Vol.2011-HPC-130 No.8
2011/7/27

視覚神経系モデルシミュレーションの複数GPUによる高速化

大村純一[†1] 佐藤俊治[†1] 江頭明[†1]
三好健文[†1] 入江英嗣[†1] 吉永努[†1]

人間の視覚系の「機能」を細胞の入出力関係だけに着目して線形モデルで表し，計算機でシミュレーションを行う手法が広く用いられている．しかし膨大な細胞数のシミュレーションには時間がかかるため，モデルを簡略化し，再現すべき視覚機能を限定したシミュレーションが行われることが多い．そこで，本研究では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，GPU を搭載した PC クラスタを用いて実行する．本稿では，シミュレーションで必要となる演算を，NVIDIA C1060 と C2070 の異なるアーキテクチャの GPU で実行したときの性能の違いについて示す．また，C1060 を搭載した PC クラスタ上での並列実行により，16 台の GPU を用いたシミュレーションにおいて高い並列化効率が実現できたことを示す．

Multi-GPU Acceleration of Numerical Simulation for the Linear Model of Visual Neurons

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

A popular approach to understand our human visual “functions” is performing the computational simlutaions of its linear models created with a focus on input-output relations of cells. However, due to a lot of simulation time for a huge amount of cells, it often happen that only simplified models of selected visual “functions” have been dealt with. So, in order to speed up the simulation, in this study we attempt to parallelize the program and perform it on a GPU-accelerated PC cluster. We show a performance comparison between NVIDIA C1060 and C2070, which are difference in their architecture, in a case of performing the core operations required in the simulation. Moreover, we show that high parallelization efficiencies are observed when parallelizing the program on a PC cluster, which has 16 nodes equipped with C1060 cards.

1. はじめに

脳の視覚系細胞による柔軟な情報処理能力は，既存の工学的パターン処理アルゴリズムの能力を超える場合もあるため，その仕組みを理解・記述・シミュレーションすることができれば，さまざまな分野へ応用することができる．一方，視覚は不完全な性質も併せ持つ．たとえば錯視はその代表例であるが，錯視特性を含めた視覚の記述・シミュレーションは，特定の条件下での事故の回避や原因の究明などに役立つ．

神経系の詳細なシミュレーションを目的とした研究の一つとして，Blue Brain Project[1] が挙げられる．このプロジェクトでは IBM Blue Gene/L を用いて，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並列計算によりシミュレーションしている．コンパートメントモデルのシミュレーションによって，細胞個々の複雑な特性を記述・再現することができる．しかし，シミュレーション対象としているモデルの粒度が小さすぎるため，視覚系の「機能」の再現や理解のためには適切なモデルとは言えない．

一方で，モデル粒度を大きくし，細胞の入出力関係だけに着目したモデルが様々提案されている．その中でも線形モデルは最も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と言える．事実，テクスチャ解析や色情報処理などの視覚機能モデルの多くが線形モデルによって表現されている[2)–4)]．すなわち，視覚系の機能をシミュレーションするためには，単純な線形モデルで十分である場合が多い．

個々の細胞は単純な線形モデルで十分であるが，視覚機能をシミュレーションするためには膨大な細胞数を扱わなければならないため，実行時間の問題などが生じる．事実，多くの視覚研究者は，問題を簡略化し，再現すべき視覚機能を限定したシミュレーションを実行することが多い．様々な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．

線形モデルシミュレーションで最も時間を要する箇所は，畳み込み計算 (相関計算) である．そのため，畳み込み計算を高速化することによる上記問題の解決が期待される．畳み込み計算の高速化手法として，FFT(Fast Fourier Transform) が挙げられる．FFT はターゲット (入力信号) とカーネル (フーリエ基底) の畳み込みを高速に計算する際に用いられる

†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

が，利用できるのはカーネルが時空間で均一である場合に限られる．しかし，視覚細胞は空間的に異なる性質を持つ (空間的に異なるカーネルを持つ) ため FFT を適用することができない．

この問題を解決するために先行研究[5] では，MPI を用いて畳み込みを並列化して，PC クラスタ上で実行することによりシミュレーション時間を短縮できることを示した．しかし，処理対象は，問題を簡略化した 1 次元輝度配列の時間変化データに限定していた．本研究では 2 次元輝度配列の時間変化データ，つまり一般的な動画データに適用できるように拡張を行った．次元の追加はターゲットだけでなくカーネルについても行われるため，必要な計算量は，ターゲットとカーネルそれぞれに追加された次元のサイズの積だけ倍増する．例えば，ターゲットに追加された次元のサイズが 240 で，カーネルに追加された次元のサイズが 15 であれば，1 次元輝度配列の時間変化データの場合の 3600 倍もの計算が必要となる．

このように急激に増加する計算量を CPU だけで補うのは難しいため，本研究では，2 次元の動画データを対象とするシミュレーションを，GPU を用いることによって高速化する手法について研究を行った．

本稿では，実装したシステムを用いて，2 次元の動画データを対象とする畳み込みを高速に計算できることを報告するとともに，異なるアーキテクチャの GPU(NVIDIA C1060, C2070) を用いた場合の性能の違いについても考察する．また，GPU(NVIDIA C1060) を搭載した PC を 16 台接続したクラスタ環境において，MPI を用いた並列化を用いることにより，高い並列化効率が得られたことを報告する．

2. 視覚神経系モデルシミュレーション

2.1 基本的な数理モデル

視覚神経系の入出力特性を数値シミュレーションするためには，視覚神経系の様々な機能を数学的にモデル化する必要がある．**図 1** に示すように，視覚神経系の各機能は膨大な数の神経細胞が相互に影響を与えることによって実現されている．一般に個々の神経細胞は，前段の神経細胞の出力信号を入力として受け，後段の細胞へ出力信号を伝える．

この仕組みを簡単に図示すると，図 1 の左下のようなモデルで表現できる．n は各神経細胞が受け取る信号の範囲，w はシナプス荷重と呼ばれる重み係数，o は前段の細胞の出力，すなわち後段の細胞への入力である．出力 u は，o(ターゲット) と w(カーネル) の積和，すなわち畳み込み (もしくは相関計算) で表現することができる．

このような関係を，画像の座標空間である 2 次元空間に拡張し，入力信号の時間変化につ

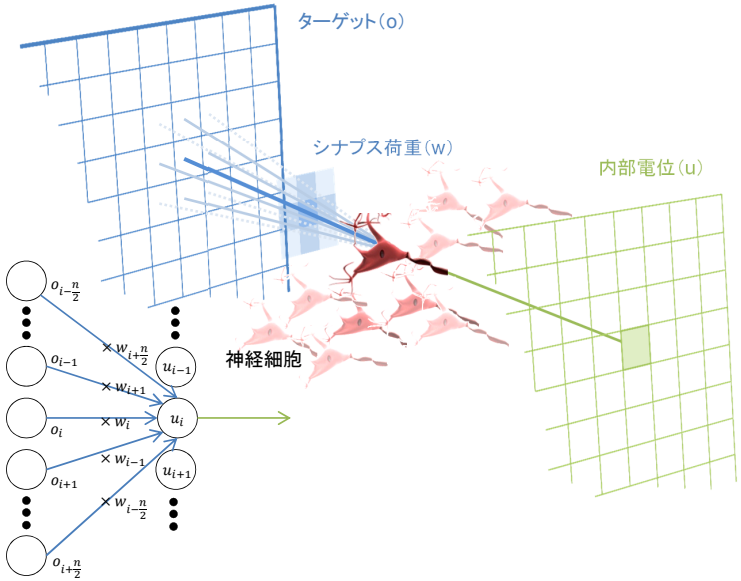

図 1 基本的な視覚神経系機能のモデル化
Fig. 1 Modeling of a basic function of the human visual system.

いても考慮すると，空間座標 (x,y) に位置する細胞の時間 t における出力 u は，以下の式で表すことができる．

$$u(x,y,t) = \sum_{\xi=-\frac{n}{2}}^{\frac{n}{2}} \sum_{\eta=-\frac{m}{2}}^{\frac{m}{2}} \sum_{\tau=0}^{l} w_{x,y}(\xi,\eta,\tau) o(x-\xi, y-\eta, t-\tau) \quad (1)$$

n,m,l は重みの時空間的範囲 (カーネルサイズ) を表す．

工学的な画像処理では通常，小さなカーネルサイズを用いてエッジ検出などを行う．一方，視覚神経系のシミュレーションでは，3 次元カーネルの各次元のサイズが十数〜数十である場合が多い．また，各神経細胞は，同じ機能に関わるものであっても，一つ一つ特性が異なるため，空間座標 (x,y) ごとに異なるカーネル $w_{x,y}$ を用いる必要がある．

視覚神経系のシミュレーションは，このように計算量の大きな畳み込み計算を，多段に組み合わせて表現される．これに加えて畳み込み計算の出力の調整や集約のために時空間座標ごとの四則演算を必要とするモデルもある．例えば，1 つの入力に対して異なるカーネルで

畳み込みを行い，それぞれのカーネルで得られた時空間座標ごとの出力を，四則演算を用いてまとめ，1 つまたは複数の出力に変換するといったモデルが存在する．このような四則演算に必要な計算量はわずかであるが，これらの計算を実施するタイミングによってはデータ通信量の削減などで処理時間に影響を与える可能性も否定できない．そこで本研究では，このような四則演算も含め，高速化の検討を行う．

2.2 本研究での高速化のポイント

本研究では複数 GPU を用いてシミュレーションを高速化するために，以下の課題について検討を行う．

複数カーネル対応 空間座標ごとに異なるカーネルを用いて畳み込みを行うためには，大量のカーネルデータを扱う必要がある．軸ごとの 1 次元のカーネルに分離し，その外積として表現できるカーネルを，Separable な (分離可能な) カーネルと呼ぶ．カーネルが Non-separable(分離不可能) の場合は，**表 1** に示すような大量のデータ領域が必要となる．しかし，カーネルが Separable な場合であれば，これを**表 2** に示すサイズまで削減することができる．視覚系機能のシミュレーションのためには Non-separable なカーネルの処理も必要となるが，Separable なカーネルでシミュレーションが可能なモデルも存在する．本研究では，初期実験としてこのような Separable なカーネルでシミュレーションが可能な場合に限り，GPU を使って高速化する手法について検討を行う[⋆1]．

複数畳み込みの一括実行 同じ入力について複数の異なる機能をシミュレーションする複数の畳み込みが必要な場合の高速化の手法について検討を行う．

ノード内通信の削減 CPU ⇔ GPU 間の通信や，GPU 内の演算コア[⋆2]とオフチップメモリ間[⋆3]の通信など，ノード内で発生する通信により転送されるデータ量を削減することにより処理時間を短縮する手法について検討を行う．

GPU を搭載した複数ノードによる並列実行 最後に，GPU を搭載した複数のノードを Gigabit Ethernet 2 リンクで接続した PC クラスタ上で，MPI を用いて畳み込みを並列実行することにより，シミュレーションを高速化する手法について検討を行う．一括実行可能な 3 つの畳み込みを主な要素とするシミュレーションにおいて，適当な大きさのカーネルを用いるときに，処理速度をどの程度リアルタイムに近づけることができるか検証する．

⋆1 複数 GPU による並列実行により 1GPU で必要とされるカーネルの数が少なくなれば，Non-separable なカーネルについても扱うことができると考えられるが，これは今後の検討課題である．
⋆2 C1060 のアーキテクチャでは Streaming Processor，C2070 のアーキテクチャでは CUDA コアと呼ばれる．
⋆3 グローバルメモリとも呼ばれる，GPU カード上のメモリデバイス．

表 1 空間サイズ・カーネルサイズに対する合計カーネルデータサイズ (non-separable kernel)
Table 1 Total size of all kernel data for varying space size and kernel size (non-separable kernel).

	15 × 15 × 15	20 × 20 × 20	25 × 25 × 25
320 × 240	0.97GB	2.29GB	4.47GB
512 × 384	2.47GB	5.86GB	11.44GB
640 × 480	3.86GB	9.16GB	17.88GB

表 2 空間サイズ・カーネルサイズに対する合計カーネルデータサイズ (separable kernel)
Table 2 Total size of all kernel data for varying space size and kernel size (separable kernel).

	15 × 15 × 15	20 × 20 × 20	25 × 25 × 25
320 × 240	13.2MB	17.6MB	22.0MB
512 × 384	33.8MB	45.0MB	56.3MB
640 × 480	52.7MB	70.3MB	87.9MB

3. 提案手法・実装

1 フレーム[⋆4]ごとに CPU ⇔ GPU 間のデータ転送と GPU 上での計算を行う．GPU 上での計算時間に比べ，CPU ⇔ GPU 間のデータ転送に要する時間は非常に小さいため，GPU へのデータ転送，GPU 上での計算，GPU からのデータ転送の処理は順次行う．GPU 上のデバイスメモリには，カーネルの時間軸方向の大きさ分のフレーム数だけ入力データを保持する．カーネル・ターゲットともにデータ依存性がなくデータの再利用性が低いため，従来から GPU で行われているグラフィック処理に近い特徴を持つ演算となる．

NVIDIA 製 GPU ではメモリレイテンシの削減のために，共有メモリと呼ばれる高速なオンチップメモリを用いる実装が広く用いられている．しかし，容量の小ささから本研究では共有メモリは用いていない．代わりに，キャッシュによる高速化が期待でき，扱えるデータの容量が大きなテクスチャメモリを用いて，ターゲット・カーネルデータの読み込みを高速化する手法を用いた．

3.1 複数カーネル対応

本研究では，**図 2** のような各空間座標ごとに異なる Separable なカーネルを用いる場合

⋆4 ある時間 t における 2 次元輝度配列データを，動画データフォーマットの例に倣い，本稿ではフレームと表記する．

Vol.2011-HPC-130 No.8
2011/7/27

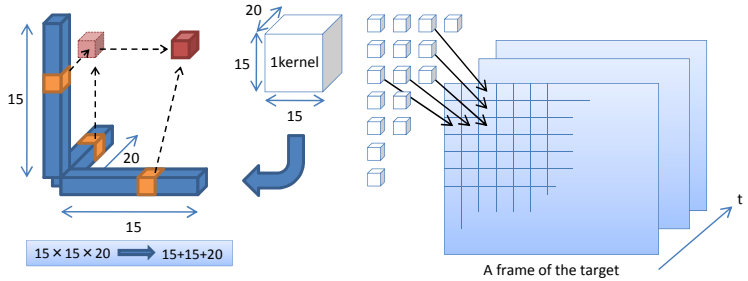

図 2 各畳み込みにて複数カーネル (Separable) を用いる
Fig. 2 Multiple kernels (separable) for each convolution.

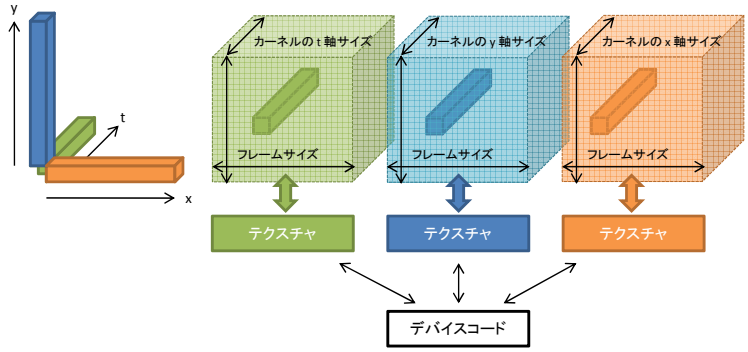

図 3 複数カーネル (Separable) の格納・アクセス方法
Fig. 3 A method to store and access to multiple kernels (separable).

について取り扱う．3 次元の Separable なカーネルは，図 2 のようにそれぞれの軸ごとの 1 次元カーネルの外積として表現される．この 3 つの 1 次元カーネルの値を GPU 上のそれぞれ異なる 3 つの 3 次元配列に格納し，テクスチャを利用してデバイスコードより取得する実装を用いた．デバイスコードからはフレーム上の空間座標 (x,y) と，1 次元カーネル上の座標 (x または y または t) を指定してそれぞれのテクスチャより値を取得する．この様子を図 **3** に示す．

3.2 複数畳み込みの一括実行

図 4 に示すように，同じ入力に対して異なる機能に相当する畳み込みを行うモデルでは，これらの畳み込みを 1 つのデバイスコードでまとめて一括処理することにより，ターゲットデータへのアクセス回数を減らし高速化できると期待できる．

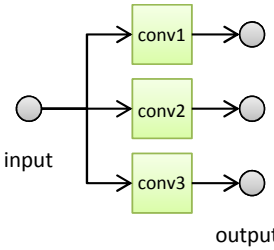

図 4 複数畳み込みの一括実行
Fig. 4 A batch processing of multiple convolutions.

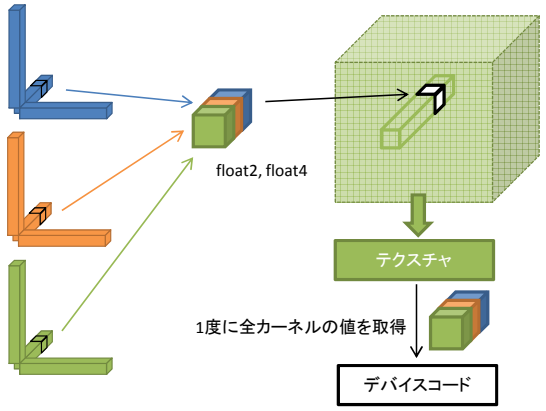

図 5 一括実行する畳み込みのカーネルの格納・アクセス方法
Fig. 5 A method to store and access to kernels for convolutions processed in a batch.

このような一括実行で，それぞれの畳み込みのカーネルデータを取得するために，本研究では図 **5** に示すような float2,float4 型[⋆4]を利用した単一の配列を用いる手法を実装した．

また，3 つの float 値をメンバとする構造体 float3 も定義されているが，float3 型のテクスチャを使用するプログラムは正常に動作しないため，3 つの畳み込みの一括実行時にも float4 型を利用する実装とした．

3.3 ノード内通信の削減

前節のように一括実行により得られた複数の畳み込みの結果を，四則演算などにより 1 つまたは複数の結果にまとめるようなシミュレーションが存在する．このようなシミュレーションでは，図 **6** に示すように，GPU 上で結果をまとめる演算までを行うことにより，GPU 内の演算コア⇒ GPU 内オフチップメモリ⇒ CPU 間で転送されるデータ量を削減する手法を用いる．

視覚神経系のシミュレーションで扱う畳み込み計算は，各時空間座標ごとに異なる大量

⋆4 本研究の実験で利用している NVIDIA 製 GPU を利用するプログラム開発環境の NVIDIA CUDA Toolkit[6] で定義された組み込み型．それぞれ 2 つまたは 4 つの float 値をメンバとする構造体．

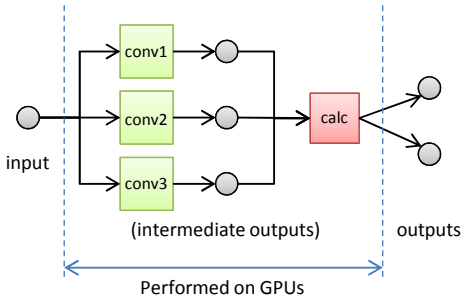

図 6　ノード内通信の削減
Fig. 6　Reducing the communication volume.

のデータへのアクセスが必要なため，GPU での処理では，計算にかかる時間よりもデータアクセスにかかる時間の方がはるかに大きい傾向にあると予想できる．そのため，GPU 上での計算を増やし，GPU ⇔ CPU 間 (および GPU 内のオフチップメモリ⇔演算コア間) のデータ転送量を削減することで，処理時間が短縮されることが期待できる．

3.4　GPU を搭載した複数ノードによる並列実行

GPU を搭載した複数ノードで分散処理を行うために，**図 7**(左) のように各フレームの処理を全ノードで分担する手法を用いた．このように空間方向で分割することで，各ノードが必要とするカーネルの数が削減でき，より大きなターゲットデータを扱うことが可能になると期待できる．また，ターゲットの時間方向の大きさには依存しないため，ストリーム入力にも対応することが可能になる．

一方，ターゲットデータは，カメラや単一ノード上に保存されたファイルより与えられることを想定すると，単一ノードで取得し，全ノードに分配する必要がある．出力についても，同じように集約することを考えると，図 7(右) のように，以下のような処理が必要となる．

Step 1　入力データの読み込み (単一ノード)
Step 2　入力データの分配 (ノード間通信)
Step 3　GPU での計算 (全ノード)
Step 4　出力データの集約 (ノード間通信)
Step 5　出力データの書き出し (単一ノード)

本研究では，各ノードで 1 プロセスを実行し，各プロセスが上記の処理を行う別々の 5 つのスレッドを起動する実装とした．各スレッドは 1 フレームごとに処理を行い，処理が終

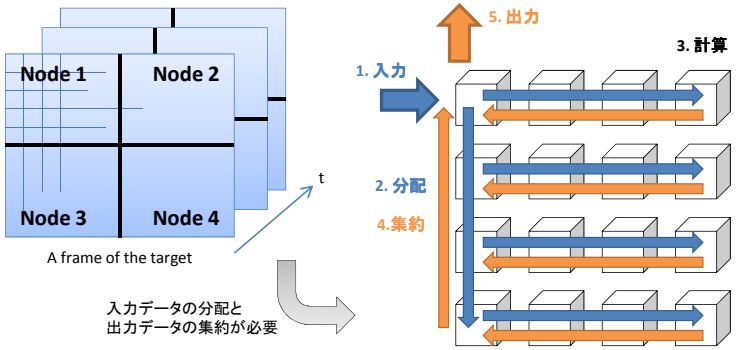

図 7　複数ノードによる並列実行
Fig. 7　Distributed processing on the multiple nodes.

了したフレームの情報は，スレッド間の通信用キューを用いて次のスレッドに通知される．これにより，それぞれの処理がオーバーラップされて実行されるため，通信にかかる時間が計算にかかる時間に比べて短ければ，高い並列化効率を得られることが期待できる．

カーネルのデータは処理の途中で変化することがないため，シミュレーションの初期化処理でまとめて GPU 上のメモリに配置する．

通信量は “入力データの大きさ” に，計算量は “入力データの大きさ” と “1 時空間座標ごとにかかる計算量” との積に比例する．1 時空間座標ごとにかかる計算量はカーネルの大きさに比例し，今回の実験で用いたカーネルでは，1 万回以上 ($15\times15\times20\times2$(Multiply+Add)+[カーネルの計算に必要な計算量]) の浮動小数点計算が 1 時空間座標ごとに必要となる．仮に 1 万回とした場合，1TFlops の演算性能があれば，1 時空間座標あたりの計算時間は 10^{-8} 秒，一方で通信は，分配と集配で 8 バイトの転送が必要なため，Gigabit Ethernet 2 リンクで 1 時空間座標あたり 4×10^{-9} となる．このように理論値でも通信時間の方が一桁小さく，また，演算性能はメモリレイテンシその他の影響で大きく理論値を下回ることが考えられるため，今回の実験の規模 (C1060 を搭載した PC ノード 16 台) では，高い並列化効率を得られる可能性が高い．

4.　評　　価

複数畳み込みの一括実行，ノード内通信の削減による効果については，ノード間通信による影響を排除して観察を行うために，1 ノードのみでの実験を行った．また，異なるアーキ

表 3　ハードウェア・ソフトウェア環境 (NVIDIA C1060 搭載 PC)
Table 3　The hardware and software specification about the PC equipped with NVIDIA C1060.

プロセッサ	Intel Xeon Quad-Core CPU W3520 (2.67GHz)
メモリ	6GB
GPU	NVIDIA C1060 (GT200 アーキテクチャ)
Streaming Processor コア数	240
プロセッサコアの周波数	1.296GHz
単精度浮動小数点演算性能	933GFlops
メモリ	4GB
メモリ帯域幅	102GB/秒
システム I/F	PCI Express x16 Generation 2
OS	CentOS 5.3
C Compiler	Intel C compiler 11.1
CUDA	CUDA Toolkit 3.2

表 4　ハードウェア・ソフトウェア環境 (NVIDIA C2070 搭載 PC)
Table 4　The hardware and software specification about the PC equipped with NVIDIA C2070.

プロセッサ	Intel Xeon Quad-Core CPU W5667 (3.07GHz) x 2
メモリ	24GB
GPU	NVIDIA C2070 (Fermi アーキテクチャ)
CUDA コア数	448
プロセッサコアの周波数	1.15GHz
単精度浮動小数点演算性能	1.03TFlops
メモリ	6GB
メモリ帯域幅	144GB/秒
システム I/F	PCI Express x16 Generation 2
OS	Fedora 11
C Compiler	Intel C compiler 11.1
CUDA	CUDA Toolkit 4.0

テクチャの GPU である NVIDIA C1060 と C2070 を搭載した PC(**表 3**，**表 4**) を用いて処理時間を計測することで，GPU のアーキテクチャの違いにより性能の傾向に変化がないかどうかを確認することを試みた．

一方，GPU を搭載した複数ノードによる並列実行による効果については，NVIDIA C1060 を搭載した PC(性能は，表 3 に同じ)16 台を Gigabit Ethernet 2 リンクで接続した環境を用いて計測を行った．比較的低速なインターコネクトを用いた環境ではあるが，前述のように，通信にかかる時間は計算にかかる時間よりも十分に小さくなると予測でき，並列化効果は期待できる．また，このように比較的安価なインターコネクトを用いた環境で高速化が実現できれば，実用性も高いと評価できる．ノード間通信には OpenMPI 1.4.1 を用いた．

また，すべての実験において，カーネルのサイズは，初期実験のため小さすぎず大きすぎないものとして $15 \times 15 \times 20$ というサイズを選択した．これより，“カーネルの値の計算” と “カーネルの値とターゲットの値の積和の計算” を合計した畳み込み 1 回あたり，1 時空間座標あたりに必要な浮動小数点演算回数は，13800 回となる．シミュレーションは単精度浮動小数点の精度での計算で十分な精度を確保できる．C1060 では，倍精度浮動小数点を用いた場合の演算性能が単精度浮動小数点を用いた場合に比べて $\frac{1}{10}$ 以下と非常に低いことを考慮し，すべての演算を単精度で行うこととした．

ノード内通信の削減による効果を計測するに当たっては，物体の移動速度 (オプティカルフロー) の導出モデルとして検討されているモデルを想定して実験を行った．このモデルは，前出の図 6 に示した形のモデルで，3 つの畳み込みに加えて 12 回の浮動小数点計算を各フレームの各空間座標ごとに行う．これにより，前述の 13800 回に比べて非常にわずかな演算回数の増加で，出力データの大きさを $\frac{2}{3}$ に縮小する．

今後様々な解像度での実験を行っていくことを予定しているが，今回の評価では，インターネットなどで一般的な 320×240 のフレームサイズを持つ動画データを入力とした実験を行った．複数ノードでの並列実行では，このサイズを 640×480 まで変化させた時の性能の変化を観測した．選択したフレームサイズは縦も横も 16 で割り切れる，動画データの解像度として一般的に用いられているものを選択した．

5. 結　　果

5.1 複数畳み込みの一括実行による効果

複数の畳み込みを 1 つ実行した場合と 2〜4 つ並列に実行した場合の処理時間の違いを**図 8** に示す．

1 つの畳み込みだけを実行している場合は，演算性能も高く，メモリ帯域幅も広い C2070 の方が C1060 に比べ実行時間が短い．テクスチャを使用してはいるが，データの再利用性が低いために，テクスチャのキャッシュヒット率はあまり高くないと考えられる．そのために，演算コアのストールは大量に発生していると考えられ，C1060，C2070 ともに単位時間当たりの演算性能は理論値に比べ数%と非常に小さくなった．

そして，複数の畳み込みを一括実行した場合だが，2 つの畳み込みを一括実行した時点で，C1060 と C2070 の性能はほぼ同じとなり，3 つ以上の畳み込みを一括実行した場合に

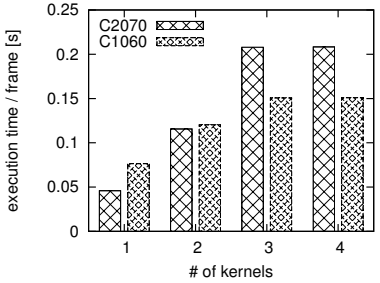

図 8 複数畳み込みの一括実行時間
Fig. 8 Execution times of multiple convolution parallel executions.

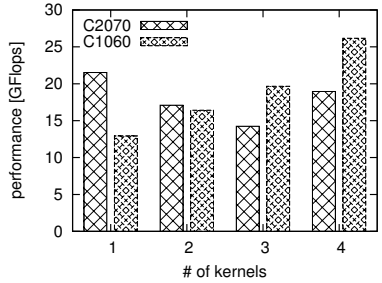

図 9 複数畳み込みの一括実行性能
Fig. 9 Performance of multiple convolution parallel executions.

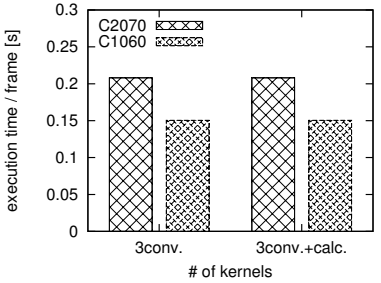

図 10 通信の削減による効果
Fig. 10 An effect of reducing communication volume.

は，C1060 が C2070 の性能を上回る結果となった．

このような結果は，C1060 と C2070 のアーキテクチャの違いによるものと考えられる．C2070 は Fermi アーキテクチャを採用している．このアーキテクチャでは，従来に比べて大きな共有メモリや，従来はなかった L1 キャッシュ，テクスチャでも利用できる大容量の L2 キャッシュを利用できる．これらを活用することにより，大きなメモリレイテンシによる性能の悪化を防ぐことができる仕組みになっている．テクスチャによる読み込みアクセスにおいても，オフチップメモリへのデータの書き出しにおいても 768KB の L2 キャッシュが働くことにより，1 畳み込み時の性能が大きく改善されたものと考えられる[⋆5]．

しかし，並列に実行する畳み込みの数を 2，3，4 と増加させ，テクスチャから一度に読み込むカーネルデータの量が 2 倍，4 倍，4 倍となると，C2070 の性能は急激に悪化した．これは，1 度に読み込まれるカーネルデータの量が増えたことにより，キャッシュの効果が少なくなったことによるものと考えられる．Fermi アーキテクチャでサポートされた ECC(Error check and correction) の影響で，実効メモリ帯域幅が物理的な帯域幅よりも小さくなっていることなども影響しているものと考えられが，詳細な原因の究明とチューニングの作業は今後の課題である．

単位時間当たりの浮動小数点演算回数で示した性能は**図 9** のように変化しており，C1060 では期待通りに性能が向上することを確認できた．しかし，C2070 では，1 畳み込みの場合が最も性能が高く，一括実行による性能の向上は達成できなかった．

⋆5 C1060 が採用するアーキテクチャでは，テクスチャ専用のキャッシュ256KB が搭載されている．

5.2 通信の削減による効果

3 つの畳み込みを処理する場合と，3 つの畳み込みに加えて 12 回の浮動小数点計算を各時空間座標ごとに行い，出力データの量を $\frac{2}{3}$ にした場合との比較を行う．GPU 上での計算により得られる出力データは，演算コアのレジスタから GPU 内のオフチップメモリへ書き出され，その後，ホストメモリへ PCI Express によって転送される．出力データの削減は，これらの通信量を削減することができるため，処理時間が短くなることが期待された．比較結果を**図 10** に示す．

出力されるデータのサイズは，計算のために読み込まれるカーネル・ターゲットのサイズに比べて比較的小さいこともあり，影響は限定的であった．C1060, C2070 のいずれでも処理時間は短くなったが，短縮時間はわずかであった．

5.3 GPU を搭載した複数ノードによる並列実行による効果

複数ノードによる並列実行による効果を**図 11** に示す．実験は，320×240，512×384，640×480 の 3 種類の入力サイズで行った．

小さな入力サイズ (320×240) では，1GPU で処理するデータ量の縮小によりテクスチャのキャッシュの影響などと考えられる 1 以上の並列化効率が 2〜4 のノード数で観測された．具体的には，2 ノードの場合に 2.5 倍，4 ノードの場合に 5.1 倍の性能向上が観測された．

どの入力サイズでも，8 ノードまではノード数に比例して性能が向上した．12 ノード以上では，データの分配・集約にかかる通信時間が計算時間を上回り性能向上が頭打ちとなった．データの分配・集約に必要な通信のタイミングの最適化による高速化が今後の課題となる．

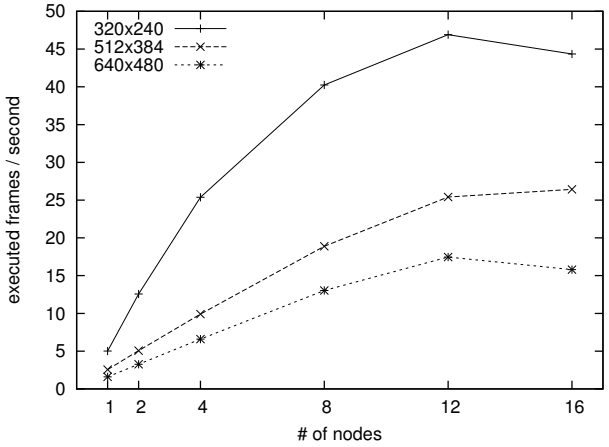

図 **11** 複数ノードによる並列実行性能 (frames per second)
Fig. 11 The number of executed frames per second when using multiple nodes.

6. おわりに

本稿では，視覚神経系のシミュレーションを GPU を用いて行うプログラムを実装し，評価を行った結果について報告した．カーネルが空間座標ごとに異なる点など，視覚神経系のシミュレーションに特有の問題を取り扱うために行った実装の効果が，従来のアーキテクチャの GPU では観察できたが，新しいアーキテクチャの GPU では観察できなかったことを示した．

Non-separable なカーネルの扱いや，GPU のハードウェアの特性に合わせたチューニング，Separable なカーネルについても計算量を減らすためのアルゴリズムの工夫など，高速化のために更なる工夫を検討し実験を行っていく必要があると考えられるため，今後も研究を進めていく．

また，GPU を搭載した PC クラスタ上で，MPI を用いて畳み込みの並列・分散処理を行うことにより，8 ノードまではノード数に比例して性能が向上することを確認できた．この結果，$15 \times 15 \times 20$ という大きさのカーネルで，実際のモデルを想定した実験を行った場合に，フレームサイズが 320×240 の動画データにおいて 40〜50FPS の性能を得ることができた．640×480 の比較的大きな動画データにおいても 15FPS 程度の性能が得られており，通信と計算のオーバーラップをより効率的なスケジューリングで行うことができれば，さらに処理時間を短縮できるのではないかという観測が得られた．

インターコネクトに用いている Gigabit Ethernet の帯域幅と実際にノード間で転送されるデータ量からは，フレームサイズが 640×480 の時に 1 フレームごとに必要な通信時間は，理論値で約 0.01 秒になる．本稿で示した FPS はこの理論値の 15%程で頭打ちとなっており，さらなる最適化により性能が改善する余地は十分にあると考えられる．よって，この点についてはさらなる改良を検討していきたい．

また，さらに大規模な階層構造を持つシミュレーションについても検討を行っていく予定である．

参 考 文 献

1) Markram, H.: The Blue Brain Project, *Neuroscience*, Vol.7, pp.153–160 (2006).
2) Li, Z.: A neural model of contour integration in the primary visual cortex, *Neural Computation*, Vol.10, pp.903–940 (1998).
3) Motoyoshi, I. and Kingdom, F. A. A.: Differential roles of contrast polarity reveal two streams of second-order visual processing, *Vision Research*, Vol. 47, pp. 2047–2054 (2007).
4) Satoh, S. and Usui, S.: Computational theory and applications of a filling-in process at the blind spot, *Neural Networks*, Vol.21, pp.1261–1271 (2008).
5) 齋藤祐典，佐藤俊治，大村純一，三好健文，入江英嗣，吉永努：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，情報処理学会研究報告，Vol.2011-HPC-129, No.4, pp.1–8 (2011).
6) NVIDIA: CUDA Toolkit. http://developer.nvidia.com/cuda-toolkit-sdk.