

## GPUを用いた球面SOMの実アプリケーションによる評価

西本 要<sup>†1</sup>    吉見 真聡<sup>†2</sup>  
廣安 知之<sup>†3</sup>    三木 光範<sup>†2</sup>

GPUはアルゴリズムの並列性を有効に引き出す比較的安価なシステムとして注目され、様々なアプリケーションが報告されており、今後も発展が予測される。また一方で、広く利用されており、並列性の高いアプリケーションの一例として学習型ニューラルネットワークであるSOMが挙げられ、GPUを含め、様々な高速化技術が試みられてきた。本研究報告では、GPUを用いて球面上にデータが配置される球面SOMの実装を行い、マイクロプロセッサとの実行と比較した。その結果からGPUの性質を評価するとともに、球面SOMのアルゴリズムと並列性に関する検討を行う。

### Evaluation of Spherical self-organizing map by real application software on a GPU

KANAME NISHIMOTO,<sup>†1</sup> MASATO YOSHIMI,<sup>†2</sup> TOMO HIROYASU<sup>†3</sup>  
and MITSUNORI MIKI<sup>†2</sup>

Graphic Processing Unit (GPU) is focused as a promising and comparatively reasonable computational environment to extract parallelism in an algorithm efficiently. Several applications for GPU has been implemented and reported. One such example is a Self-Organizing Map (SOM), which has high degree of data-level parallelism. The algorithm is known as a learning neural network, and various techniques including GPU implementation have been studied to accelerate the calculation of SOM. This report compares the performance of spherical-SOM between general micro-processor and several GPUs. The spherical-SOM is a modified SOM to map neurons on a surface of a sphere. In addition, we discuss the characteristics of GPU and parallelism of spherical-SOM algorithm.

### 1. はじめに

1989年にKohonenが発表した自己組織化マップ(Self-Organizing Map:SOM)は、教師なしの学習型ニューラルネットワークである。SOMは、 $n$ 次元のデータベクトルであるニューロンが正方格子状や六角格子状に配置されたマップに対して、対象となる入力データを学習させて可視化する手法である。

SOMは高次元のデータベクトルを低次元に落とし込む手法であり、医学、農学、社会科学など、様々な分野で利用されている有用なデータマイニングツールである<sup>1)</sup>。しかし最も基礎的なSOMでは、学習の不均一さが大きな問題として知られている。これは、ニューロンを平面上に配置する関係上、マップの端付近とそれ以外の部分では隣接ニューロンの数が異なり、図1に示すように学習エリアの大きさに差が出るのが原因である。この問題を解決する手法として、Kohonenによるヒューリスティックな重み付けや局所線形平滑化が提案されている。また、数学的手法以外にも、SOMのデータ構造をトーラスや球面などの閉じた曲面にする手法も提案されている。トーラスは実装が容易であるが、3D空間ではドーナツ型をとってしまい可視化ツールとしては不向きである。また、ニューロンを球の表面上に配置する球面SOMは、Ritterが提案した正多面体をもとにした測地ドームと呼ばれる準正多面体を用いたものが多く使用されている<sup>8)</sup>。測地ドームは球面状にほぼ均一にニューロンを配置することができる。しかし、平面のSOMと比べて実装が複雑となり、高速化が困難である。

SOMは入力データごとに全ての重みベクトルに対して演算を行う必要があり、演算回数が莫大になる一方で高いループレベル並列性を持っており、様々な高速化手法が試みられてきた。従来の平面SOMは、SIMDプロセッサやストリックアレイ構造の計算機による実装が試みられ<sup>2)3)</sup>、2000年頃からは大規模システムの運用コストの問題に対処するため、FPGAを用いた専用ハードウェアによる実装例も報告されている。また、近年ではGPUを用いた実装についても報告されている<sup>6)</sup>。

本研究報告では、測地ドームを正20面体の展開図と見立て、2次元平面のデータとして

<sup>†1</sup> 同志社大学工学部

Faculty of Engineering, Doshisha University

<sup>†2</sup> 同志社大学理工学部

Faculty of Science and Engineering, Doshisha University

<sup>†3</sup> 同志社大学生命医科学部

Faculty of department of life and medical science, Doshisha University



中心付近 縁付近  
図1 位置による学習範囲の違い

扱うことで球面 SOM において SOM の持つ並列性の高さを残した実装を可能とした。以上の内容により、GPU における実装と評価を行った。

## 2. SOM のアルゴリズム

### 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  $x = \{x_1, \dots, x_v\}$  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  $m_i = \{m_{i1}, \dots, m_{iv}\}$  が設定される。また、学習の効果と範囲を示す近傍距離  $N_c$  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  $m_i$  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  $x$  を与える
- (3)  $x$  と各  $m_i$  の距離  $\|m_i - x\|$  を評価し、式 (1) を満たす勝者ニューロン  $c$  を探索する

$$\|m_c - x\| = \min_i \|m_i - x\| \quad (1)$$

- (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

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

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

- (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  $\|a - b\|$  の計算にはユークリッド距離を用いる。プログラム全体の流れを図 2 にフローチャートとして示す。

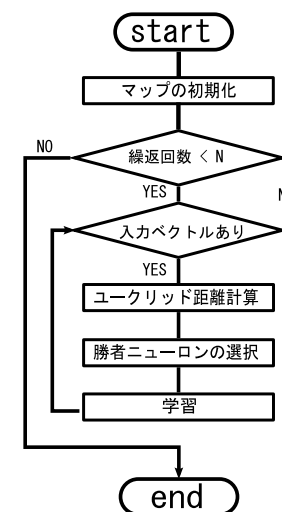


図2 SOM アルゴリズムのフローチャート

## 3. 球面 SOM のデータ構造

球面上のデータを 2 次元平面上のデータとして扱うために、測地ドームの展開図を用いる<sup>7)</sup>。測地ドームとは正多角形の 1 面を細分化することで得られる、図 3 に示すような準正多面体のことである。本研究報告では元となる正多角形として、最も面数の多い正多面体である正 20 面体を採用した。測地ドームを正 20 面体の辺に沿って展開すると図 3 のような

格子図が得られる．この格子に対し，軸  $U$  と軸  $V$  を図 4 に示したように定める．この 2 軸を直行するように歪ませ，軸  $U'$  と軸  $V'$  で表現し，各頂点が SOM マップにおいてニューロンを配置する位置と考える．また，展開図を組み立てた際に重なり合う頂点は削除する．この行程により，測地ドームを 2 次元配列上に置くことができる．

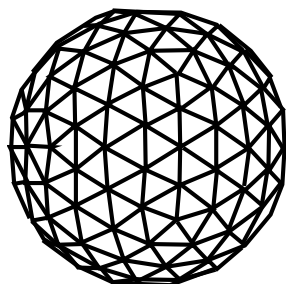


図 3 測地ドーム

#### 4. 関連研究

##### 4.1 SOM のハードウェア・アクセラレータ

1987 年に Carpenter らによって SOM のハードウェア・アクセラレータに関する研究が行われている．

1992 年には Speckmann らが SOM 用の SIMD 型プロセッサ COKOS (COprocessor for Kohonen's Self-organizing map) を開発している．COKOS は並列に動作する 8 つの計算ユニットで構成され，SOM が並列化向きであることに着目している．

FPGA を用いた SOM 用アクセラレータは，Pormann, 田向らにより実装されている．Pormann らは，リング状に接続された 6 つの計算コアを用いることで SOM を並列実装している．Xilinx XCV812E-6 上での実装では，ベクトル次元数 9，マップサイズ  $250 \times 250$  の場合に 4400MCUPS の結果が確認されている．

田向らは，入力ベクトルと重みベクトル間の距離にマンハッタン距離を用いることで演算を簡略化し，回路面積の削減に成功した．ベクトル次元数 128，マップサイズ  $16 \times 16$  の SOM を単一の Xilinx XC2V6000-C FPGA 上に実装した．各重みベクトル毎に，データを格納するメモリ，距離計算と学習を行う計算器を組み合わせた回路ユニットを対応付け，この

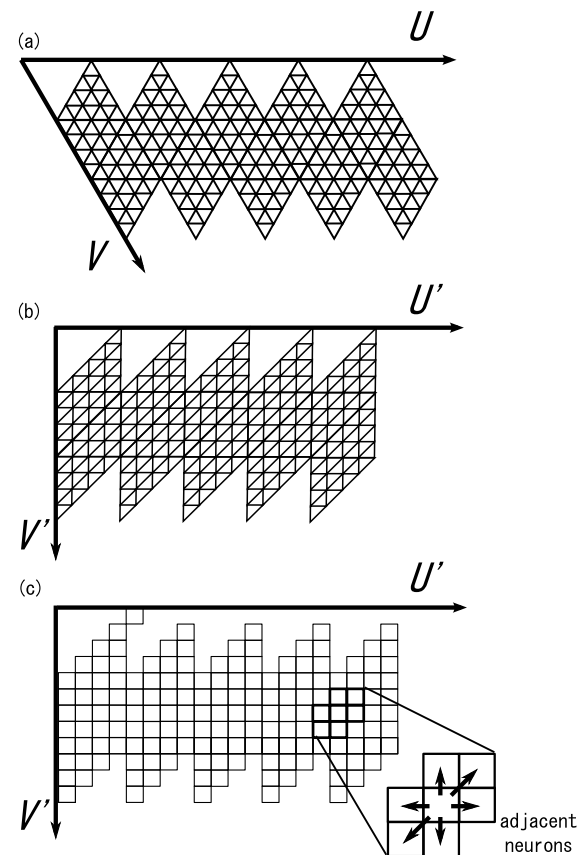


図 4 展開図から 2 次元配列への写像

ユニットを多数並べることで並列計算を可能にした。結果、マップサイズに影響を受けず、常に一定の時間で計算が可能となった。実際には Intel Xeon 2.80 GHz CPU に対して約 350 倍の CUPS 値 (17500MCUPS) まで達成している。

## 5. 実装

### 5.1 マップデータの構築

マップは 2 次元配列で表現され、図 3 で示した測地ドームの展開図を配置する。

学習の際には隣接したデータを探索する必要がある。このとき、中心付近で隣り合っているならば容易に探索が可能だが、縁付近で離れていると探索の度に計算が発生する。SOM では近傍探索を繰り返し行うため、非常に計算コストを要する。そこで、隣接するデータのアドレスをあらかじめ格納する手法をとった。

上記の処理をはじめに行うことで、SOM アルゴリズムを走らせる際の重複した計算を削除した。

### 5.2 CPU での実装

#### 5.2.1 距離計算

重みベクトルと入力ベクトルとのユークリッド距離を求める。この処理を逐次的に実行することで、全ての重みベクトルに対して入力ベクトルとの類似度を計算する。

#### 5.2.2 勝者選択

距離計算で得られたユークリッド距離を比較し、入力ベクトルとの差が最小となる重みベクトルを選択する。最小となる重みベクトルの位置を保管し、マップの最初から比較と保管位置の置き換えを繰り返すことで、ユークリッド距離が最小となる勝者ニューロンを選択する。

#### 5.2.3 近傍探索

SOM アルゴリズムの学習の行程では、勝者ニューロンからの距離計算が必要となる。距離はニューロン間の移動におけるステップ数で定義する。そこで、幅優先探索による近傍探索を行った。

勝者ニューロンをルートノードとした木構造が図 5 のように定義できる。図 5 では、ルートとの距離が勝者ニューロンからの距離を表し、線で結ばれているノードは隣接していることを意味する。上のノードから探索が行われ、隣接するニューロンのアドレスをリストに格納することで次ノードの一覧を取得する。この行程を近傍幅の数だけ繰り返すことで、近傍を探索する。

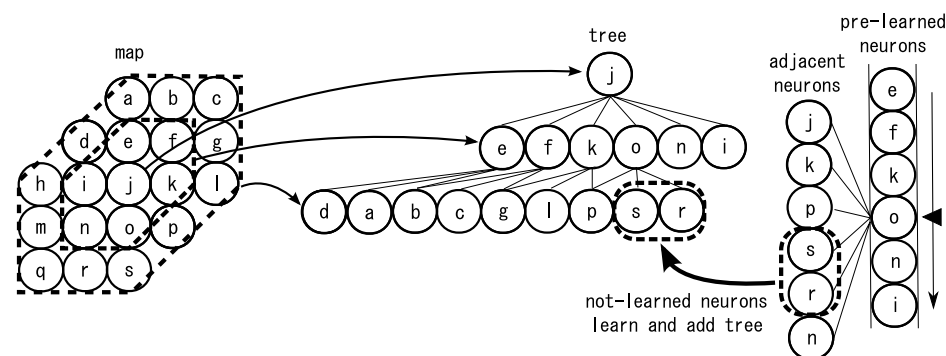


図 5 幅優先探索を用いた近傍探索

### 5.3 GPU での実装

CPU 上で動くアプリケーションを作成し、後に CUDA プログラムとして GPU で実装した。その際の方法を以下に述べる。

#### 5.3.1 距離計算の並列化

SOM のアルゴリズムの中で最も時間を必要とするのが、入力ベクトルと重みベクトルとのユークリッド距離の計算である。しかし、この行程は重みベクトル毎に独立なため、並列化が容易に実現できる。そこで、ユークリッド距離計算を GPU で並列実装することで高速化を試みた。

GPU のグローバルメモリに重みベクトルと距離を格納する配列が確保されている。重みベクトルを CPU のメモリからコピーする。コピーした重みベクトルと送られてきた入力ベクトルのユークリッド距離計算を行い、結果をグローバルメモリに確保していく。最後に計算結果の一覧を CPU のメモリにコピーし、GPU の計算を終了する。

#### 5.3.2 勝者選択の並列化

勝者選択は、先に計算したユークリッド距離の最小値探索である。CPU と同様の選択方式を用いると、 $O(N)$  の計算量が必要となり、並列化が不可能である。そこで、二分木を用いたトーナメント型の選択方式を実装する。計算量は逐次型と同じであるが、並列化が可能のため最大  $O(\log(N))$  の計算時間での実装が可能となる。

#### 5.3.3 近傍探索・学習の並列化

前節で述べたが、近傍探索には幅優先探索を用いている。近傍探索の際には、事前に用意した学習済みニューロンのリストから隣接しているニューロンのリストを作成し、学習を

表 1 GPU の性能

名称	SP 数 (個)	コアクロック (GHz)	メモリクロック (GHz)	メモリサイズ (MB)	メモリアインターフェイス (bit)	メモリ帯域 (GB/秒)
8400GS	16	0.45	0.40	256	64	6.4
GTX280	240	1.27	1.1	1024	512	141.7
C1060	240	1.30	0.8	4096	512	102.0

表 2 実行時間

	実行時間 (ms)	CPU との速度比
CPU	75170	1.00
8400GS	29340	2.56
GTX280	5320	14.13
C1060	5250	14.31

行っている。学習は重みベクトル毎に独立なため、学習済みリストの全ての隣接ニューロンの学習を並列化することが可能である。

## 6. 実験

### 6.1 概要

GPU 上で球面 SOM を動作させ、その性能を評価する。GPU には、GeForce8400GS、GeForce GTX280、Tesla C1060 の 3 種を用いる。それぞれの GPU の性能を表 1 に示す。さらに CPU とも比較を行う。比較する CPU は Opteron 1210 HE である。

SOM の設定は、ニューロンを 5 次元、重みベクトルの数を 50000(20×50<sup>2</sup>) 個、近傍幅を [0,9] とした。

### 6.2 結果

それぞれの GPU および CPU の実行時間を表 2 に示す。さらに、球面 SOM の処理手順である、SOM マップの初期化、ユークリッド距離の計算、勝者ニューロンの選択、学習のそれぞれにおける処理時間を図 6 に、処理時間の配分を図 7 示す。

表 2 から、CPU と比較して GPU は処理速度が向上していることが分かる。

## 7. 考察

### 7.1 メモリ帯域の影響

図 6 から、全体の処理時間における距離計算の割合については、8400GS の値が大きいこ

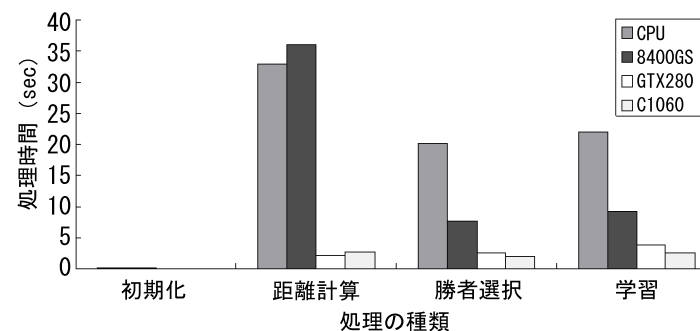


図 6 GPU ごとの実行時間

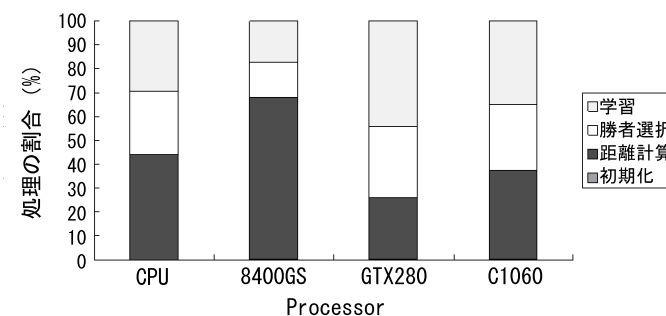


図 7 処理内容ごとの実行時間比

とがわかる。この理由は入力ニューロンの受け渡し方法に原因があると考えられる。本実装では学習を行う際に、対象となる入力ニューロンを引数として GPU 関数に渡している。そのため、全体として「試行回数 × 入力ニューロン数」のデータ送信となるため、メモリ帯域が他の GPU と比べて低い 8400GS は距離計算に時間を要したのだと考えられる。

### 7.2 近傍探索の課題

図 7 より、CPU では距離計算が最も時間を要するが、GTX280, C1016 では学習時間が最も時間を要することがわかる。

本実装では近傍探索に幅優先探索を用いているため、同距離の全ニューロンを探索する前に、次の距離にあるニューロンの探索を行うことはできない。そのため、近傍全体を並列して学習することができないと考えられる。

そこで、学習を並列化する手法を提案する。図 4 に示すように、今回用いたデータ構造は正 20 面体の展開図を模している。今までの手法では、幅優先探索を導入することで、スリットの部分の近傍探索を円滑に行っていた。しかし、スリットの影響を受けない位置での学習は、近傍領域と勝者ニューロンからの距離を判別できる。マップ上において、幅優先探索が必要となる領域と必要とならない領域の日はを調べた所、約 1:2 であった。このことから、近傍探索を条件に加えることで実行時間の削減が可能であると考えられる。

次に、近傍探索と学習の分離する手法を提案する。近傍探索が配列のインデックスを処理対象としていることに対し、学習はニューロンの次元数だけのパラメータを処理対象としているため、学習の実行時間は近傍探索に比べ十分大きいと考えられる。そのため、近傍探索の手法が学習の並列化におけるボトルネックとなっていることが考えられる。そこで、先に近傍探索だけを行い、マップに勝者ニューロンからの距離を記録することで、後から近傍の全ニューロンの学習を並列して行うことができると考えられる。

これらの手法の実装と評価は今後の課題である。

### 7.3 ニューロン処理の並列化

本実装ではマップに対する処理の並列化を行った。しかし、ニューロンの次元が大きな場合は、ニューロン処理の並列化も考慮に行うことで、さらなる並列化が行えると考えられる。今回の球面 SOM におけるニューロンとは、単精度浮動小数点によるベクトルである。そのため、容易に GPU における並列化が可能である。

この手法の実装と評価は今後の課題である。

## 8. ま と め

本研究報告では、GPU による球面 SOM の実装と評価を行った。CPU では Opteron 1210 HE, GPU では GeForce8400GS, GeForce GTX280, Tesla C1060 を用いて比較することで、実行時間の短縮に成功していることを示した。さらに、結果から分かった課題を明確にし、その解決法を提案した。球面 SOM の GPU 実装には課題が多く、これからの更なる速度向上が期待できる。

## 参 考 文 献

- 1) T. Kohonen 自己組織化マップ シュプリング・フェアラーク東京 (1996)
- 2) G. Carpenter, S. Grossberg **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)
- 3) H. Speckmann, P. Thole, W. Rosentiel, I. Aleksander, J. Taylor **Hardware Implementations of Kohonen's Self-Organizing Feature Map** *Artificial Neural Networks, 2, Vol. 2, pp. 1451-1454* (1992)
- 4) M. Parrmann, M. Franzmeier, H. Kalte, U. Witkowski, U. Ruckert **A Reconfigurable SOM Hardware Accelerator** *Proceedings of the 10th European Symposium on Artificial Neural Networks, ESANN, pp. 337-342* (2002)
- 5) H. Tamukoh, T. Aso, K. Horio, T. Yamakawa **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)
- 6) R. D. Prabhu **SOMGPU: An Unsupervised Pattern Classifier in Graohical Processing Unit** *IEEE Congress on Evolutionary Computation, pp. 1011-1018* (2008)
- 7) 高塚 正浩, Ying Xin WU 球面 SOM のデータ構造と量子化誤差の考察 およびインタラクティブティの向上 知能と情報 (日本知能情報ファジィ学会誌) *Vol. 19, No 6, pp. 611-617* (2007)
- 8) 徳高 平蔵, 岸田 悟, 藤村 喜久郎 自己組織化マップの応用 多次元情報の 2 次元可視化 海文堂 (1999)
- 9) 設楽明宏, 西川由理, 吉見真聡, 天野英晴 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 *IPSI SIG Notes pp.31-36* (2009)