社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－ARC－182 (6)
2009－HPC－119 (6)
2009／2／26

グラフィックプロセッサを用いた自己組織化マップの実装と評価

設樂明宏† 西川由理††
吉見真聡†† 天野英晴††

本稿では，NVIDIA 社製のグラフィックプロセッサを用いた自己組織化写像 (SOM) のアルゴリズムを実装と性能評価について述べる．プログラムには NVIDIA 社の提供する CUDA を用い，並列プログラミング，データフローの最適化よびプロファイリングを行った．評価にはグラフィックカードを 3 種類用い，ストリームプロセッサ数やメモリ容量と性能の関係について調査した．また問題 (マップ) サイズおよび次元数，学習係数などの SOM に関するパラメータを変化させ，実行速度への影響を調べた．その結果，GTX280 を用いた場合，マップサイズ 1372×1372, ベクトル数 128, 学習サイズ 128 個のとき、Intel Core 2 Quad 2.40GHz と比べて 150 倍程度の性能が示された．

Implementation and Evaluation of Self-Organizing Map Algorithm on a Graphic Processor

Akihiro Shitara,†† Yuri Nishikawa,†† Masato Yoshimi††
and Hideharu Amano††

In this paper, we introduce an implementation of algorithm for self-organizing map(SOM) using GPUs and discuss its evaluation. We used CUDA provided by NVIDIA Corporation for parallel programming, profiling, and data flow optimization so as to exploit inherent data-level parallelism of the algorithm. By using three NVIDIA's graphic cards for evaluation, we investigated the relationships among the number of processor elements, amount of memory device and performance. As the result of performance evaluation with various parameter combinations, we found that implementation on GTX280 achieved 150 times higher performance of Intel Core 2 Quad 2.40 GHz when parameters of map size, number of vectors and learning size were 1372×1372, 128 and 128, respectively.

1. はじめに

自己組織化マップ (Self-Organizing Maps: SOM) は，1989 年に Kohonen[1] が提案した教師なし学習ニューラルネットワークである．その用途は，ロボット制御や画像処理など幅広い[2]．

従来，SOM のアルゴリズムは，SIMD またはシストリックアレイ構成のクラスタを用いて実行されてきた[3][4]．またクラスタ計算機の運用コスト等の問題に対処するため，2000 年以降は高性能化した FPGA を用いた専用ハードウェアの実装例が登場し[5][6]，汎用 CPU に対して数十から数百倍の高速化が可能であることが確認されている．しかし，これらの実装例は，対応する問題 (マップ) のサイズが小さく，低い実用性の改善が課題となっていた．

本研究では，SOM のアルゴリズムが持つ高いデータレベル並列性を活かし，グラフィックプロセッサを用いた実装を行ない，その性能を評価した．プログラミングには NVIDIA 社の提供する CUDA (Compute Unified Device Architecture) を用い，パイプラインレイテンシの隠蔽や，結合アクセス，分岐命令の削減といった最適化を行うとともに，実行時間のプロファイリングを行った．評価では，同系列でピーク性能の異なるグラフィックプロセッサを 3 種類用いた．その結果，ローエンドなグラフィックプロセッサを用いても，対応するデータサイズは大きく，高い実用性とコスト対性能比を実現した．ハイエンドな NVIDIA GeForce GTX280 を用いた場合，Core 2 Quad 2.40 GHz の約 150 倍の性能であることを確認した．

† 慶應義塾大学理工学部
Faculty of Science and Technology, Keio University
†† 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University

2. SOM : Self Organization Map

2.1 SOM のアルゴリズムの概要

SOM は，多量のデータの特徴量を抽出する目的で利用されるニューラルネットワークの一種である．

基本的な SOM では，競合層と呼ばれるニューロンが配置された空間が，順に入力される入力データを学習していく．入力ベクトル $\boldsymbol{x} = \{x_1, \cdots, x_v\}$ は，v 次元の数ベクトルである．各ニューロン i には，入力ベクトルと同じ v 次元の重みベクトル $\boldsymbol{m}_i = \{m_{i1}, \cdots, m_{iv}\}$ が設定される．また，学習の効果と範囲を示す近傍距

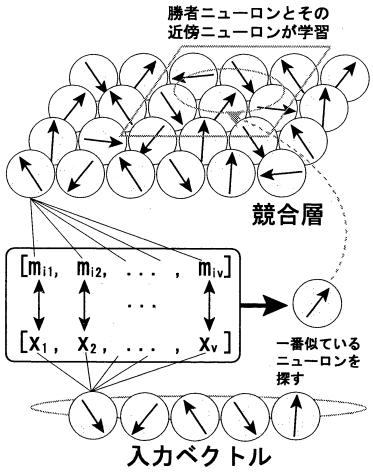

図 1 自己組織化マップ・アルゴリズムの概念図

離 N_c および，近傍関数 h，学習率係数 $\alpha(t)$ が設定される．全入力データの学習が T 回繰り返された後，競合層に学習結果が形成される．

2.2 SOM の計算手順

基本的な SOM のアルゴリズムは，以下の手順で実行される．

(1) 各パラメータおよび競合層の全ニューロンの重みベクトル $\boldsymbol{m_i}$ の値を乱数で初期化する．制御変数を設定する $t \leftarrow 0$

(2) 入力ベクトル $\boldsymbol{x}$ を与える

(3) $\boldsymbol{x}$ と各 $\boldsymbol{m}_i$ の距離 $||\boldsymbol{m}_i - \boldsymbol{x}||$ を評価し，式 (1) を満たす勝者ニューロン c を探索する

$$||\boldsymbol{m}_c - \boldsymbol{x}|| = \min_i ||\boldsymbol{m}_i - \boldsymbol{x}|| \qquad (1)$$

(4) 勝者ニューロン c とその近傍 N_c 内にある全ニューロンの重みベクトルを，式 (2) を用いて更新し，学習させる．式 (2) の近傍関数 h_{ci} は学習率係数 $\alpha(t)$ を用いて式 (3) で定義される

$$\boldsymbol{m}_i' = \boldsymbol{m}_i + h_{ci}(\boldsymbol{x} - \boldsymbol{m}_i) \qquad (2)$$

$$h_{ci} = \begin{cases} \alpha(t) & (i \leq N_c) \\ 0 & (i > N_c) \end{cases} \qquad (3)$$

(5) 次の入力ベクトルの処理を (2) へ戻って繰り返す．全入力ベクトルの計算が終了した場合，$t < T$ ならば $t \leftarrow t+1$ の後に (2) へ戻って最初の入力ベクトルから繰り返す．$t = T$ ならば計算終了

SOM の場合，一般的に距離 $||\boldsymbol{a} - \boldsymbol{b}||$ の計算にはユークリッド距離を用いる．

3. GPU アーキテクチャ

本研究では，SOM のプラットホームとなる GPU として，Geforce GTX280，9800GTX+，および 9600GT を用いる．GTX280 は GT200 シリーズ，

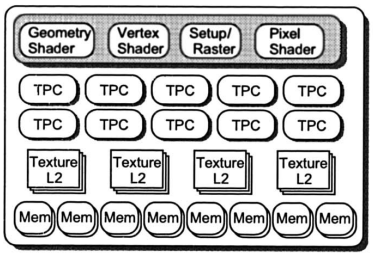

図 2 GT200 シリーズの構造

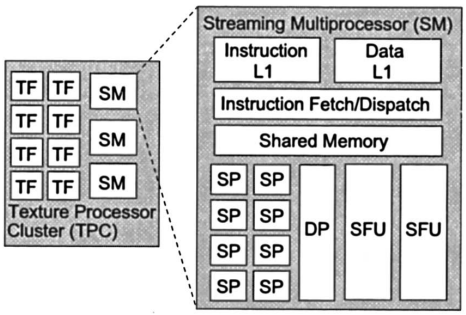

図 3 Texture Processor Cluster(TPC) の構造

9800GTX+と 9600GT は G80 シリーズに大別され，特に前者は倍精度浮動小数点演算をサポートしていることが特徴である．その点を除き，両シリーズはハードウェア構成および制御方式が類似しているため，ここでは GT200 シリーズを例にとって解説する．

3.1 グラフィックプロセッサアーキテクチャ

3.1.1 Texture Processor Cluster (TPC)

図 2 に，NVIDIA 製グラフィックカードに搭載されている GT200 シリーズデバイスのアーキテクチャの概観を示す．また以下に主な構成を述べる．

- Texture Processor Cluster (TPC)
- スレッド実行マネージャ
- デバイスメモリ
- L2 キャッシュ

図 2 における Texture Processor Cluster (TPC) は，複数のコアを持つ演算処理部である．2009 年 1 月現在，GT200 シリーズには GTX280 と GTX260 プロセッサがあり，前者は TPC を 10 個，後者は 8 個持つ．各 TPC 内部には，Texture Filtering Unit(図中の TF) および複数個の Streaming Multiprocessor(SM) を持つ．各 SM の構成を以下に述べる．

- Streaming Processor(以下，SP)(8 個): 単精度型浮動小数演算パイプライン
- Special Function Unit(SFU)(2 個): 超越関数計算用演算パイプライン
- DP(1 個): 倍精度型浮動小数演算パイプライン
- 共有メモリ: 8 個の SP で共有されるオンチップメモリ
- 命令およびデータ用 L1 キャッシュ

- プログラムカウンタ

3.1.2 SIMT アーキテクチャ

NVIDIA 社製の GPU の制御方式は，Flynn による 2 種類の分類にあてはめることができる．GPU アーキテクチャ全体の制御方式は，各 TPC が独自のデータを用いて独立したスレッドを実行できるため，MIMD 型 (Multiple Instruction, Multiple Data) として捉えることができる．一方，各 SM は後述する SIMT(Single Instruction, Multiple Thread) と呼ばれる方式にて制御される．

SM 内部の 8 個の SP では，プログラムカウンタが共有され，1 サイクル内に同一の命令を実行する．これは 1 サイクル単位で見れば，SIMD(Single Instruction, Multiple Data) 制御方式のように見える．しかしここで，SM が命令を発行するクロックレートは TPC の 1/3 程度である点に注意する必要がある．TPC が SM に対して 1 サイクル毎に命令を発行すると，SM 内の演算パイプライン稼働率が低下する．そこで演算効率を向上させるため，各 SM は図 4 に示すように，同一の命令を複数サイクルに渡って実行する必要がある．例えば，あるサイクルにて SP0 がスレッド 0 を，SP1 がスレッド 1 を実行する場合，次のサイクルではそれぞれスレッド 8，スレッド 9 を実行する．この同一の命令による複数スレッドの実行を，SIMD に代わって SIMT と呼ぶ．GTX200 シリーズでは，一命令が 8 個の SP により 4 サイクル，すなわち 32 スレッド実行したとき，最も高い演算効率が得られる．この 32 スレッドの単位は warp と呼ばれる．

なお，CUDA の利点として，スレッド数を明示することなくプログラムを記述できることが挙げられる．よってプログラマは，スレッド毎の例外処理 (分岐後の振舞いなど) で異なる点があれば，それを記述するのみでよい．この特徴は，SIMD 処理の並列度を意識する Intel SSE 命令を用いた SIMD プログラミングとは大きく異なる．

3.2 高速化のテクニック

3.2.1 パイプラインレイテンシの隠蔽

Vasily らは，$a = a*b+c$ や $a = a*b+c$ のような

表 1 GPU の諸元

GPU name	GeForce GTX280	GeForce 9800GTX+	GeForce 9600GT
マルチコア数	30	16	8
マルチコア数/TPC	3	2	2
コアクロック (GHz)	1.27	1.84	1.60
レジスタ/コア	64KB	32KB	32KB
共有メモリ/コア	16KB	16KB	16KB
メモリクロック (GHz)	1.1	1.1	0.9
メモリバス幅 (bit)	512	512	256
メモリプロセッサ間 バンド幅 (GB/sec)	141.7	70.4	57.6
メモリ容量	1GB	512MB	512MB

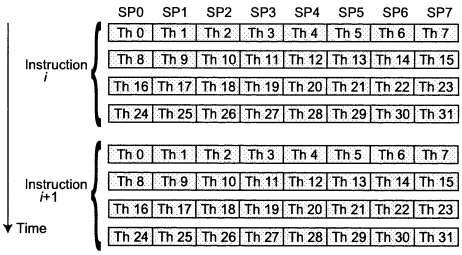

図 4 warp によるスレッドの実行

命令レベル並列性を引き出すことの困難なレジスタ-レジスタ間演算を，ループアンローリングした状態で反復実行し，パイプラインレイテンシを見積っている．その結果，レイテンシは 24 サイクルであることがわかり，これを隠蔽するために，SM あたり最小 6 つの warp を実行する必要があることを示した[7]．この値は本研究における実装にも適用されている．

3.3 結合アクセス

前節にて述べたように，SM 内部の各 SP は SIMT 方式であるため，メモリアクセス命令も同一に行われる．このとき，アクセス対象のメモリ番地が連続するケースが多いため，個々ではなくまとめてアクセスすることにより高速化を図るデータ転送方式を，結合アクセスと呼ぶ．

G80/GT200 シリーズでは，アライメントされた連続する 64 バイトのデバイスメモリ領域に対し，16 スレッド単位で協調しながらアクセスする場合が最も高効率であり，同一のデータ量を単一に転送する場合と比べて 10 倍程度の高速化が可能である[8]．

3.3.1 分岐の削減

1 つの warp に含まれる全てのスレッドが，同一のパスを実行する様子を図 4 に示す．ここで，実行中の命令が条件分岐命令であり，warp 内のスレッドによって次に実行するパスが異なる場合を考える．1warp は SIMT 実行されるため，異なる命令は同時に処理することができない．まず一方のパスをとるスレッドのみを含んだ warp を実行し，次のサイクルにて片方のスレッドを含む warp を実行することで両者の分岐先の命令実行時間が生じる．条件分岐の少ないプログラムを記述することは，GPU 高性能を達成する上で重要である．

4. 関連研究

4.1 SOM 用ハードウェアアクセラレータ

SOM のハードウェアアクセラレータに関する研究が現在までに行われている．1987 年に Carpenter らにより SOM のハードウェア実行に関する検討が行われた[3]．1992 年には Speckmann らが，並列に動作する 8 つの計算ユニットで構成される SOM 用に設計された SIMD 型プロセッサ COKOS(COprocessor

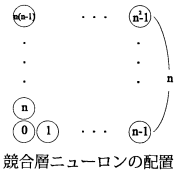

図 5 競合層ニューロンの配置

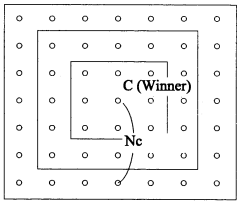

図 6 学習範囲の定義

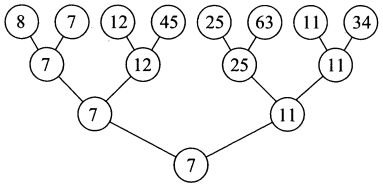

図 7 リダクション演算

for KOhonen's Self-organizing map) を開発している[4]. 1996 年には Ruping らによって SOM 用プロセッサ NBISOM25 が開発され, 定量的評価が行われた[9].

FPGA を用いた SOM 用アクセラレータは, Porrmann ら, Tamukoh らによる実装がそれぞれ開発されている[5][6]. Porrmann らは, リング状に接続された 6 つの計算コアで SOM を実行するシステムを開発した. Xilinx XCV812E-6 上での実装で, ベクトル次元数 9, マップサイズ最大 250×250 の SOM を実行でき, 4400MCUPS の性能を確認している. Tamukoh らは, SOM の学習過程において, 入力ベクトルと重みベクトル間の距離をマンハッタン距離で定義することによって, 演算の簡素化を図った.[6] これにより, 実装した際の回路面積を削減することに成功し, 16 ビット幅データで 128 次元, 16×16 のサイズの SOM を単一の Xilinx XC2V6000-C FPGA 上に実装した. この中で, 1 つのマップに対して, 距離と学習を行う回路ユニット, 重みベクトル格納用のメモリを組み合わせた回路を多数並べることで, 並列計算を可能にした. 結果, マップサイズの変化から影響を受けない一定の時間で計算できることが示され, Intel Xeon 2.80 GHz CPU に対して約 350 倍の CUPS 値 (17500MCUPS) を達成した.

5. 実　　装

5.1 自己組織化マップのデータ表現

競合層の重みベクトル, 入力ベクトルは全て整数型の 1 次元配列として表す. また, 競合層ニューロンは, 図 5 のように $n \times n$ の正方形状に配置し, 0 から順に番号をつける. 全ての重みベクトルのデータは, 結合アクセスを可能にするため, 同一の属性のデータの構造体を並べた Structure of Array(以下 SoA) のデータとして表す.

配列の大きさは, パイプラインレイテンシを隠蔽できる数に, スレッド数は結合アクセスを行いやすい数に設定する. プロセッサの SP 演算ユニットのパイプラインレイテンシを隠蔽するのに必要なブロックあたりの最低スレッド数である 192 スレッドに設定し, 確実に隠蔽する. 結合アクセスは, 16 個のスレッド (64KB) 単位で行うことができるので 192 スレッド数という値はレイテンシを最小化し, 最適な結合アクセスを行うための条件を満たす. ニューロンを SoA のデータとして表現するために, 1 つの属性の配列の長さを 192 個 (768 バイト) 単位で増やしながら十分な長さの配列長を決定する. さらに同じ大きさのメモリ領域を確保し, その先頭から次の属性の重みベクトルの要素を順に格納する.

また, 勝者ニューロンの近傍距離 N_c を図 6 のように定義し学習させる.

5.2 カーネル

SOM のアルゴリズムにしたがい, 下の 3 つのカーネルを実装した.

5.2.1 calc_euclid_kernel()

calc_kernel() は, 入力ベクトルと全ての重みベクトルとのユークリッド距離を求めるカーネルである. 以下に, 1 つの競合層ニューロンに割り当てられたスレッドの動作を示す.

(1) 重みベクトルの 1 次元分の要素をデバイスメモリから読み出し, レジスタに格納する. ここでは, スレッドが必ず結合アクセスによって読み出される.

(2) 重みベクトルの要素と入力ベクトルの各要素の差の自乗和を求める.

(3) 1 と 2 を次元数の回数だけ繰り返す.

結合アクセスの増加や, 分岐による実行する warp 数の増加を避けるなどの理由から, SoA 内の重みベクトルの要素として実際には利用されていないデータについても自乗和を求める. また, 計算量を減らすために, 自乗和の値を用いて大小を比較する.

5.2.2 get_winner_kernel()

get_winner_kernel() は, calc_euclid_kernel() で求めたユークリッド距離の配列の中から最小値をもつインデックスを探し出すカーネルである. 最小値は, 図 7 のように, 二分木のデータ構造を利用してリダクション演算を行う. GPU は, 1 つのブロックに最大 512 個のスレッドを生成することができるので, 1024 個ずつにデータを分け, 繰り返し演算を行う. 以下は, 1 つのスレッドの動作である.

(1) ブロック内のユークリッド距離のデータが 1024 個埋まっていなければ, GPU で表せる整数型の最大の値を代入する. ここで, アライメントされて連続した 64 バイトのデータを読み出すのであれば, 結合アクセスによって高速にアクセスが可能である。

(2) 配列前半の 512 データと後半 512 データについてを 1 つずつ大小関係を比較し，前半 512 個の領域小さい方の値を書き込み，後半 512 個にはそのニューロンの番号を書き込む．そして 1024 個から 512 個へのリダクションを行う．

(3) 以降，512→ ... → 4→2→1 個と，リダクションを行いながら小さい値を選び，ブロック内での最小値を決定する．

5.3 learn_kernel()

learn_kernel() は，勝者ニューロンとその周辺のニューロンの重みベクトルの値を更新し，学習させるカーネルである．calc_euclid_kernel() と同様に，1 スレッドに 1 ニューロンの更新を割り当てる．以下は，1 つの競合層ニューロンに割り当てられたスレッドの動作である．

(1) スレッド ID，勝者ニューロンの位置を元に，割り当てられたスレッドのニューロンが，学習させる範囲の中でどの位置にあるのかを求める．

(2) 割り当てられたスレッドのニューロンがマップサイズ全体でどの位置にあるのかを求める．

(3) 割り当てられたスレッドのニューロンの位置がマップ全体からはみ出していなければ，次元の低い方から順に重みベクトルの要素を更新する．

6. 性能評価

設計した SOM の CUDA プログラムを用いて，マップサイズ，ベクトル次元数，学習させる範囲の大きさと，カーネルの実行速度，メモリ-マルチプロセッサ間の転送バンド幅の関係について評価を行った．本稿では，表 2 の実行環境で評価を行った．

6.1 パラメータ

本研究報告で設計した SOM のプログラムは，以下の 3 つの項目をパラメータ化している．

- 競合層ニューロンマップの 1 辺の個数: n
- ベクトル次元数: dim
- 勝者ニューロンの近傍距離 N_c

これらはいずれも，プログラム実行時の引数として与えられ，3 つの値のみで確保するメモリの領域や各カーネルのスレッド数やブロック数といった変数が決まる．

6.2 近傍距離と実行時間の関係

ベクトル次元数を 128，マップ数を 1372×1372 とし，近傍距離 N_c を変化させたときの GeForce 280GTX での実行時間を図 8 に示す．

表 2 実行環境

CPU	Intel Core 2 Quad Q6600 2.40GHz
Memory	4GB
OS	Linux 2.6.23 (Fedora 8)
CPU コンパイラ	gcc-4.1.2
デバイス実行環境	CUDA 2.0
GPU コンパイラ	nvcc 2.0
GPU 測定環境	CUDA Visual Profiler 1.0

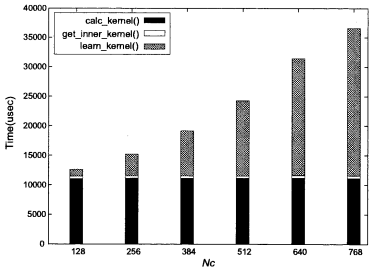

図 8 学習サイズ Nc と実行時間の関係

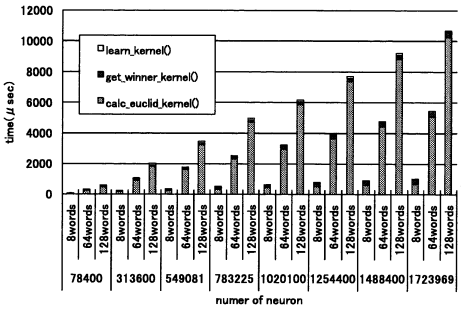

図 9 マップサイズ，ベクトル次元数の変化と実行時間の関係

6.3 マップサイズ，ベクトル次元数の実行時間との関係

SOM のマップサイズ，ベクトル次元数を変化させた場合の実行時間を図 9 に示す．GPU は GeForce GTX280 を使用し，近傍距離 N_c を 128 とした．競合層ニューロン数の増加に対して，calc_euclid_kernel()，learn_kernel() が線形に増加している事が確認できる．

getWinner_kernel() は，リダクションにより最小値を求めるので，ニューロン数の増加に対して，$O(n \log n)$ で増加するが，128 次元ベクトルでの実行において，GeForce GTX280 上で実行可能な最大ニューロン数が 1400×1400 程度なので getWinner_kernel() は最大でも 3 回のみの実行となるため全体での割合は少ない．

ベクトル次元数の増加に対しても calc_euclid_kernel()，learn_kernel() が線形に増加する．これは，ユークリッド距離を求める際の積和演算や，重みベクトルの要素の更新が，ベクトルの次元数の回数で行われるからである．

6.4 CPU との実行速度比較

表 1 の 3 枚のグラフィックカードを用いて，CPU との実行時間の比較を行った．ベクトル次元数を 128，近傍距離 N_c を 128 とし，マップサイズを変化させたとき，CPU と GPU の実行時間比の変化を図 10 に示す．値は (CPU の実行時間 / GPU の実行時間) として示す．

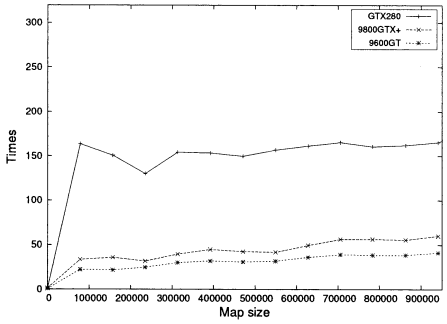

図 10 CPU と GPU の実行時間比

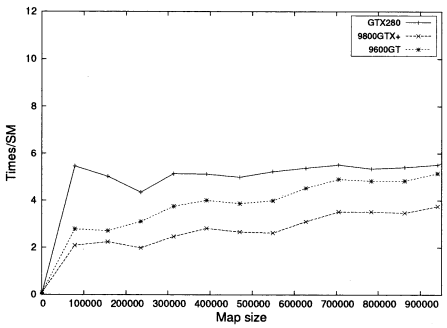

図 11 SM 1 個あたりの性能向上

さらに，図 10 の値を各 GPU に搭載される SM の数で割ることで，SM1 個あたりの SOM の計算性能の向上を求めた．図 11 に示す．

6.5 FPGA との比較

Tamukoh の実行速度と比較する．比較には，CUPS (Connection Updata Per Second) という，1 秒間に書き換え可能なニューロン数を表す単位を用いる．Tamukoh の実装は，マンハッタン距離での比較により勝者ニューロンを決定する．その結果，128 次元のベクトル，1154×1154 のマップサイズにおいて，68GCUPS であった．Tamukoh による実装は，128 次元のベクトル，16×16 のマップサイズで 17.5GCUPS の 3.89 倍である．FPGA の実装と比べ，実質的にマップサイズの制約が無くなり，問題の形状変動に対して高い柔軟性を持つとともに，より高速な実行が可能であることが確認された．

7. ま と め

本稿では，GPU を用いた SOM のプログラムの実装を行い，評価した．実装には，パイプラインレイテンシの隠蔽，結合アクセス，分岐の削減の 3 つの最適化を行った．マップサイズやベクトル次元数，近傍距離を変化させて 3 種類の GPU 上で実行したところ，128 次元，280×280 のマップサイズ，近傍距離 128 の条件下において，GeForce GTX280 が Intel Core 2 Quad 2.40 GHz と比べ約 150 倍高速であることを確認した．さらに，128 次元のベクトルのマップで 68GCUPS の性能を示し，Virtex XC2V6000 FPGA を用いた実行時間と比較したところ，3.89 倍の性能が得られることがわかった．本実装は，実質的にマップサイズの制約を解消し，問題の形状変動に対する高い柔軟性と，高速性を併せ持つことが確認された．

謝 辞

本研究の一部は，国立情報学研究所共同研究「ネットワークオンチップにおけるストリーミング処理に関する研究」による．

参 考 文 献

1) Kohonen, T.: *"Self-organization and associative memory"*, Springer-Verlag New York, Inc. New York, NY, USA (1989).
2) 伊藤則夫, 白木渡, 安田登: "地盤物性値の空間分布推定問題への自己組織化特徴マップの応用", 土木学会論文集, Vol. VI-47, No. 651, pp. 145–156 (2000).
3) Carpenter, G. and Grossberg, S.: "A massively parallel architecture for a self-organizing neural pattern recognition machine", *Computer Vision, Graphics, and Image Processing*, Vol. 37, No. 1, pp. 54–115 (1987).
4) Speckmann, H., Thole, P., Rosentiel, W., Aleksander, I. and Taylor, J.: "Hardware Implementations of Kohonen's Self-Organizing Feature Map", *Artificial Neural Networks, 2*, Vol. 2, pp. 1451–1454 (1992).
5) Porrmann, M., Franzmeier, M., Kalte, H., Witkowski, U. and Ruckert, U.: "A Reconfigurable SOM Hardware Accelerator", *Proceedings of the 10th European Symposium on Artificial Neural Networks, ESANN*, pp. 337–342 (2002).
6) Tamukoh, H., Aso, T., Horio, K. and Yamakawa, T.: "Self-organizing map hardware accelerator system and its application to realtime image enlargement", *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, Vol. 4 (2004).
7) Volkov, V. and Demmel, J. W.: "Benchmarking GPUs to Tune Dense Linear Algebla", *SC'08* (2008).
8) NVIDIA: "NVIDIA CUDA Compute Unified Device Architecture" (2008).
9) Ruping, S. and Ruckert, U.: "A scalable processor array for self-organizing feature maps", *Microelectronics for Neural Networks, 1996., Proceedings of Fifth International Conference on*, pp. 285–291 (1996).