Vol.2011-HPC-129 No.4
2011/3/15

視覚神経系数理モデルシミュレーションのMPIによる並列化

齋藤祐典[†1] 佐藤俊治[†1] 大村純一[†1]
三好健文[†1] 入江英嗣[†1] 吉永努[†1]

人間の視覚機能を解明するために，その機能の線形モデルを計算機でシミュレーションする手法がある．しかし，計算負荷の問題から簡略化したモデルあるいは一部分だけのシミュレーションのみが行われている．そこで，シミュレーションを高速化するために，プログラムを並列化し，PC クラスタを用いて実行する．本稿では，シミュレーションのコアである畳み込み演算を MPI により並列化することで，最大 43%高速化を達成した．また，実装したシミュレータを用いて錯視画像のオプティカルフローを求めたところ， 錯視現象の要因が得られたことを示す．

Parallel Numerical Simulation for the Linear Model of Visual Neurons with MPI

Yusuke Saito,[†1] Shunji Satoh,[†1] Ohmura Junichi,[†1]
Takefumi Miyoshi,[†1] Hidetsugu Irie[†1]
and Tsutomu Yoshinaga[†1]

Numerical simulation for the linear model of visual neurons is the most important approach to understand our visual system from computational viewpoints. We attempt to parallelize the time-consuming simulation on a cluster computer system. We achieved 43% reduction in simulation time by MPI implementation of spatio-temporal convolution formulated in the linear model. Moreover, by analyzing the simulation results, unknown factors on visual illusion are unveiled.

†1 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

1. はじめに

人間の脳の機能に注目し，その優れた機能を様々な情報処理機器へ応用する取り組みが盛んになってきている．脳神経系の仕組みの中でも，特に視覚系の仕組みを理解しシミュレーションできれば，例えば，事故回避やロボットビジョンなど様々な分野への応用ができる．脳の仕組みを理解するために，Blue Brain Project[1)] は IBM Blue Gene/L を用いた並列シミュレーションを行っている．このプロジェクトではコンパートメントモデルと呼ばれる詳細な神経細胞モデルをシミュレーションしている．しかし，これらの取り組みは視覚機能の仕組みを理解する上で適切なモデルではない．なぜならば，Blue Brain Project は分子レベルで神経細胞の働きをシミュレーションするため，視覚機能の再現や理解のためには粒度が小さすぎるからである．一方で，分子レベルの挙動を近似して，細胞の入出力関係だけに着目したモデルが提案されている．その中でも線形モデルは単純なモデルであるが，視覚機能の再現と理解のために十分なモデルである．しかしながら，シミュレーションすべき細胞数は膨大であるため，大量の線形視覚細胞モデルをシミュレーションすることは一般に困難である．事実，多くの視覚研究者は，少数の細胞のシミュレーションにとどまることが多い．視覚機能を正確に記述，再現，理解するためには線形モデルの大規模シミュレーションが必要不可欠である．

線形モデルによる大規模なシミュレーションを行うためには，最も time-consuming な畳み込みを高速化する必要がある．一般的な畳み込みの高速化手法として，FFT(Fast Fourier Transform) が用いられている．しかし，細胞一つ一つが少しずつ異なる性質を持つため，一律のカーネルを用いた畳み込みではないため FFT を利用できない．

本稿では，視覚神経系の数理モデルを MPI による並列シミュレーションとして実装したことで，シミュレーション時間を短縮できたことを報告する．また，本稿では一次視覚野と呼ばれる方位選択性と空間周波数によく反応する視覚処理領域のシミュレーションとして，入力画像からオプティカルフローを導出するシミュレーションを行った結果を示す．

2. 視覚系数理モデル

2.1 基本的な数理モデル

視覚系の仕組みを数値計算としてシミュレーションするために，神経細胞を数学的にモデル化する必要がある．本稿の数理モデルは，畳み込み式を基本とした線形モデルを用いる．この線形モデルは，**図 1** に示す神経細胞とニューロンの関係が基本要素である． これを数理モデルとして表すと、空間座標に並ぶ細胞 x の時間 t おける内部電位 u に関する式 (1) に示す畳み込み式で表せる．

Vol.2011-HPC-129 No.4
2011/3/15

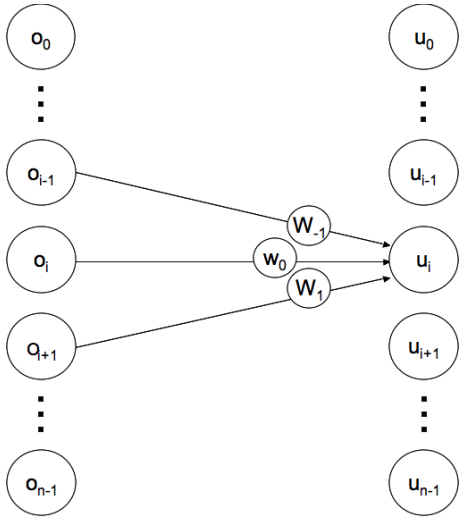

図 1 基本的なニューロンのモデル

$$u(x,t) = \sum_{\xi=-\frac{n}{2}}^{\frac{n}{2}} \sum_{\tau=0}^{m} w(\xi,\tau) o(x-\xi, t-\tau) \tag{1}$$

ここで，n と m は受容野の大きさ，w はシナプス荷重と呼ばれるカーネル，o はターゲットと呼ばれる畳み込み対象である．このとき，各細胞 u はそれぞれ少しずつ特性が異なるため，各細胞のカーネル w はそれぞれ内容が異なる．

2.2 オプティカルフロー

人間の眼球は固視微動と呼ばれる常に手ぶれのような動きがあることがわかっている．実際に手ぶれを起こしているように見えていないのは，固視微動をキャンセルする機能を有しているからではないかと考えられている．固視微動をキャンセルするためには，微動を含む画像 (網膜像) から，微動量を計算する必要がある．この微動量は広義のオプティカルフローということができる．

オプティカルフローとは物体の移動速度のベクトルである．オプティカルフローは画像の輝度分布 I に関する空間微分と時間微分の比を用いる Lucas-Kanade 法によって求めることができる．式 (2) は，x 方向に等速で移動する入力画像 $I(x-vt)$ に対し空間微分と時間微分の比を計算することで，速度 v が計算できることを示している．式 (3) の $\hat{v}$ は式 (2) にゼロ除算を回避するために微少量 ε を導入した式である．

$$v = -\frac{I_t}{I_x} \tag{2}$$

$$\hat{v} = -\frac{I_x}{{I_x}^2 + \varepsilon} I_t \tag{3}$$

式 (2) と式 (3) の時空間微分をシミュレーションするためには，これらを差分化する必要がある．例えば，時間微分は次式のように時間差分に置き換える必要がある．

$$\begin{aligned}
\frac{\partial}{\partial t} I(x,t) &= \lim_{\Delta t \to 1} \frac{I(x,t+\Delta t) - I(x,t)}{\Delta t} \\
&= I(x,t+1) - I(x,t) \\
&= (+1) I(x,t+1) + (-1) I(x,t) \\
&= w_t(-1) I(x,t+1) + w_t(0) I(x,t) \\
&= \sum_{\tau=\{-1,0\}} w_t(\tau) I(x,t-\tau)
\end{aligned}$$

このときの時間微分カーネルの値は $w_t = \{1,-1\}$ である．同様に，空間微分カーネルは $w_x = \{1,-1\}$ となる．しかし，これらのカーネルは神経生理学的に妥当性が高いとは言えないため，これまでに様々な細胞特性を持つカーネルが提案されている．本稿では，時間微分カーネルとして遅れ系を，空間微分カーネルとしてガウス関数を用いたカーネルでシミュレーションを行った．

2.2.1 時間微分カーネル

時間微分を行うために，遅れ系カーネルを用いる．人間の視覚では，フラッシュのような瞬間的な光の変化を緩やかな光として認識される．例えば，蛍光灯は 50/60Hz の連続したフラッシュであるが，実際には常に光り続けているように見えているが，遅れ系を用いることでフラッシュのような瞬間的な輝度の変化を緩やかな変化として認識できる．n 次の遅れ系は式 (4) で表される．

$$d(t) = \frac{t^{n-1}}{\Gamma(n)} e^{-\frac{t}{\tau}} \tag{4}$$

時間微分カーネルでは式 (4) を微分した，

$$D(t) = \frac{t^{n-2}}{t\Gamma(n)} \left(\tau(n-1) - t\right) e^{-\frac{t}{\tau}} \tag{5}$$

を用いる．ここで τ は時定数である．このカーネルは**図 2** のように，前述した w_t カーネルを緩やかにしたようなカーネルとして表される．

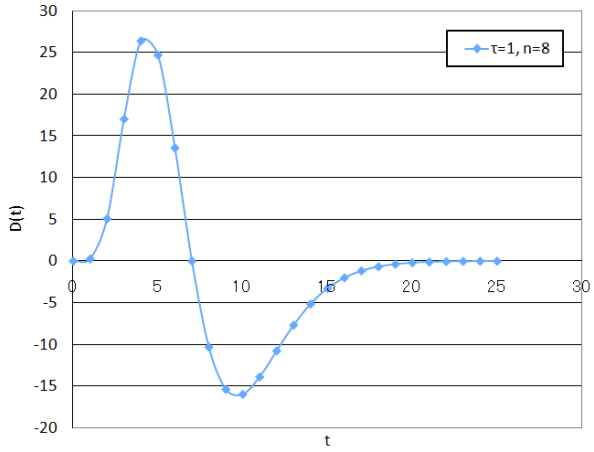

図 2 微分した遅れ系

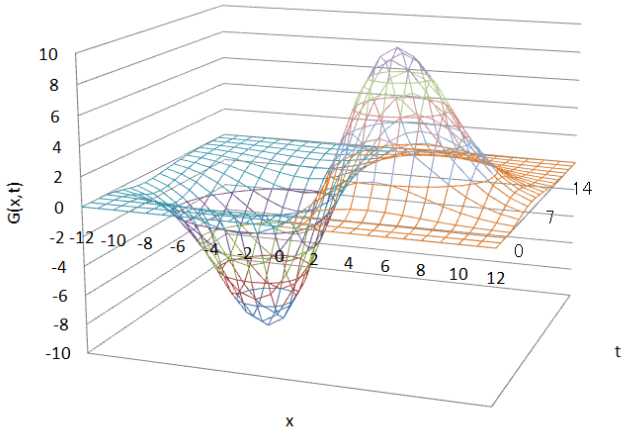

図 3 遅れ微分ガウス関数

2.2.2 空間微分カーネル

空間微分のために，空間座標に微分したガウス関数，時間座標に式 (4) を用いる．ガウス関数は画像工学的には平滑処理として用いられる．

$$\frac{d}{dx}g(x) = -\frac{x}{\sigma^3\sqrt{2\pi}}e^{-\frac{x^2}{2\sigma^2}} \tag{6}$$

空間微分カーネルとして式 (6) と式 (4) を掛け合わせ，式 (7) に示すカーネルを用いる．空間微分カーネルも**図 3** に示すように w_x カーネルを緩やかにしたグラフとなる．

$$G(x,t) = -\frac{xt^{n-1}}{\sigma^3\sqrt{2\pi}\Gamma(n)}e^{-(\frac{t}{\tau}+\frac{x^2}{2\sigma^2})} \tag{7}$$

ここで，σ は標準偏差である．

3. 提案手法と実装

本稿では，大規模なシミュレーションを行うために，シミュレーションの中心となる畳み込みの高速化を行う．畳み込みの高速化手法として，MPI を用いた通信と畳み込みのオーバーラップによる並列化を提案する．

3.1 MPI による並列化

膨大な細胞をシミュレーションする場合，畳み込みはボトルネックとなる．本稿では，畳み込みを並列化によって高速化するするために，MPI を用いた並列化を行う．複数の計算ノードを用いた大規模なシミュレーションを行うためには，ノード間通信をサポートした並列システムが必要になる．MPI は，複数の計算ノードからなる大規模な計算システムに対応した，複数プロセスによる並列計算を行うためのデファクトスタンダードである．

複数の計算ノードを用いた環境で畳み込みを行う場合，カーネルとターゲットを各ノードに配布し，計算結果をルートへ集める必要がある．MPI ではターゲットの配布に MPI_Scatter，計算結果の収集に MPI_Gather を用いることができる．しかし，ターゲットと計算結果はデータサイズが大きいため，各プロセスへのデータの送信時間とルートへの収集時間が非常に大きくなる．通信にかかるオーバーヘッドを隠蔽する手法として，Ron らはオーバーラップの有用性を示している[2]．本稿でも，畳み込みと通信をオーバーラップさせることで高速化を実現する．**図 4** に示す畳み込みのイタレーションの中で，時間 $t+1$ の畳み込みで必要になる領域の Scatter と完了した計算結果の Gather を畳み込み処理と同時に行うことで，高速化が実現できる．

ここで，MPI に標準で実装されている MPI_Scatter と MPI_Gather は非ブロッキング通信に対応していない．そのため，本手法を実現するために MPI の非ブロッキング通信である MPI_Isend と MPI_Irecv を用いて，本シミュレータ専用の非ブロッキング Scatter と Gahter を実装し，畳み込みと通信のオーバーラップによる高速化を行う．

```
for(t = result_base_dim0; t < result_bound_t; t++){
    Scatter(target[dim0+1]);
    for(x = result_base_dim1; x < result_bound_x; x++){
        kernel = map[dim1];
        for(tau = 0; tau < kernel_bound_tau; tau++){
            for(xi = 0; xi < kernel_bound_xi; xi++){
                sum += kernel[tau][xi] * target[t - tau][x - xi];
            }
        }
     result[dim0][dim1] = sum;
    }
   Gather(result[dim0]);
}
```

図 4 通信と畳み込みのオーバーラップとカーネルの切り替え

3.2 カーネルの切り替え

細胞ごとにカーネルのパラメータが異なるため，各細胞の畳み込みの度にカーネルを切り替えて畳み込みを行わなければならない．畳み込み中にカーネルを切り替えるために，細胞の数分の大きさを持つ Map 配列を用意する．あらかじめ Map 配列に使用するカーネルの先頭アドレスを格納し，図 4 に示す空間畳み込みの度に Map 配列からアドレスを読み出すことでカーネルを切り替える．

3.3 netCDF[3] によるファイルの入出力

シミュレーション結果をファイルとして出力するために，netCDF(network Common Data Form) を用いた．netCDF は数値データを記録するだけでなく，そのデータに対しメタデータを付加することができる．さらに，高速・並列 IO，非環境依存であるため，大規模なシミュレーションに対応できる．臼井らの取り組み[4] から，今後視覚系シミュレーションのフォーマットとして推奨されることを考慮し，本シミュレータでも netCDF を用いた．

3.4 シミュレータの性能評価

表 1 に示すサイズのターゲットに対し，オプティカルフローを導出するシミュレーション時間を計測した．実験環境は Gigabit Ethernet で接続された 4 台のスイッチと 8 台の計算ノード，1 台のファイルサーバからなる PC クラスタである．各ノードとソフトウェアの環境を**表 2** と**表 3** に示す．ターゲットの最小サイズは，一般的に少ない細胞数で行われるシミュレーションを基準として 1000×400 をターゲットサイズとして設定した．ターゲットの最大サイズはノード 1 台における物理メモリにプログラムのメモリ使用量が収まるおおよそのサイズとして 100000×400 を設定している．

計測結果を**図 5** に，**図 6** にターゲットサイズが 100000×400 の実行時間の詳細を示す．図 5 からわかるように，50000×400 と 100000×400 では並列化の効果が得られたことがわかる．1000×400 はサイズが小さすぎるため，ほとんど並列化による速度向上が見られなかった．100000×400 では，1 プロセスでの実行時間は約 89.5 秒である．8 ノード 64 プロセスで実行し場合，43%高速化し，38.8 秒であった．空間微分の畳み込みはプロセス数が増えるに従って並列化の効果が得られ，32 プロセスで約 1/10 まで計算時間が短くなった．一方，時間微分の畳み込みでは，計算量が少ないため，並列化による速度向上が得られなかった．また，空間微分カーネルのブロードキャストの通信時間が大きくなってしまった．IO に約 22 秒も時間がかかってしまっているのは，ファイルの入出力先がファイルサーバーだからである．

表 1 ターゲットサイズとカーネルサイズ

ターゲット	1000×400, 10000×400, 50000×400, 100000×400
空間微分カーネル	25x20
時間微分カーネル	1x20

表 2 ハードウェア環境

ノード数	8
CPU コア数	4
プロセッサ	Intel Xeon Dual-Core 5160
周波数 (GHz)	3.00
IPC	2
キャッシュ(MB)	2
メモリ (GB)	4
ネットワークバンド幅 (Mbps)	1000

表 3 ソフトウェア環境

OS	CentOS 4.7
C compiler	ICC 11.1
MPI ライブラリ	OpenMPI 1.4.1[5]
I/O ライブラリ	NetCDF 4.0.1

これらの結果から，大規模シミュレーションを行う上でボトルネックである畳み込みに要する時間は十分小さくできたことがわかる．しかし，カーネルのブロードキャスト時間とメモリの制約を改善する必要がある．

4. オプティカルフロー導出シミュレーション

本シミュレータを用いて，正弦波と Rotating Snake のオプティカルフローをシミュレーションする．正弦波は一次視覚野においてよく反応する画像パターンである[6]．まず，正弦

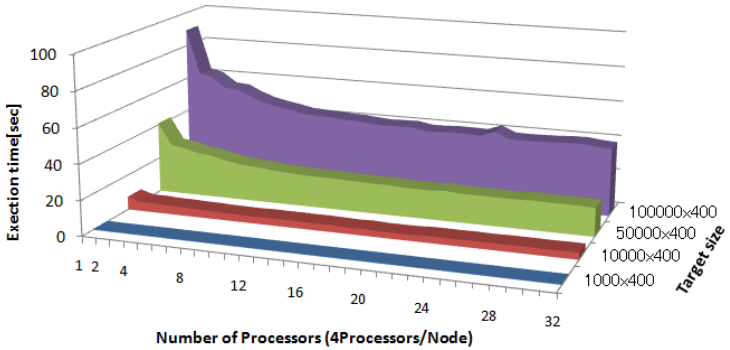

図 5 シミュレーション時間

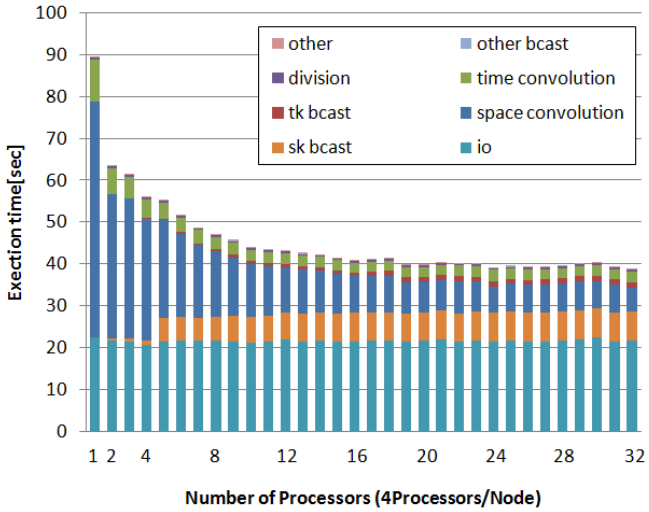

図 6 100000 × 400 シミュレーション時間

表 4 正弦波のシミュレーション環境

空間周波数 (Hz)	0.004 0.008 0.016
速度 v	1.0
σ	4
τ	1
n	8
ε	0.03^2

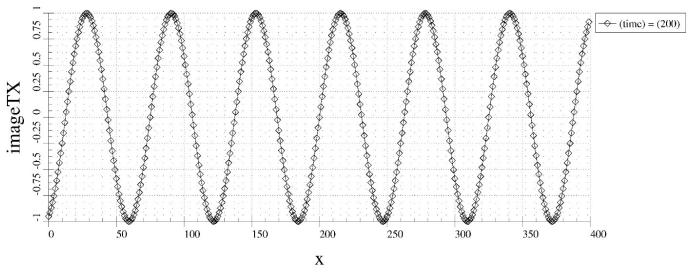

図 7 $f = 0.016$ の正弦波

波を用いてオプティカルフローを求めた後，これと同様のパラメータセットのカーネルを用いて Rotating Snake のオプティカルフローを求めた．

4.1 正弦波によるシミュレーション

速度 v で動く正弦波格子 $I(x,t) = 2\pi f sin(x - vt)$ をターゲットとしたオプティカルフローを求めるシミュレーションを行う．**表 4** にシミュレーションするターゲットとカーネルのパラメータを示す．

本シミュレーションの実行時間は約 38 秒である．**図 7**，**図 8**，**図 9** は空間周波数 f が 0.016Hz，0.008Hz，0.004Hz の場合の正弦波，**図 10**，**図 11**，**図 12** は各空間周波数が速度 $v = 1$ におけるオプティカルフローのグラフである．$f = 0.016$Hz の場合，ターゲットの速度 v とオプティカルフローのピークはほぼ同じであることがわかる．一方で，$f = 0.008$Hz，$f = 0.004$Hz の場合，オプティカルフローのピークは入力速度よりも低い結果を示している．これらの結果から，空間周波数が高い場合は速度 v に近づき，低いほど v よりも遅いオプティカルフローが得られることがわかる．これは，人間の視覚では正弦波の空間周波数に対して知覚している速度が異なっている可能性を示している．

4.2 Rotating Snake によるシミュレーション

Rotating Snake は**図 13** に示す静止画であるにもかかわらず動いて見える錯視画像の一つである[7]．このような錯視画像に対し，どのようなオプティカルフローが得られるのかシ

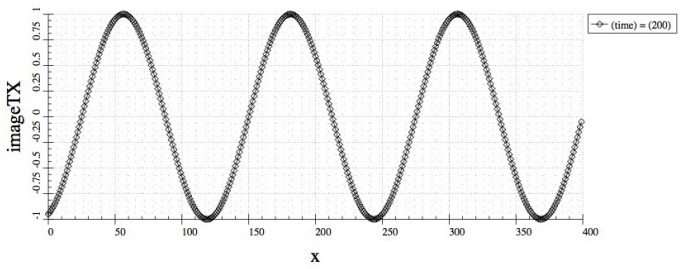

図 8 $f = 0.008$ の正弦波

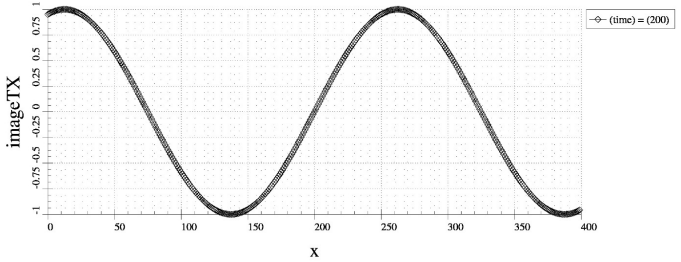

図 9 $f = 0.004$ の正弦波

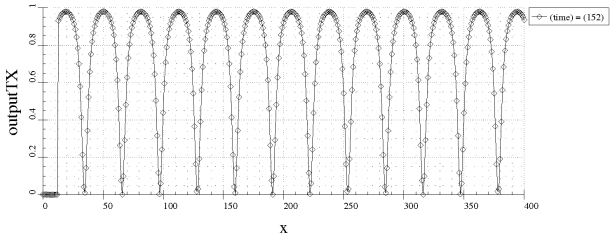

図 10 $f = 0.016$ の正弦波のオプティカルフロー

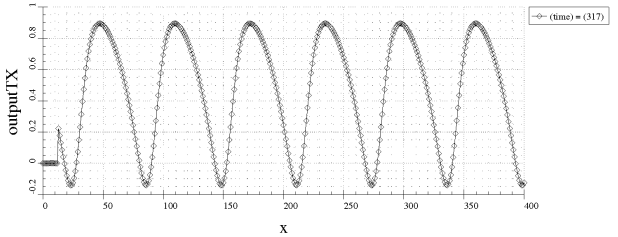

図 11 $f = 0.08$ の正弦波のオプティカルフロー

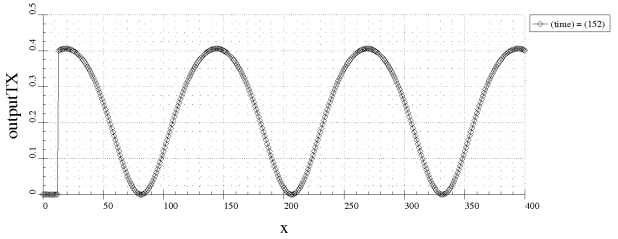

図 12 $f = 0.004$ の正弦波のオプティカルフロー

ミュレーションする．簡単のため，配列に Rotating Snake の輝度のパターンを並べ，一次元 Rotating Snake としてシミュレーションを行う．ターゲットとカーネルのパラメータを**表 5**，各輝度の幅が 5pixel の一次元 Rotating Snake を**図 14**，輝度の幅が 10pixel の一次元 Rotating Snake を**図 15** に示す．シミュレーション結果として，輝度幅が 5pixel の速度 $v = -1$ で動いている場合のオプティカルフローを**図 16**，$v = 1$ の場合を**図 17** に示す．同様に，輝度幅が 10pixel における $v = -1$ のオプティカルフローを**図 18**，$v = 1$ のオプティカルフローを**図 19** に示す．

本シミュレーションの実行時間は約 38 秒である．輝度の幅が 5pixel の場合，$v = -1$ の Rotating Snake のオプティカルフローは速度のピークが $\hat{v} = -1.64$ であった．一方で，$v = 1$ を与えた場合は $\hat{v} = 1.81$ であった．どちらの結果でも，入力速度 v よりもオプティカルフロー $\hat{v}$ の方が絶対値は大きい結果となった．また，$v = -1$ の結果と $v = 1$ の結果は，移動方向が逆であるだけだが，ピーク値が異なる結果となった．輝度の幅が 10 ピクセルの場合では，$v = -1$ で $\hat{v} = -1.55$，$v = 1$ で $\hat{v} = 1.49$ であった．この結果では $v = -1$ の方がオプティカルフローのピーク値が大きかった．輝度の幅が 5pixel の結果では，$v = -1$ の絶対値のピーク値を基準に固視微動をキャンセルする場合，$v = 1$ の最大値の方が大きいた

Vol.2011-HPC-129 No.4
2011/3/15

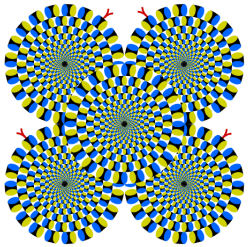

図 13 Rotating Snake

表 5 RotatingSnake のシミュレーションパラメータ

輝度パターン	0.2, 0.0, 0.8, 1.0
輝度幅	5 10
速度	-1 1
σ	4
τ	1
n	8
ε	0.03^2

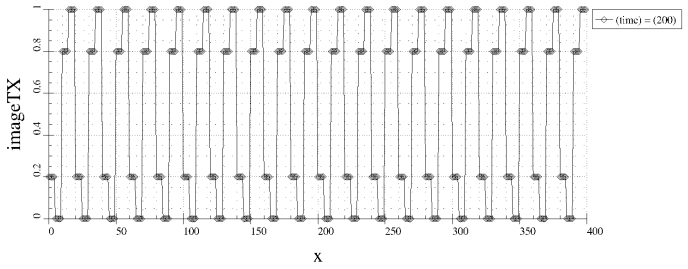

図 14 輝度幅が 5pixel ずつ並んだ一次元 Rotating Snake

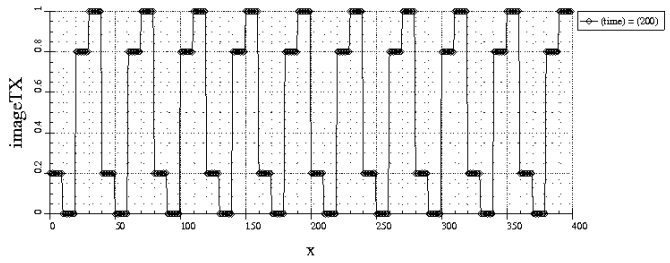

図 15 輝度幅が 10pixel ずつ並んだ一次元 Rotating Snake

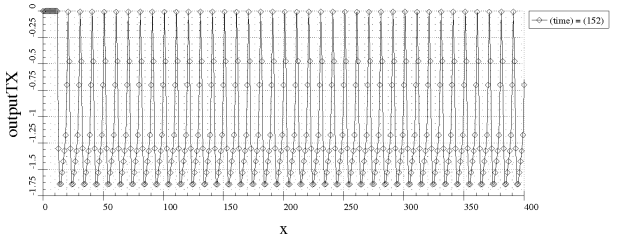

図 16 $v = -1$ の Rotating Snake のオプティカルフロー

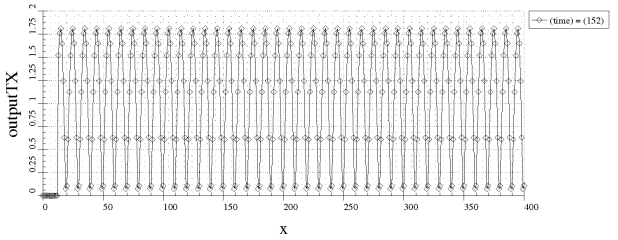

図 17 $v = 1$ の Rotating Snake のオプティカルフロー

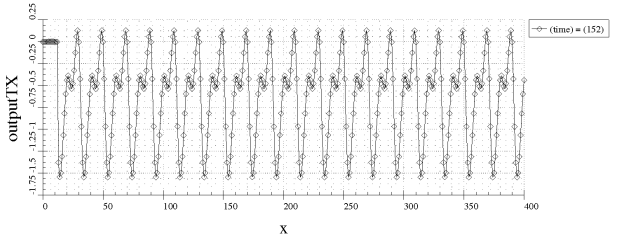

図 18 $v = -1$ の Rotating Snake のオプティカルフロー

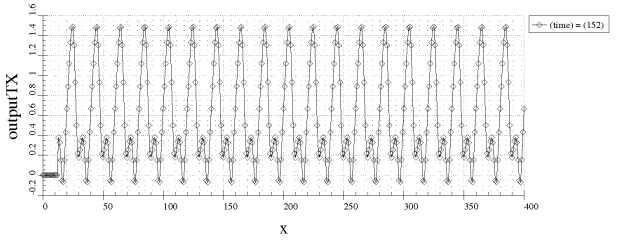

図 19 $v = 1$ の Rotating Snake のオプティカルフロー

め，右に動いて見えると考えられる．また輝度の幅が 10pixel では，$v = -1$ の場合のピーク値の方が大きい．そのためこのパターンでは左に動くと考えられる．これらの結果は，入力速度 v よりもオプティカルフローの方が大きいということを示す．また，動く向きに対してピーク値は異なるという値を示す．このような特徴は錯視現象の要因の一つである可能性がある．また，輝度の幅が小さい場合右に動くが，輝度の幅が大きい場合左に動く．これは，空間微分カーネルのパラメータによって変化する可能性がある．

5. 関 連 研 究

Blue Brain Project は IBM Blue Gene/L を使い，コンパーチメントモデルという詳細なシミュレーションを行うために立ち上げられたプロジェクトである[1)]．細胞膜をコンデンサとしてモデル化しており，物質のイオンの流れや電流のスパイクなどの複雑な振る舞いをシミュレーションする．

MOOSE もまた詳細な細胞モデルのシミュレータである[8)]．MPI による並列処理が可能である．線形モデルのシミュレーションもできるが，Blue Brain Project と同様に粒度が細かすぎるため，視覚機能のシミュレーションに向いていない．

6. お わ り に

本稿では，大規模シミュレーション実現のために，MPI を用いて畳み込みの並列・高速化を行った．その結果，ターゲットサイズが 100000×400 の場合に，最大で 43%の性能向上が得られ，大規模シミュレーションのボトルネックとなる畳み込み時間を短縮することができた．ただし，カーネルのブロードキャスト時間が大きくなる結果となってしまった．

オプティカルフローを導出するシミュレーションでは，正弦波と Rotating Snake をターゲットとしてシミュレーションを行った．その結果，本シミュレータによって，錯視現象の要因と考えられる結果を示すことができた．正弦波のシミュレーションから空間周波数によって物理速度と人間の認識している速度が異なる可能性があることがわかった．Rotating Snake を用いたシミュレーションでは，入力速度よりもオプティカルフローのピーク値の方が大きく，さらに移動方向に対して非対称という結果が得られた．これらの結果は，錯視現象の原因の一つである可能性がある．

本稿では，メモリ制約上 100000×400 というサイズのシミュレーションしか行えなかった．複雑型・超複雑型細胞や視覚情報処理領域間のカースケードといった階層構造を持つ大規模なシミュレーションを行う場合，このような問題点を改善する必要がある．今後は，上記の問題点と，畳み込みの結果を次の畳み込みへ柔軟に渡せるように改良し，大規模なシミュレーションを行っていく予定である．

参 考 文 献

1) Markram, H.: The blue brain projcet, *Neuroscience*, pp.153–160 (2006).
2) Brightwell, R. and Underwood, K.: An analysis of the impact of MPI overlap and independent progress, *International Conference on Supercomputing (ICS)* (2006).
3) netCDF(network Common Data Form). `http://www.unidata.ucar.edu/software/netcdf/`.
4) S.Usui, K.Inagaki, T.Kannon, S.Satoh, NL.Kamiji, Y.Hirata, A.Ishihara and H.Shouno: A Next Generation Modeling Environment Platform for Collaborative Brain System Modeling, *LNCS 5863*, pp.84–90 (2009).
5) OpenMPI: Open Source High Performance Computing. `http://www.unidata.ucar.edu/software/netcdf/`.
6) Valois, D., L, R., Valois, D. and K.K: Spatial Vision, *Oxford University Press* (1988).
7) 北岡明佳：北岡明佳の錯視のページ. `http://www.ritsumei.ac.jp/~akitaoka/`.
8) Cubert, R. and Fishwick, P.: MOOSE: An Object-Oriented Multimodeling and Simulation Application Framework, *Simulation* (1997).