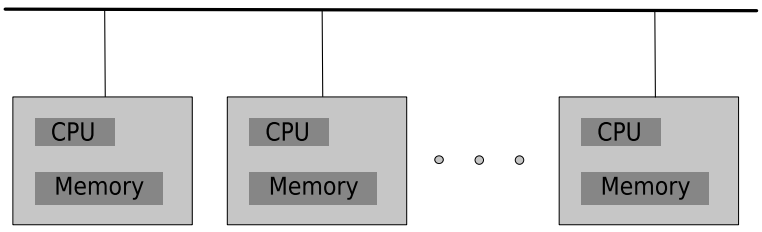

図 1.5: 分散メモリ並列

図 1.5 のように、CPU-メモリが一組になったものが、複数個接続された形態を分散メモリ並列という。基本的に、CPU は別の組のメモリを参照することはできず、全体を接続しているバスは CPU-メモリ間のデータ転送速度に比べて低速であ

る。従って、頻繁に情報をやりとりすると、バスの速度がボトルネックとなり十分な高速化が行われなくなる。しかし、CPU-メモリを追加していく事により容易に全体の処理能力を高めることが出来る点で後述する共有メモリ並列よりも優れている。また、大容量のメモリを使用できる点でも優れている。

1.4.2 共有メモリ並列

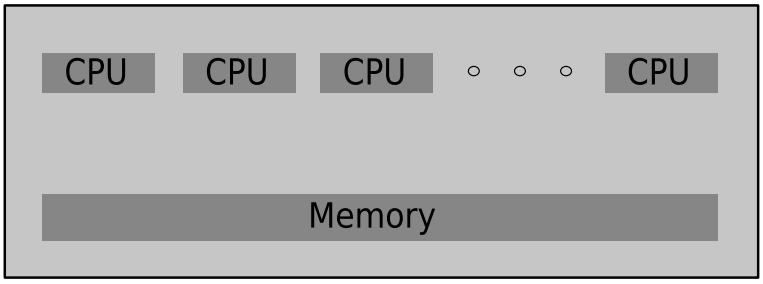

図 1.6: 共有メモリ並列

図 1.6 のように、複数の CPU が一つのメモリを共有する形態を共有メモリ並列という。情報の共有が容易であり、プログラミングは比較的簡単である。しかし、メモリアクセスの排他処理を正しく行わなければ処理結果の一貫性は保たれなくなる。CPU を増やしたりメモリを増量し規模を大きくすることで、ある程度までは処理能力が向上する。しかし、次第にメモリアクセスの排他処理のオーバーヘッドが大きくなってしまい、処理能力が向上しにくくなる。

1.5 Cell/B.E.

Cell/B.E.(Cell Broadband Engine) は、ソニー、IBM、東芝の 3 社が共同開発した CPU である。ヘテロジニアスマルチプロセッサ (異なる種類のプロセッサを複数搭載した CPU) であり、2006 年に発売された PlayStation3 の CPU として搭載されたモデルでは 1 つの汎用プロセッサである PPE(PowerPC Processor Element) と 8 つの計算用に特化されたシンプルなプロセッサである SPE(Synargistic Processor Element) を持つ [2][3]。但し、PlayStation3 が利用できる SPE はそのうちの 7 個であり、さらに Linux を利用する際に PlayStation3 の OS 側で SPE を 1 つ占有するため、ユーザが利用できる SPE は 6 個に制限されている。構成の概要を図 1.7 に示す。

SPE は 256KB の LS(Local Store) という独自のメモリ領域を持ち、メインメモリ間とは DMA(Direct Memory Access) 転送によりデータの受け渡しをする分散メモリ並列設計がなされている。多数のプロセッサを持ったマルチコア CPU を持つ環境としては入手・構築が容易であり、比較的低コストで導入が可能である。

PlayStation3 では Linux が利用できるため、Linux 上で動作するプログラムは基本的にそのまま動作するが、その場合は図 1.8 に示すように汎用プロセッサである

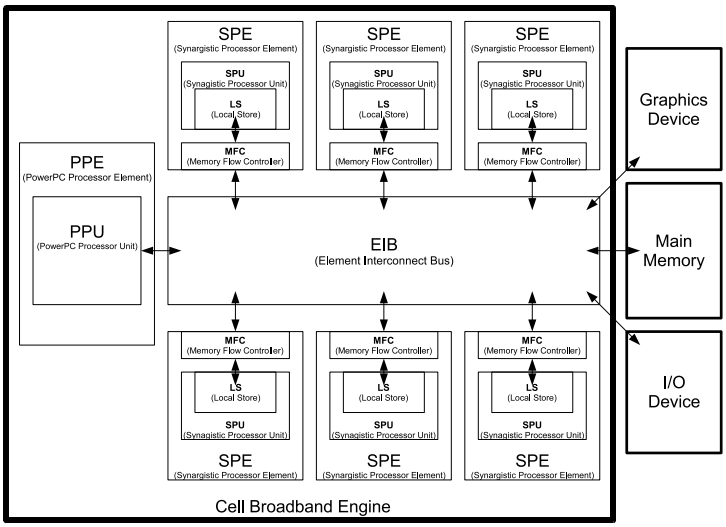

図 1.7: Cell/B.E. の構成概要

PPE のみが動作し SPE は全く動作しておらず、Cell/B.E. の真価を発揮できない。Cell/B.E. の性能をフルに発揮させるためには、

- SPE に処理を効率良く割り振る
- 演算を高速な SIMD 演算に置き換える
- メインメモリ-SPE 間のデータ転送を効率良く行う

といった要素をユーザ側が留意してプログラムを製作する必要がある [4][5][6]。

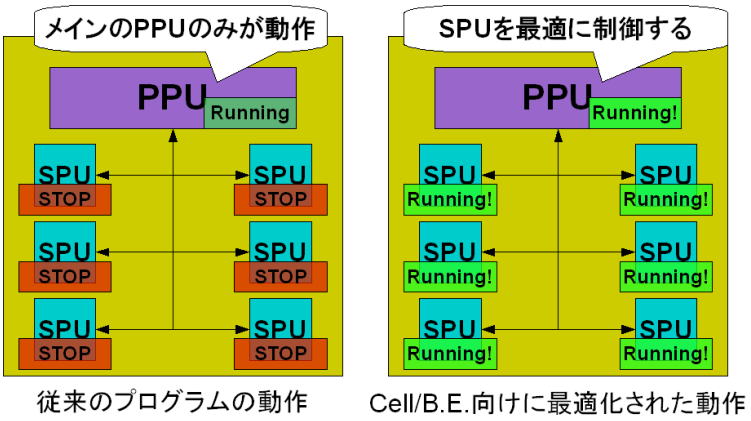

図 1.8: Cell/B.E. 動作の効率化の概念

SIMD 演算

SIMD(Single Instruction Multiple Data) 演算とは、1 回の命令で複数データに対する処理を同時に行う演算である。Cell/B.E. においては、メモリ上に連続して存在する 4 つの値について SIMD 演算を行うことができる。これらの連続した値を 4 値のベクトルとして考えると、例えば、

$$
\begin{aligned}
1 + 5 &= 6 \\
2 + 6 &= 8 \\
3 + 7 &= 10 \\
4 + 8 &= 12
\end{aligned}
$$

のような 4 回の加算処理は、

$$
\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} + \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \end{pmatrix} = \begin{pmatrix} 6 \\ 8 \\ 10 \\ 12 \end{pmatrix}
$$

という 1 回の加算処理に置き換えることができる。Cell/B.E. は SIMD 演算に特化した演算論理装置で構成されているため、可能な限りプログラム中の各種演算を SIMD 演算に置き換える事によりプログラムの高速化が期待できる [7]。

1.6 目的

前述の PXCA モデルを用いることで、計算機上で多細胞の活動の様子を扱うことが可能である。しかし、実在の生物のように非常に多くの細胞の組織化のシミュレーションを行う場合、非常に大きな領域について計算を行う必要があり、各種処理における計算量も膨大である。特にユーザがシミュレーションを実時間で観測する、もしくはリアルタイムでシミュレーションの状態に介入する必要のある場合、大規模な計算を高速に処理することが可能なプロセッサが求められる。Cell/B.E. は並列処理により非常に高いパフォーマンスを発揮するプロセッサであり、その入手の容易性とコスト面の低さ、そして細胞の動作の並列性から、PXCA モデルを用いた多細胞組織化シミュレーションに向いていると考えられる。

そこで実際に PXCA モデルによるシミュレーションプログラムを Cell/B.E. 向けに実装しそのプログラムの最適化を行い、動作時間や大規模化について調査、その有用性を検討する事を目的とする。

第2章　マンデルブロ集合

Cell/B.E. における計算の最適化の実装例として、マンデルブロ集合を計算するプログラムの実装について示す。

2.1　マンデルブロ集合

マンデルブロ集合とは、以下の数式で定義される複素数列 $\{z_n\}_{n\in\mathbb{N}}$ が $n \to \infty$ で無限大に発散しないような複素数 c の集合である。

$$\begin{cases} z_{n+1} &= {z_n}^2 + c \\ z_0 &= 0 \end{cases}$$

また、$z_n = a_n + ib_n$、$c = x + iy$ とおくと、以下の関係が成立する。

$$\begin{cases} a_{n+1} &= {a_n}^2 - {b_n}^2 + x \\ b_{n+1} &= 2a_n b_n + y \end{cases}$$

このとき xy 平面上の点 (x, y) について各点が無限大に発散するまでの速度を求め、その速度によって平面上の各点を色分けすると、図 2.1 のようなフラクタル図形が得られる。

図中の黒い部分がマンデルブロ集合を示している。

マンデルブロ集合の計算はプログラムによる実装が比較的容易であり、また計算領域の解像度の大きさ、および発散するまでの速度 n の最大値を大きくすることにより計算量が膨大になるため、各種コンピュータの性能の比較に有用である。

2.2　マンデルブロ集合計算の並列化

マンデルブロ集合を計算するのに必要な情報は xy 平面上における位置情報のみであり、他の情報を必要としない。従ってデータ間に依存関係がなく、処理の並列化に非常に向いているといえる。Cell/B.E. 上での計算においては、図 2.2 のように計算する xy 平面の領域を計算に使用する SPE にそれぞれ分割して割り当て、各 SPE において SIMD 演算を用いて同時に 4 点の計算を行う事により最も効率的な並列化が行えると考えられる。

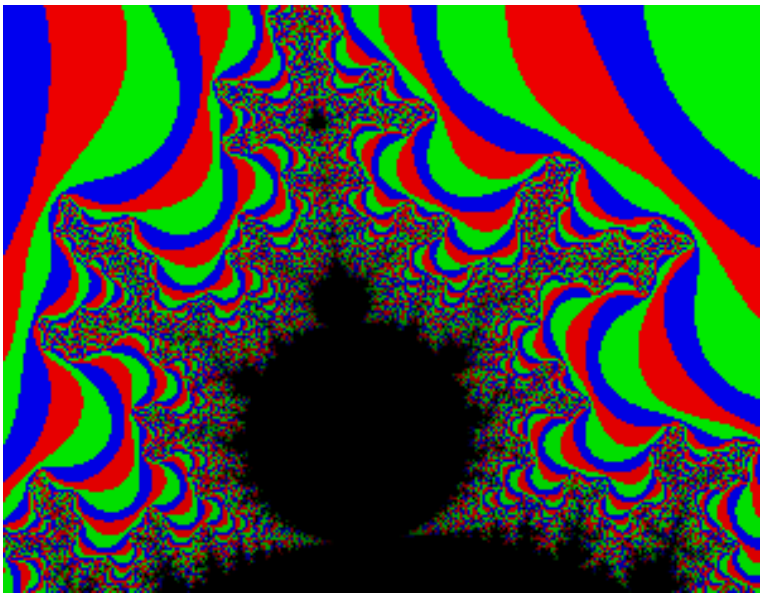

図 2.1: マンデルブロ集合

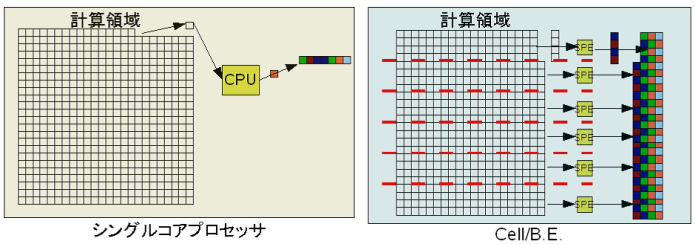

図 2.2: マンデルブロ計算の概念の対比

2.3 実行時間

以下の CPU を用いた環境でマンデルブロ集合を計算するプログラムを動作させ、その実行時間の比較を行った。

- Cell/B.E. 1PPE、SIMD 演算を使用しない
- Cell/B.E. 1PPE、SIMD 演算を使用する
- Cell/B.E. 1PPE+6SPE、SIMD 演算を使用する

プログラムのコンパイルには gcc 4.1 および SPE 向けの C コンパイラである spu-gcc 4.1 を利用した。解像度は 960×960 とし、各点における発散の速度を示す n の最大値は 1024 とした。

以上の条件で各プログラムを実行したときの各実行時間を表 2.1 および図 2.3 に示す。

表 2.1: マンデルブロ計算の実行時間

実行環境	実行時間 [単位:sec]
Cell/B.E. 1PPE、SIMD 演算を使用しない	17.906
Cell/B.E. 1PPE、SIMD 演算を使用する	9.067
Cell/B.E. 1PPE+6SPE、SIMD 演算を使用する	0.591

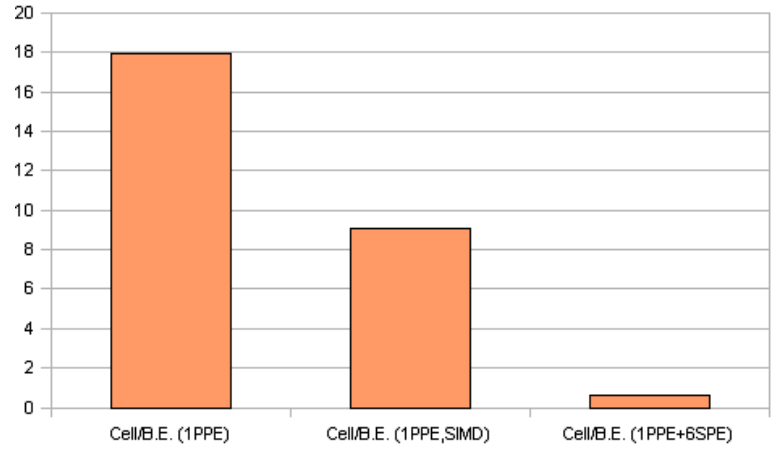

図 2.3: マンデルブロ計算の実行時間

PPE において SIMD 演算を使用したときの実行時間は使用しない場合のおよそ 1/2 であり、また計算処理を 6 個の SPE に割り振ったときの実行時間は SPE を使用しない場合のおよそ 1/18 となった。以上の結果より、Cell/B.E. は SIMD 演算を用いて SPE に処理を効率的に割り振ることにより高い性能を発揮することがわかる。

第3章　プログラム

3.1　プログラム処理の概要

PXCA モデルによるシミュレーションプログラムにおいては、主に 2 つの処理をループさせる。プログラムの流れの概念を図 3.1 に示す。

各細胞の更新処理

各細胞について、前述した PXCA モデルの性質に基づいた更新処理を行う。各細胞はピクセル場に存在する自細胞所属ピクセルの位置情報や周辺細胞との接続因子などの情報を元に、各細胞クラスの持つ情報を更新していく。

ピクセル場の更新処理

各ピクセルについて状態の更新を行う。最近接細胞からの距離、隣接細胞間の接着力などの細胞クラスの情報を元に各ピクセルの状態を決定する。

また、必要に応じてピクセル場の更新処理後に各種情報を画面に描画を行う。この描画により、ユーザが視覚的にシミュレーションの様子をとらえることが可能になる。

3.2　高速化

図 3.2 に示すように細胞の更新処理とピクセル場の更新処理の間にはデータの依存関係がある。この 2 つの更新処理を同時に行わせるように並列化する事はデータの依存性を保つことが難しくなるため難しい。

しかしながら、細胞の更新処理、ピクセル場の更新処理を個別に考えると、これらは共に各要素について 1 回の更新処理が行われ、かつデータの依存性も小さいため、図 3.3 のように各要素について SPE に分担して割り振ることで各 SPE が並列処理を行え、高速化に繋がると考えられる。プログラムにおいてはこれらの処理を要素ごとに SPE に分割、各々の処理について可能な限り SIMD 演算を用いる。

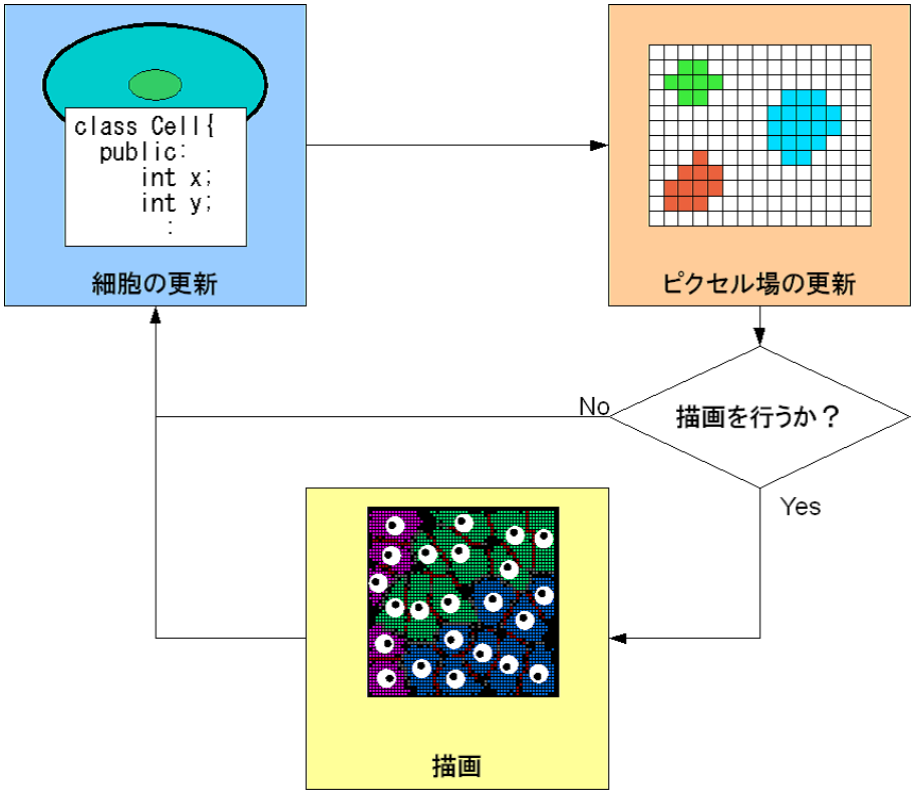

図 3.1: PXCA モデルによるシミュレーションのフロー

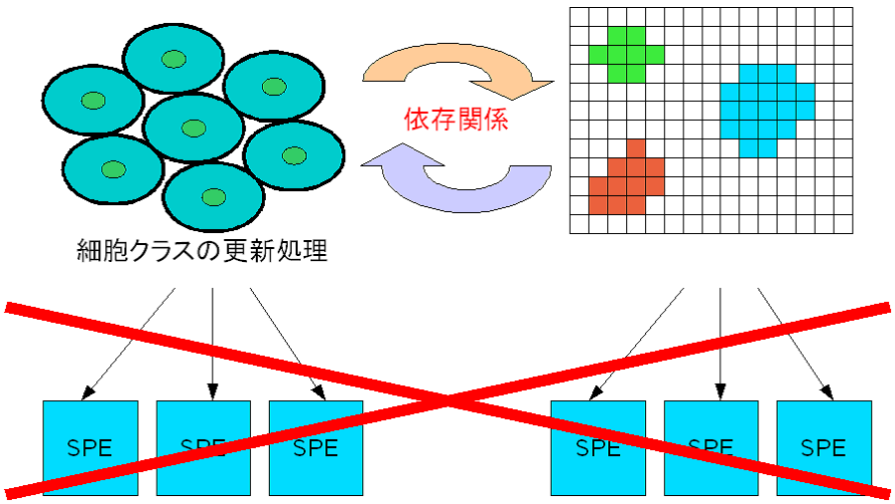

図 3.2: 細胞更新処理とピクセル場更新処理の依存性

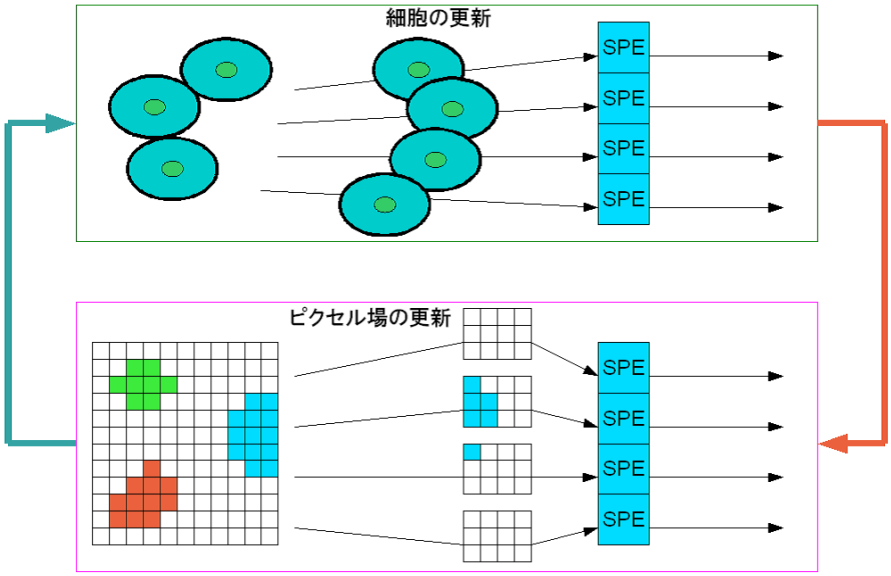

図 3.3: 処理の並列化の概念図

第4章　実装方法及び測定内容

4.1　実装方法

4.1.1　実装したPXCAモデルの特徴

今回の研究では以下のような特徴を持つPXCAモデルを実装した。

粘着性

図4.1のように同種の細胞が連結する様子を再現する性質である。細胞c_iの周辺の同種の細胞間の連結因子であるn個のピクセルの集合pに対し、時刻tにおける細胞の接着力$\vec{a}_{c_i}(t)$は以下の式で示される。

$$\vec{a}_{c_i}(t) = \sum_{j=0}^{n} \left\{ \epsilon_{ad} \left(\frac{\vec{p_j} - \vec{c_i}}{\left|\vec{p_j} - \vec{c_i}(t)\right|} \right) \left(\left|\vec{p_j} - \vec{c_i}(t)\right| - r_0 \right) \right\}$$

但し、$\vec{c_i}(t)$は時刻tにおける細胞c_iの核の位置、$\vec{p_j}$はj個目のピクセルの位置、ϵ_{ad}は全体的な接着力の強さを定義する定数$(0 \leq \epsilon_{ad} \leq 1)$であり、$r_0$は接着する細胞間の最短距離を定義する定数である。

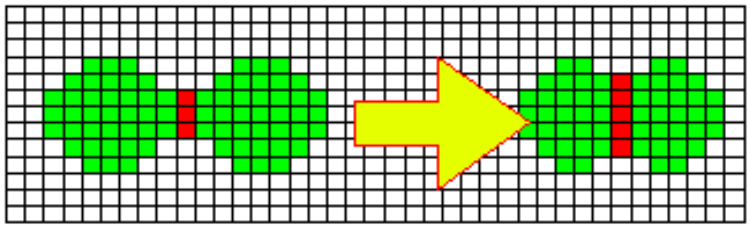

図4.1: 粘着性

成長

図4.2のように細胞の成長を再現する様子である。モデルの実装の手法は多数あるが、今回は細胞c_iに隣接するピクセルが空いている場合、ピクセルと細胞c_i間の距離が細胞の成長限界r_{c_i}以下ならばそのピクセルを細胞c_iに属する、とした。

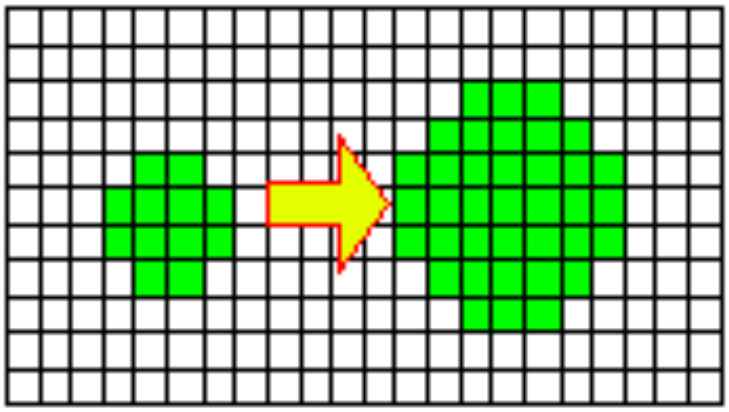

図 4.2: 成長

移動

細胞は粘着力や重心の変化、および外部からの力の作用によって図 4.3 のように移動する。時刻 t における細胞 c_i の重心への力 $\vec{g}_{c_i}(t)$ を重心への力を定義する定数 g と細胞 c_i に属するピクセルの重心 $\vec{g}_{p_{c_i}}$ を用いて表すと、

$$\vec{g}_{c_i}(t) = g\left(\vec{g}_{p_{c_i}}(t) - \vec{c_i}(t)\right)$$

となり、このとき時刻 $t+1$ における細胞 c_i の位置 $\vec{c_i}(t+1)$ は以下の式によって決定される。

$$\vec{c_i}(t+1) = \vec{c_i}(t) + (1-d)\left\{\epsilon_{nu}\left(\vec{g}_{c_i}(t) + \vec{a}_{c_i}(t) + \vec{f_{c_i}}(t)\right)\right\}$$

但し、d は運動の減衰の大きさを示すパラメータ $(0 \leq d \leq 1)$、ϵ_{nu} は細胞の運動の大きさを示す定数 $(0 \leq \epsilon_{nu} \leq 1)$、$\vec{f_{c_i}}(t)$ は時刻 t において細胞 c_i に与えられる外部からの力である。

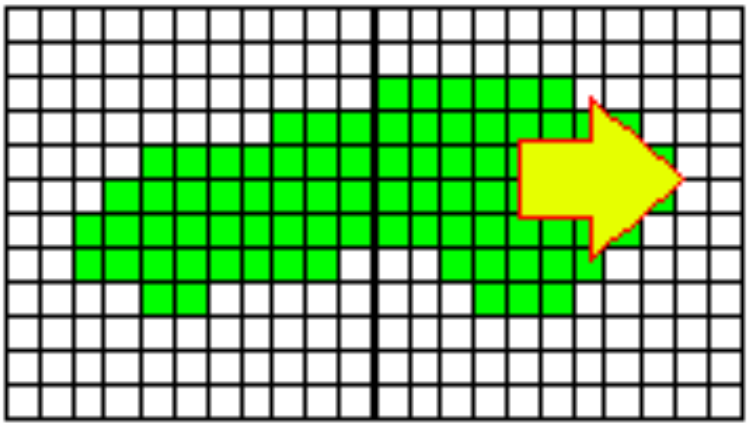

図 4.3: 移動

分裂

図 4.4 のように細胞は分裂により新たな細胞を増やす。今回の実装においては一定数以上の所属ピクセルを持つ細胞に対し、一定の確率で分裂を起こすよう設定した。分裂した細胞は分裂前の細胞の特性を引き継ぎ、分裂する方向は細胞極性により決定される。時刻 $t+1$ における細胞 c_i の細胞極性 $\theta_{c_i}(t+1)$ は n 個の細胞が c_i に隣接するとき、以下の式を用いて決定される。

$$\theta c_i(t+1) = \theta c_i(t) + \epsilon_{po} \sum_{j=1}^{n} \phi_{ij}(t)$$

但し、ϵ_{nu} は細胞極性の変化の大きさを示す定数 $(0 \leq \epsilon_{nu} \leq 1)$、$\phi_{ij}(t)$ は時刻 t における細胞 c_i に隣接する細胞 c_j が細胞 c_i に与える極性の影響であり、

$$\phi_{ij}(t) = \begin{cases} \sin\left(\varphi_{ij} - \theta_{c_i} + \frac{\pi}{2}\right) & (0 \leq \theta_{c_j} - \varphi_{ij} < \pi) \\ \sin\left(\varphi_{ij} - \theta_{c_i} - \frac{\pi}{2}\right) & (\text{otherwise}) \end{cases}$$

で示される。但し、φ_{ij} は細胞 c_i と細胞 c_j の中心核がなす角である。この式は細胞極性が細胞接触面と平行になり、かつ隣接細胞の極性に近づこうとする様子を示している。

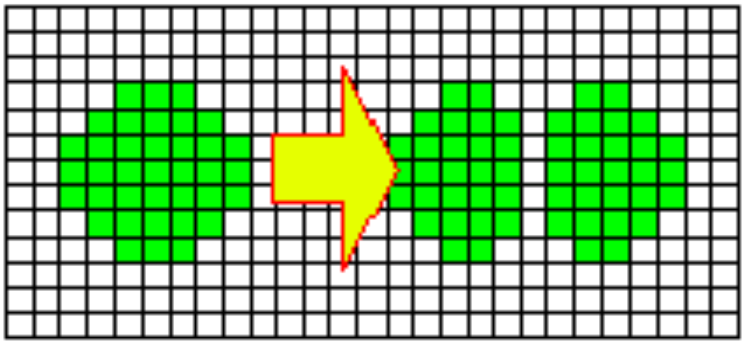

図 4.4: 分裂

死滅

最終的に図 4.5 のように細胞は活動を停止し、死滅する。今回用いる PXCA モデルにおいては各細胞の更新時に一定の確率で死滅するように設定している。

4.1.2 プログラムの製作環境

PlayStation3 に Linux ディストリビューションである Fedora Core 9 をインストールし、その環境でプログラムの製作及び実行を行った。プログラムの製作言語には C++を利用、コンパイルには PPE で動作するプログラムでは gcc 4.3、SPE で動作するプログラムでは spu-gcc 4.1.1 を用いた。

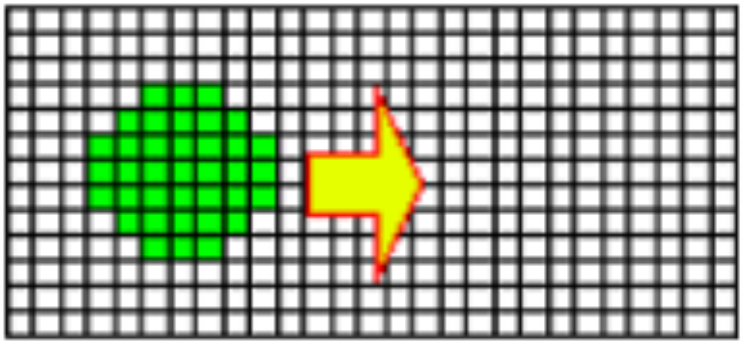

図 4.5: 死滅

4.1.3　描画の実装について

計算結果の描画処理や入力インタフェースは OpenGL(mesa+freeglut) を用いて実装したが、PlayStation3 のグラフィックボードはゲスト OS からアクセスできない仕様であるため、1 回の描画更新についておよそ 6 秒かかるなど、描画が非常に低速であった。今回の実行時間の測定では、描画に関しては行わず、シミュレーションの計算のみを行っている。

4.2　測定条件

本研究では、PXCA モデルによるシミュレーションを 10 回実行し、その平均処理時間を各種実行条件で比較、シミュレーションの有用性の検討を行った。なお、描画は行わず、1000 世代のシミュレーションの計算にかかった実行時間のみを測定した。

プログラムの実行条件は以下の通りである。

- 1PPE のみを使用、SIMD 演算による最適化を行わない
- 1PPE のみを使用、SIMD 演算による最適化を行う
- 1PPE+1SPE、SIMD 演算による最適化を行う
- 1PPE+2SPE、SIMD 演算による最適化を行う
- 1PPE+3SPE、SIMD 演算による最適化を行う
- 1PPE+4SPE、SIMD 演算による最適化を行う
- 1PPE+5SPE、SIMD 演算による最適化を行う
- 1PPE+6SPE、SIMD 演算による最適化を行う

以上の条件で、以下の 2 つのケースの平均実行時間を計測する。

- case 1 : 初期細胞数 30、最大細胞数 120、ピクセル場 120×120
- case 2 : 初期細胞数 30、最大細胞数 120、ピクセル場 240×240

なお、初期細胞の位置、細胞の種類といったパラメータは rand 関数を用いて決定する。

第5章　結果及び検討

5.1　結果

5.1.1　case 1 における実行結果

case 1 における平均実行時間の測定結果を表 5.1、および図 5.1 に示す。

PPE のみを用いた演算の場合について、SIMD 演算を用いて最適化することで実行時間はおよそ 1/2 となった。SPE を用いて並列化した場合、使用する SPE 数を増やすにつれて実行時間が減少、使用 SPE 数が 5 個以上のとき PPE のみで処理を行う場合よりも処理時間が短縮された。また、SPE を用いた処理では PPE のみを用いる場合に比べ実行時間の標準偏差が少ない。一方、並列化に使用する SPE が少ない場合 PPE のみの処理に比べ実行時間が増加しているが、これは SPE 内にプログラムイメージや各種データを格納する領域である LS(Local Store) にメインメモリ内のデータを転送する処理にかかる時間が並列化により短縮された処理時間を上回っている結果だと考えられる。

表 5.1: case 1 におけるプログラムの実行時間

使用プロセッサ	実行時間 [sec]	標準偏差 [sec]
1PPE	29.312	0.339
1PPE(Use SIMD)	11.966	0.079
1PPE+1SPE	32.806	0.021
1PPE+2SPE	19.066	0.031
1PPE+3SPE	14.708	0.031
1PPE+4SPE	12.421	0.013
1PPE+5SPE	11.467	0.020
1PPE+6SPE	10.826	0.019

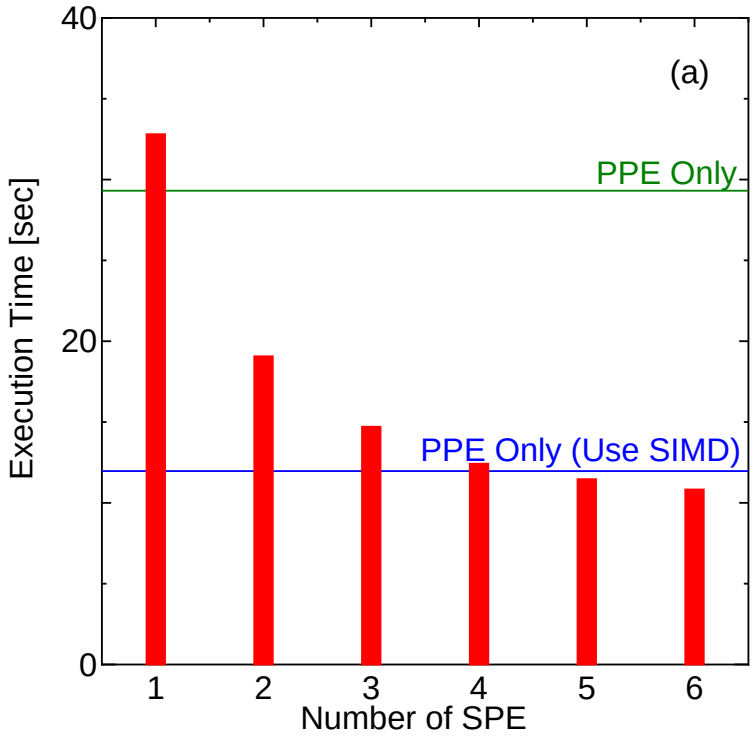

図 5.1: case 1 の実行時間

5.1.2 case 2 における実行結果

case 2 における平均実行時間の測定結果を表 5.2、および図 5.2 に示す。

case 1 と同様、SIMD 演算を用いることで PPE のみを使用した場合の処理時間はおよそ 1/2 となり、また SPE に処理を並列化した場合実行時間の標準偏差が低下、使用した SPE 数に応じて実行時間が減少した。SPE が少ない場合 PPE のみの処理に比べ実行時間が増加しているのも case 1 と同様だが、SPE を 4 個以上用いたときの処理時間は PPE のみで処理を行う場合の処理時間よりも短縮されており、並列化による処理の効率は case 1 よりも高いとみられる。

表 5.2: case 2 におけるプログラムの実行時間

使用プロセッサ	実行時間 [sec]	標準偏差 [sec]
1PPE	104.629	1.348
1PPE(Use SIMD)	53.332	0.103
1PPE+1SPE	138.718	0.054
1PPE+2SPE	76.107	0.049
1PPE+3SPE	57.968	0.045
1PPE+4SPE	48.750	0.049
1PPE+5SPE	43.696	0.052
1PPE+6SPE	40.223	0.040

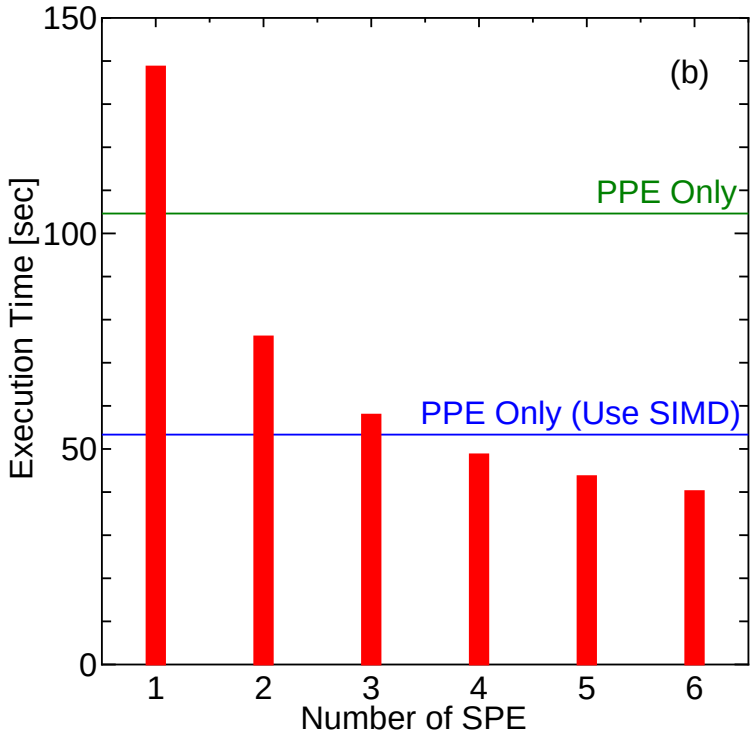

図 5.2: case 2 の実行時間

5.2 考察

両ケースにおいて、SPE を多数用いて並列化することにより PPE 単一で動作させた場合に比べプログラムの高速化が達成できた。また並列化した場合では実行時間の標準偏差が並列化しない場合に比べ小さく、実行速度が安定する傾向にある。計算する領域が case 1 のおよそ 4 倍である case 2 において、多数の SPE による処理時間と PPE のみの処理時間の差が大きくなっており、計算量が増加するほど複数の SPE を用いた並列化が有利であると考えられる。また、どちらのケースにおいても使用 SPE 数が少ない場合 LS へのデータの転送時間がボトルネックになっていると考えられるが、この点についてデータ転送手法の見直し、およびデータ転送位置の整列、小規模な転送の一括などの効率化を図ることにより、今後改善が期待される。特に全体のデータ転送量は使用 SPE 数にかかわらずある程度一定であるため、この改善は全体的なパフォーマンスの向上に繋がると考えられる。

第6章　総括

6.1　計算を行う環境としての有用性

今までの結果より、Cell/B.E. を用いることで以下のようなメリットが得られることがわかる。

- 処理の最適化による実行時間の削減
- マルチコアによる並列プログラミング環境が低コストで揃えられる

実行時間の削減に関しては最適化を行わない場合に比べ今回のマンデルブロ集合の例では処理時間がおよそ 1/18、PXCA モデルの実装の例ではおよそ 2/5 と、かなりの削減がなされていることがわかる。また低コストであると言う点も大きな利点であり、その点を活かし複数の PS3 によるクラスタを製作するといった拡張も考えられる。

SIMD 演算や特定バイト境界からのみデータが転送可能といった制限が大きいデータ転送処理といった通常のプログラムの製作にはあまり見られない手法はあるものの、得られた結果、そしてコストパフォーマンスを考えると計算環境としては Cell/B.E. を搭載した PS3 は有用であると考えられる。

6.2　シミュレーション環境としての有用性

PS3 ではゲスト OS からグラフィックボードへのアクセスが許可されておらず、すべて仮想バッファ上で間接的に処理されているため、OpenGL などのライブラリを用いた描画が非常に低速である。シミュレーションを実際に人間の目で観測する場合、画面表示が低速では観測に時間がかかり、実用的なものとは言いがたい。従って PS3 単体では観測可能な実時間シミュレーション環境を構築するのは難しいと考えられる。

これを改善する為に、図 6.1 のような PS3 を計算サーバとして用い、ある程度高速な描画が可能なコンピュータに描画に必要なデータを送信してクライアント側で画面描画を行わせるクライアント－サーバモデルを提案する。サーバで計算処理を行っている間にクライアント側で描画処理を並列で行えるため、描画にかかる時間を隠蔽でき、実時間シミュレーションを行う為に有用だと考えられる。

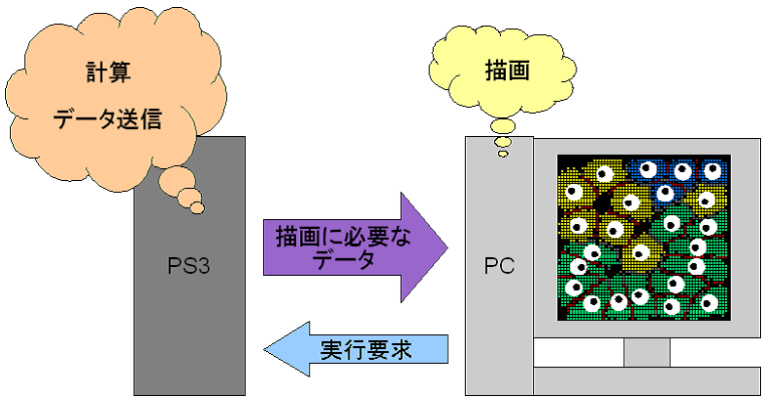

図 6.1: 実時間シミュレーションを行うためのクライアントーサーバモデル

6.3 今後の目標

今後 PXCA モデルによるシミュレーションプログラムの改良の要素として、

- PS3 を計算用サーバとするクライアントーサーバモデルの構築
- PPE-SPE 間のデータ転送を効率化することによる処理速度の向上

が挙げられる。

描画を実用的な速度で行うことのできる環境を整えることでシミュレーションの観測を行うことが可能になり、またプログラム製作作業においても処理状況を目視して動作の不明な点を発見しやすくなる。

現状の処理速度は今回の調査の case 2 において 1 秒間におよそ 25 回の更新が可能であるが、今後シミュレーションする細胞の数を増やす、またピクセル場を大きくするなどの計算量の増加に比例して 1 秒間あたりのデータ更新回数は下がると推測され、あまりに更新速度が落ちると実時間シミュレーションとして動作させることが難しくなる。目標としてはテレビの画面更新速度とほぼ同じである 1 秒間に 30 回の更新を最高の処理速度とし、ユーザの操作によって更新の速度を決定することが可能な実装が理想的だと思われる。画面の更新速度が可変であれば、例えばユーザが細胞の持つ情報に何らかの変更を及ぼした後しばらく細胞の更新される様子を遅らせて詳細に様子を見る、といった観察が可能になる。

最終的な目的として、現在 2 次元平面を対象にしているシミュレーションを 3 次元へ拡張することが挙げられる。3 次元への拡張により、立体的に細胞の様子をとらえることが可能になり、組織化の過程を解明するのにより有用になると考えられる。

謝辞

本研究を行うにあたり、多大な御指導及び御助言を賜りました小田部荘司教授、松下照男教授、木内勝助教、また有明高等専門学校の松野哲也准教授、九州工業大学大学院の五十嵐潤氏に深く感謝を致します。
また、様々な支援を頂いた小田部研究室、松下研究室の皆様に感謝の意を表します。

参考文献

[1] 松野 哲也：リアルタイムシミュレーションによるディジタル発生生物学 , FAIS マルチコアプロセッサ Workshop 2008

[2] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Shippy, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa. The design and implementation of a first-generation CELL processor. *IEEE International Solid-State Circuits Conference*, 2005.

[3] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer and D. Shippy. Introduction to the Cell multiprocessor. *IBM journal of Research and Development*, Nov.2005.

[4] Mike Kistler, Michael Perrone, and Fabrizio Petrini. Cell Multiprocessor Interconnection Network: Built for Speed. *IEEE Micro*, 26(3), May-June 2006.

[5] P. Bellens, J. M. Perez, R. M. Badia, and J. Labarta. CellSs: A programming model for the Cell BE architecture. *Proceeding of ACM/IEEE Conference on Supercomputing*, 2006.

[6] A. Buttari, P. Luszczek, J. Kurzak, J. Dongarra, G. Bosilca. SCOP3: A Rough Guide to Scientific Computing On the PlayStation 3. *Technical Report UT-CS-07-595*, May 11, 2007

[7] Sony Computer Entertainment Inc. , C/C++ Language Extensions for Cell Broadband Engine Architecture Version 2.3 , December 2006