修士論文

GPGPUによる変分混合ガウスモデルのパラメータ推定高速化

西本　宏樹
奈良先端科学技術大学院大学
先端科学技術研究科
情報理工学プログラム

主指導教員: 中島 康彦 教授
コンピューティング・アーキテクチャ研究室（情報科学領域）

令和2年3月11日提出

本論文は奈良先端科学技術大学院大学先端科学技術研究科に
修士 (工学) 授与の要件として提出した修士論文である。

西本　宏樹

審査委員：

中島 康彦 教授	（主指導教員，情報科学領域）
池田 和司 教授	（副指導教員，情報科学領域）
中田 尚 准教授	（副指導教員，情報科学領域）
張 任遠 助教	（副指導教員，情報科学領域）
Tran Thi Hong 助教	（副指導教員，情報科学領域）

GPGPUによる変分混合ガウスモデルのパラメータ推定高速化*

西本　宏樹

内容梗概

近年，コンピュータのデータ処理速度や，インターネットの通信速度の上昇により世の中に生じる様々で膨大な事象がデータとして保存が可能になった．この大量のデータはビッグデータと呼ばれており，それらを分析し，ビジネスや研究に用いることが近年のトレンドとなっている．代表的なビッグデータの分析手法の1つとして，様々な性質や特徴が混ざりあって存在しているなかから，類似性によりグループに分類しその属性を分析する，クラスタリングと呼ばれる手法がある．クラスタリングは，ユーザーのセグメント分析やブランドポジションの認知等の手段として着目されている．クラスタリングを実現する手法は様々であるが，データをいくつかのガウス分布によって分類する混合ガウス分布を用いた手法は精度が高く，更にその混合ガウス分布にベイズ推論の概念を利用し，そのガウス分布のパラメータを共役確率分布から生成される確率変数とした，変分推論法に用いたパラメータ推定で得られるガウス分布によるクラスタリングは過学習を行い難く，クラスタ数を自動決定できるという特徴がある．しかし，混合ガウス分布に関する変分推論法は収束までに時間がかかることが知られおり，データ数に比例しその計算時間は膨大なものとなる．そこで変分混合ガウス分布の大規模データの適用の第一歩として，並列演算アーキテクチャであるGPUを用いてガウス分布に関する変分推論法の並列実装及び，評価を行った．提案手法は，同アルゴリズムをCPUで実装したものと比較して同じ精度を保ちながら，データ数が16777216=2^{24}のとき約524倍，クラスタ数が256のとき約220倍それぞれ高速化できた．また，従来のEMアルゴリズムによる混合ガウスモデルのパラメータ推定(GPGPUを使用)と比較して，クラスタ数が256の際，約70倍高速化できた．

*奈良先端科学技術大学院大学 先端科学技術研究科 修士論文, 令和2年3月11日.

キーワード

GPGPU, クラスタリング, 混合ガウスモデル, 変分推論, 高速化

GPGPU Implementation of Variational Bayesian Gaussian Mixture Models*

Hiroki Nishimoto

Abstract

In recent years, the processing speed of computers and the speed of communication speed have been increased. Various enormous events that occur in the world can be saved as data. This large amount of data is called "Big Data", It is a recent trend to analyze them and use them for business and research. As one of the typical "Big Data" analysis methods, clustering can classify data into groups based on similarity and analyzes their attributes. There are various methods to realize clustering. The method using Gaussian mixture distribution, which classifies data by several Gaussian distributions, has high accuracy. Furthermore, using the concept of Bayesian inference for the Gaussian mixture distribution, the parameters of the Gaussian distribution are random variables generated from the prior distribution. Clustering with Gaussian distribution based on the parameters estimated by variational inference can eliminate overlaerning, and can determine the number of clusters automatically. However, it is known that this variational inference method for Gaussian mixture requires enormous time to the finish convergence. The calculation time is proportional to the number of data. As the first step in applying large-scale data of this variational Gaussian mixture, the variational inference method for Gaussian distribution was implemeented in parall and evaluated. The proposed method maintains the same accuracy as compared to the algorithm implemented by CPU, but it has about 524 times when the number of data is $16777216 = 2^{24}$, When the number of clusters was 256, the speed was increased by about 220 times. Also, compared with the parameter

*Master's Thesis, Graduate School of Science and Technology, Nara Institute of Science and Technology, March 11, 2020.

estimation of the Gaussian mixture model using the conventional EM algorithm (using GPGPU), When the number of clusters was 256, the speed was increased about 70 times.

Keywords:

GPGPU, Clustering, Gaussian Mixture Models, Variational Inference, Acceleration

Contents

List of Figures

List of Tables

1. はじめに

1.1 背景

近年，コンピュータのデータ処理速度や，インターネットの通信速度の上昇により，世の中に生じる様々で膨大な事象がデータとして保存が可能になった．この大量のデータはビッグデータと呼ばれており，それらを分析し，ビジネスや研究に用いることが近年のトレンドとなっている [1]．代表的なビッグデータの分析手法の 1 つとして，様々な性質や特徴が混ざりあって存在しているなかから，類似性によりグループに分類しその属性を分析する，クラスタリングと呼ばれる手法がある [2]．このクラスタリングは，ユーザーのセグメント分析やブランドポジションの認知等の手段として着目されている [3][4][5]．クラスタリングを実現する手法は様々あり，代表的なものに，最短距離法 [6] や最長距離法 [7] 等を用いる階層的手法 [8] と呼ばれる手法と，K-means アルゴリズム [9] や混合ガウスモデルを用いる手法 [10] 等の非階層的手法と呼ばれる手法があり，計算量オーダやメモリ使用量などの問題からデータ数の大きな対象の解析には，非階層的手法を用いたクラスタリングが適正である [11]．非階層的手法を用いたクラスタリングの中でも，データを混合ガウスモデルから生成されたものとし，そのパラメータを求めることで実現する手法は，パラメータの重心を算出し，そこからの距離を用いて分類する K-means アルゴリズムを用いたよりも柔軟性が高い．混合ガウスモデルのパラメータの最尤値を推定する方法として，EM アルゴリズムと呼ばれる手法 [12][13] が用いられる．しかし，この EM アルゴリズムによる最尤推定には，特定要素への過学習やデータの過度な分解など，いくつかの重大な限界が存在する．その限界を解決する方法として，EM アルゴリズムにベイジアンアプローチを導入した，変分推論アルゴリズムを用いる手法 [15] が存在する．混合ガウスモデルに関する変分推論アルゴリズムは，混合ガウスモデルのパラメータを共役確率分布から生成される確率変数としており，K-means アルゴリズムや混合ガウスモデルに関する EM アルゴリズムに比べ，過学習をし難いという特徴があり，更にハイパーパラメータであるクラスタ数を最適化する [14]．このアルゴリズムは複雑で多様な特徴を持つビッグデータの分析には非常に有用だと考えられる．

1.2 目的

一般的に EM アルゴリズムを用いた混合ガウスモデルのパラメータ推定で実現されるクラスタリングは，K-means アルゴリズムを用いたものよりも収束するまでに必要な繰り返し回数や，一度の計算量が大きく，変分推論法を用いた混合ガウスモデルのパラメータ推定は，EM アルゴリズムを用いた混合ガウスモデルのパラメータ推定よりも収束に必要な繰り返し回数が更に大きい．そのため，機械学習ライブラリである scikit-learn[16] においても，教師なしのクラスタリングにおいて，クラスタ数が既知であり，データ数が 10000 を超える場合は MiniBatch KMeans の使用が推奨されており，クラスタ数が未知であり，データ数が 10000 を超える場合の推奨アルゴリズムは存在していない [17]．そこで，クラスタ数未知の大規模データの教師なしクラスタリングを実用化するため，並列演算アーキテクチャである GPU を用いて，変分混合ガウス分布のパラメータ推定の高速化モデルの実装・評価を行った．

1.3 構成

本章では本研究の背景と目的を述べた．次の 2 章において GMM の概要，GPU の概要及び関連研究を紹介を行い，3 章で提案手法の実装を述べ，4 章では他の実装との比較評価とその結果を述べ，5 章を本稿のまとめとする．

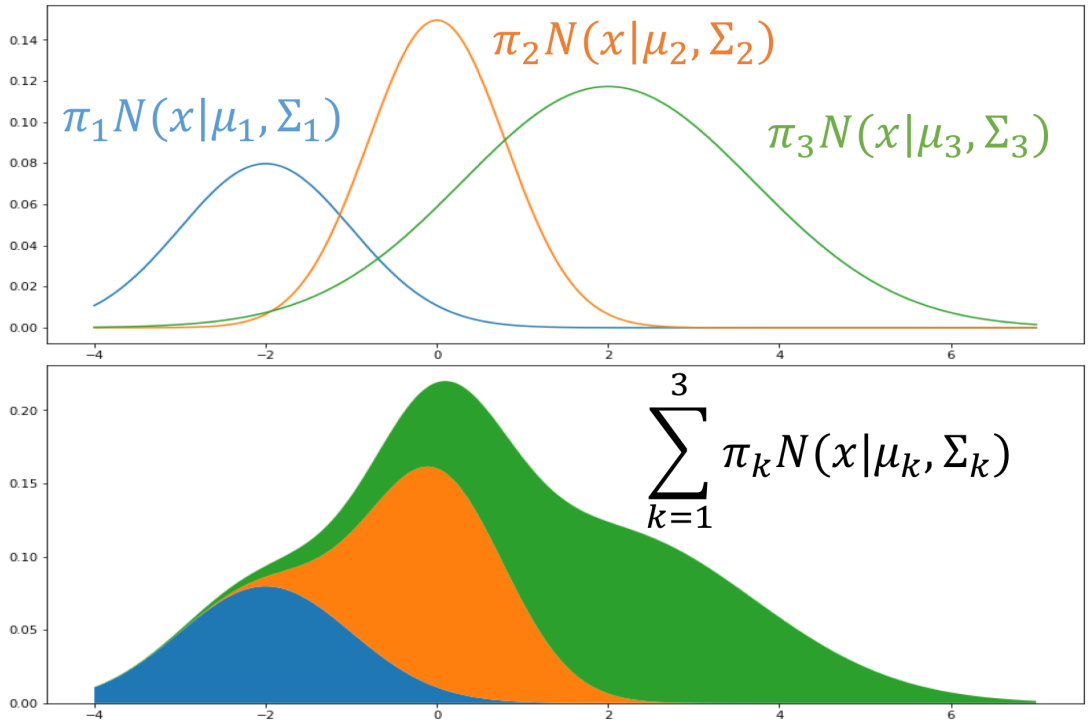

Figure 1　GMM の例

2. 関連研究

2.1 混合ガウスモデル

混合ガウスモデル (以下，GMM) は式 (1) で表される確率モデルであり，混合数 K 個のガウス分布にそれぞれの混合係数 π_k を掛けて足し合わせたものとして表される．

$$p(x|\pi, \mu, \Sigma) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k) \tag{1}$$

混合数 3，次元数 1 の GMM は図 1 のように表すことができる．

この GMM は，データ分析の為のクラスタリング以外にも，様々な応用に用いられており，Reynold ら [18] は，音声データより話者の声をモデル化するのに GMM を用い話者照合システムを構築し，NIST 話者認識評価で成功を納めてい

る．また，Toda ら [19] は，GMM を用いて声質変換フレームワークを構築している．Stauffer ら [20] は，GMM を用いてビデオストリームから背景を抽出するために，各ピクセルが背景に属するかを，属する・属さないの 2 値ではなく，確率的に求めるシステムを設計した．また，Fujita ら [21] は，無線 LAN を用いた位置推定において，GMM を用いて無線位置情報のモデル化を行い，従来手法の 5%程度にデータ量を削減し，データベースを軽量化した．

2.2 変分推論法による GMM のパラメータ推定

GMM を前述した応用に用いるためには，データに対し，GMM のパラメータである，各ガウス分布の平均 μ，分散 Σ，そして各ガウス分布に掛かる係数である π をデータ適合するように求めなければならない．しかし，最適な GMM のパラメータはデータから直接計算することが難しく，反復的な計算により尤度を最大化するアルゴリズムや，サンプリングアルゴリズムを用いて計算するのが一般的である．その最尤解を求める反復的な計算により尤度を最大化するアルゴリズムの 1 つが，1.1 節で述べた変分推論アルゴリズムを用いる手法 [15] である．このモデルは EM アルゴリズムによる混合ガウス分布の尤度関数の最大化 [13] にベイジアンアプローチを導入したものであり，混合係数 π にディリクレ分布を，平均 μ にガウス分布を，共分散行列の逆行列である精度行列 Λ にウィシャート分布を，それぞれ式 (2), 式 (3), 式 (4) のように事前分布として導入する．

$$\pi \sim Dir(\alpha) \tag{2}$$

$$\mu \sim \mathcal{N}(m, (\beta\Lambda)^{-1}) \tag{3}$$

$$\Lambda \sim \mathcal{W}(W, \nu) \tag{4}$$

このモデルのグラフィカルモデルを図 2 に示す．

この事前分布を導入することで，次の変分 EM アルゴリズムで呼ばれる計算過程で GMM のパラメータをデータに適応させることができる．

1. 初期化

   平均 μ_k，分散 Σ_k，そして混合係数 π_k を初期化し，ハイパーパラメータ $\alpha_0, \beta_0, W_0, \nu_0, m_0$ を用いて α, β, W, ν, m を初期化する．

2. 変分 E ステップ

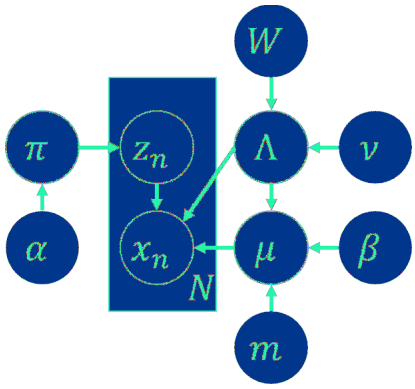

Figure 2 VBGMM のグラフィカルモデル

π, Λ を計算する

$$\ln \tilde{\pi} \equiv \mathbb{E}[\ln \pi_k] = \psi(\alpha_k) - \psi(\sum_{i=1}^{K}(\alpha_i)) \tag{5}$$

$$\begin{aligned} \ln \tilde{\Lambda_k} &\equiv \mathbb{E}[\ln |\Lambda_k|] \\ &= \sum_{d=1}^{D} \psi\left(\frac{\nu_k + 1 - d}{2}\right) + D \ln 2 + \ln |W_k| \end{aligned} \tag{6}$$

現在のパラメータ値を用いて，式 (7) で表される重み付き確率密度 ρ を計算し，式 (8) 負担率 r を計算する．

$$\rho_{nk} = \pi_k |\Lambda_k|^{1/2} \exp\left\{-\frac{D}{2\beta_k} - \frac{\nu_k}{2}(x_n - \mu_k)^T W_k (x_k - \mu_k)\right\} \tag{7}$$

$$r_{nk} = \frac{\rho_{nk}}{\sum_{j=1}^{K} \rho_{nj}} \tag{8}$$

3. 変分 M ステップ

計算されたパラメータを用いてそれぞれのパラメータを計算する．

$$N_k = \sum_{n=1}^{N} r_{nk} \tag{9}$$

$$\bar{x}_k = \frac{1}{N_k}\sum_{n=1}^{N} r_{nk}x_n \tag{10}$$

$$S_k = \frac{1}{N_k} r_{nk}(x_n - \bar{x}_k)(x_n - \bar{x}_k)^T \tag{11}$$

$$\alpha_k = \alpha_0 + N_k \tag{12}$$

$$\beta_k = \beta_0 + N_k \tag{13}$$

$$\nu_k = \nu_0 + N_k \tag{14}$$

$$m_k = \frac{1}{\beta_k}(\beta_0 m_0 + N_k \bar{x}_k) \tag{15}$$

$$W_k^{-1} = W_0^{-1} + N_k S_k + \frac{\beta_0 N_k}{\beta_0 + N_k}(\bar{x}_k - m_0)(\bar{x}_k - m_0)^T \tag{16}$$

4. 収束判定

   式 (17) に示す，変分下限 $\mathcal{L}_{new}$ を計算し，前回の繰り返しで計算した変分下限 $\mathcal{L}_{old}$ との差分が設定値を下回っていれば終了する．そうでなければ，$\mathcal{L}_{old}$ に $\mathcal{L}_{new}$ を代入し，E ステップに戻る．

$$\begin{aligned}
\mathcal{L} = \; & - \sum_{n=1}^{N}\sum_{k=1}^{K}(e^{r_{nk}} \times r_{nk}) \\
& + \sum_{k=1}^{K}\left(\nu_k|W_k| + \frac{\nu_k D \ln 2}{2} - \sum_{k=1}^{K}\psi(\nu_k)\right) \\
& + \left(\psi(\sum_{k=1}^{K}\alpha_k) - \sum_{k=1}^{K}\psi(\alpha_k)\right) \\
& - \frac{D\sum_{k=1}^{K}\beta_k}{2}
\end{aligned} \tag{17}$$

以上の処理により，対象データに適合したパラメータを得ることができる．このアルゴリズムで得られる GMM は，いくつのガウス分布から GMM が構成されるかを示すクラスタ数 K を最適化しており，EM アルゴリズムによる最尤推定で得られる GMM と異なり，データを過度に分解せずに適合する．図 3 は，クラスタ数 K が 3，次元数 2 の GMM からサンプリングしたデータを，ハイパーパラメータであるクラスタ数 K に 5 を与えた，EM アルゴリズムによる最尤推定で適合した GMM によりクラスタリングの結果であり，また，図 4 は，同じデータを

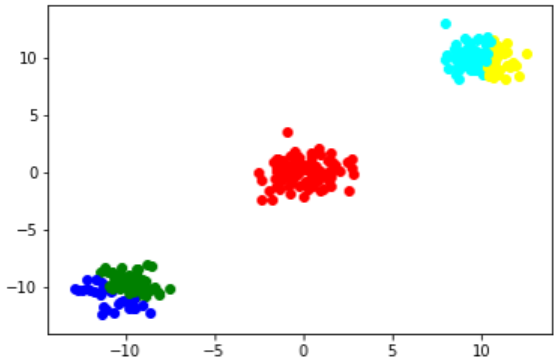

Figure 3　EM アルゴリズムを用いた GMM によるクラスタリング結果

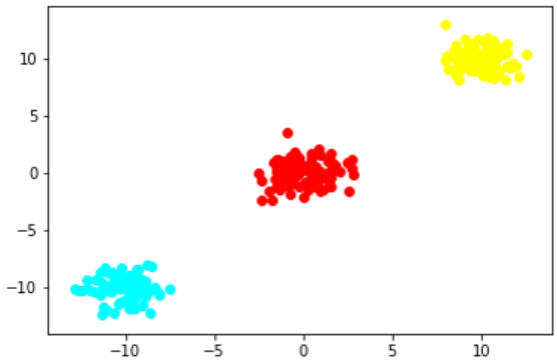

Figure 4　変分推論法を用いた GMM によるクラスタリング結果

同様にハイパーパラメータであるクラスタ数 K として 5 を与えた変分 EM アルゴリズムで適合した GMM によるクラスタリングの結果である．図 3 の EM アルゴリズムによるパラメータ推定で得られた GMM のクラスタリングでは，データを過度に分解してしまっているのに対し，図 4 の変分 EM アルゴリズムによるパラメータ推定で得られた GMM のクラスタリングでは，データを真のクラスタ数である 3 個に分割できていることが分かる．

2.3　GPGPU

この節では，GPGPU と並列処理の基本概念について述べる．GPU はグラフィックボードなどに搭載され，3D グラフィックスの生成・表示に必要な計算処理を行うプロセッサである．同一の構造を持つ多数の要素プロセッサを内蔵しており，そ

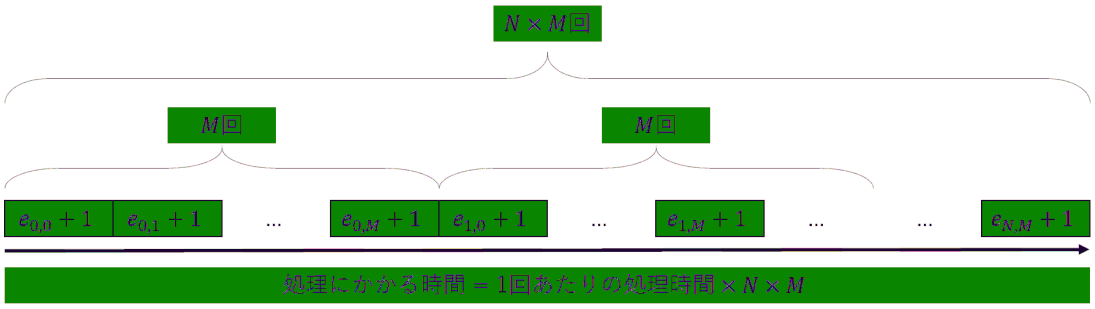

Figure 5 逐次処理での実行イメージ

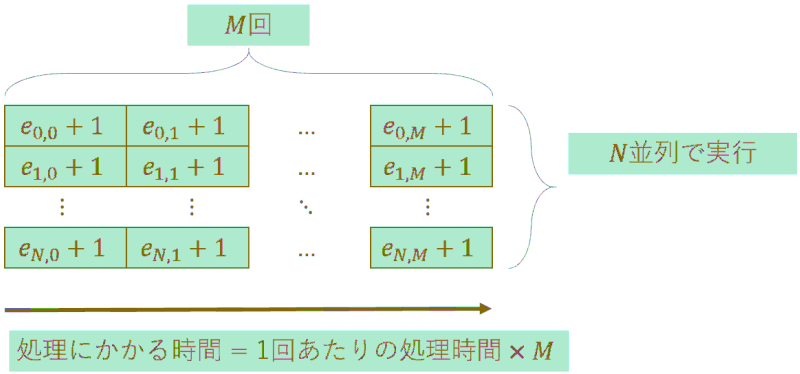

Figure 6 並列処理での実行イメージ

れらの要素プロセッサに処理を分散させ，演算を並列に行うことによって，高速化を実現している．GPGPU(General Purpose computing on GPU) は，GPU の並列処理能力の高さをグラフィックス処理以外に活用する技術及び専用デバイスのことであり，CPU よりも並列演算性能に優れ，専用設計のスーパーコンピュータと比較して流用性が高く，運用コストも低いため，HPC の分野で注目されている [22].

2.3.1 GPU と CPU

CPU の行う逐次処理と GPU の行う並列処理の違いを説明するために，$N \times M$ の行列の各要素 $e_{i,j}\{i = 1...N, j = 1...M\}$ に 1 を加算する処理を考える．このとき，CPU の処理は図 5，GPU の処理は図 6 のように表すことができる．

このとき，転送時間やパイプライン処理を考慮しない場合，1 回あたりの実行時間が同じ場合では，並列処理の方が N 倍高速に実行できることが分かる．この

ような各演算が独立に試行できるとき，並列処理による高速化を見込むことができる．

2.3.2 GPU の構造

この項では，NVIDIA 社の GPU ハードウエアの基本的な設計概念 (アーキテクチャ) について説明する．NVIDIA 社の CUDA(後述) に対応した GPU のアーキテクチャは，概ね共通しており，図 7 に示すようなアーキテクチャとなっている．GPU は多数の計算用コアとメモリ，レジスタなどの周辺回路を搭載したプロセッサである，ストリーミングマルチプロセッサ:SM がブロック上に並んだ構造の GPU チップと，SM を制御するスレッド実行マネージャー，データを格納するビデオカードで構成されている．ストリーミングマルチプロセッサ:SM の数は機種によって異なり，今回の研究に用いた GPU である NVIDIA Tesla V100 は SM の数が 80 基である．前述の通り，この SM ごとに計算用のコアが搭載されており，Tesla V100 を例にすると，単精度計算用のコアである FP32 が 64 個，整数計算用のコアである INT32 が 64 個等が搭載されている．また，SM 内にはそれぞれシェアードメモリとレジスタが搭載されており，これらのメモリは容量が小さいがアクセスが高速である．この多数のコアと高速なメモリを用いて並列演算を行う．ビデオカードは DRAM(Dynamic Random Access Memory) で実装されており，シェアードメモリなどに比べるとアクセス速度は遅いが，容量が大きい．図 7 に示した赤い矢印はデータの流れを示しており，GPU を用いた並列演算においては，ホストデバイスのメモリ上からデータをビデオカードに格納し，ビデオカードから GPU チップ上に読み込み，演算後，GPU チップからビデオカードへ格納，ビデオカードからホストデバイスのメモリに読み出し，という手順で行われるのが一般的である．黒い矢印は命令の流れを示す．

2.3.3 汎用並列コンピューティングプラットフォーム:CUDA

この項では本研究にも用いた NVIDIA の開発・提供する GPU 向け汎用並列コンピューティングプラットフォーム（並列コンピューティングアーキテクチャ），およびプログラミングモデルである，CUDA について説明する．並列演算においては，その演算処理を並列化するための構造を設定する必要があり，その最小単位をスレッドと呼び，そのスレッドを 3 次元配列の構造にまとめたものをブロック，そのブロックをまとめたものをグリッドと呼ぶ．実行時，各スレッドはブロックごとのスレッドの ID である threadId 及び，そのスレッドが属するブロックの

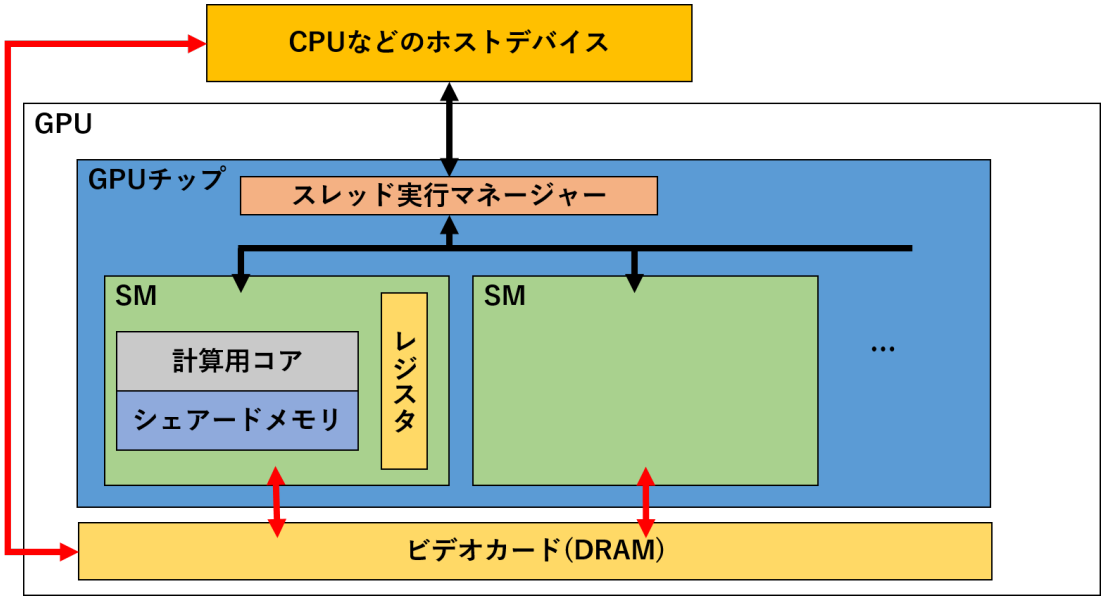

Figure 7 GPU の設計概念

ID である blockId，ブロックの次元数である blockDim を取得でき，それらの情報から，計算対象配列の一意なインデックスをスレッドごとに取得し，SIMD 命令で演算するのが一般的な CUDA による並列計算である．2.3.1 項で例に挙げた行列への足し算を考えるとき，行ごとに並列計算のために $\{1...N\}$ の一意なインデックス i を取得する．このとき，$N/32$ ブロック，32 スレッドで実行した場合，i =blockId $\times$ blockDim + threadId +1 で，表 1 に示すように一意にインデックスを取得できる．

2.4 アーキテクチャによる GMM のパラメータ推定の高速化に関する関連研究

この節ではアーキテクチャの GMM のパラメータ推定の高速化についての関連研究を紹介する．Nvidia の Kumar[23] らは GPU を用いた GMM のための EM アルゴリズムモデルを提案した．また，Guo ら [24] が，CPU での実行を高速化するため EM アルゴリズムをパイプライン処理に適したものにした．また，He ら [25] は，Guo らのパイプライン処理を更に効率化し FPGA 実装を行った．しかし，いずれも EM アルゴリズムによる最尤推定の実装であり，特定のデータへの過学習やデータの過度な分解などの問題を抱えている．そこで，変分推論による GMM

Table 1 インデックス取得の例

取得したいインデックス i	blockId	blockDim	threadId
1	0	32	0
2	0	32	1
⋮	⋮	⋮	⋮
32	0	32	31
33	1	32	0
⋮	⋮	⋮	⋮
N	(N-1)/32	32	31

のパラメータ推定の高速化を行った.

Table 2 計算に使用する配列の概要

配列名	メモリ使用量	概要
GAUSS	$N \times K$	ガウス対数確率密度を格納する配列．実装上は RESP と共有
LAMBDA	$K \times D$	4 に示した値を格納する配列
WEIGHT	$K \times D$	2 に示した，クラスタごとの重みの値を格納する配列
WLP	$N \times K$	7 に示した，重み付き対数確率を格納する配列， 実装上は GAUSS と共有
RESP	$N \times K$	8 に示した，負担率を格納する配列． 実装上は GAUSS と共有
ALPHA	K	12 に示した，重みの事前分布をパラメータを格納する配列．
BETA	K	13 に示した，平均の事前分布のパラメータを格納する配列．
NU	K	14 に示した，精度の事前分布のパラメータを格納する配列．
MEAN	$K \times D$	15 に示した，平均の事前分布のパラメータを格納する配列．
PRE	$K \times D$	16 に示した，精度の事前分布のパラメータを格納する配列．
LDC	K	W の行列式を格納した配列．
X	$N \times D$	入力データ．
SquareX	$N \times D$	入力データのパラメータの 2 乗の値を格納する配列．
arrA	$N \times K$	一時的な値を格納する配列．
arrB	$K \times D$	一時的な値を格納する配列．
arrC	K	一時的な値を格納する配列．

3. 提案手法とその実装

この章では，変分推論法による GMM のパラメータ推定の具体的な並列実装方法について説明する．本実装では He[25]，Kumar[23] ら同様に行列計算の計算量を削減するため，共分散行列ではなく，共分散行列の対角成分，つまり分散のみを扱っている．ここで対象データのデータ数，クラスタ数，次元数をそれぞれ N,K,D とする．尚，文中に現れる T_1 はデータ数，T_2 は混合数よって決まる値である．また，文中に述べるオーダは計算量のオーダである．表 2 に，計算に使用する配列の概要を示す．

以下に，主要なカーネルの実装を記す．

3.1 ガウス対数確率密度関数計算関数:GAUSS

このカーネルでは，ガウス対数確率密度関数の計算を行う．式 (18) により，配列 GAUSS の $n = \{1...N\}, k = \{1...K\}$ までの各要素が計算される．

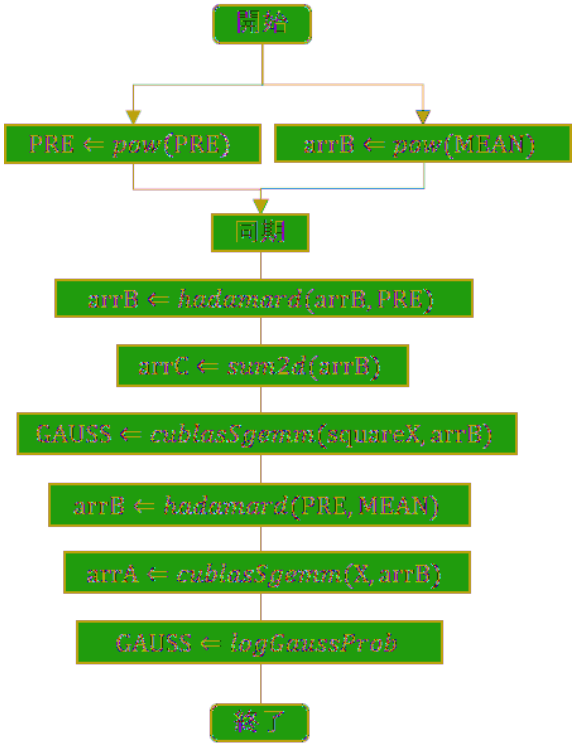

Figure 8 GAUSS の処理の流れ

$$
\begin{aligned}
\mathrm{GAUSS}_{(n,k)} &= -\frac{D}{2\beta_k} - \frac{\nu_k}{2}(x_n - \mu_k)^T W_k (x_k - \mu_k) \\
&= -\frac{D}{2\beta_k} - \frac{\nu_k}{2}\left(\sum_{d=1}^{D}(m \circ m \circ W)_{(k,d)}\right. \\
&\qquad \left. -X(m \circ W)^T + X \circ XW^T\right) \qquad (18)
\end{aligned}
$$

実際の計算手順は図 8 に示すフローチャートのようになる．

図 8 内の *pow* は入力配列の要素の 2 乗を出力するカーネル，*hadamard* は行列同士の要素の掛け算であり，どちらのカーネルも行もしくは列の数並列化されている．また，行列積の計算には CUDA の並列数値計算ライブラリ cuBLAS の行列積関数 cublasSgemm を用いた．*logGaussProb* カーネルは Algorithm1 で計算される．Algorithm1 は，N/T_1 ブロック，T_1 スレッド及び各ブロック T_1 個のシェ

Algorithm 1 logGaussProb カーネル

$n \Leftarrow$ blockId * blockDim + threadId
$tid \Leftarrow$ threadId

(a)

if $tid < K$ **then**
  $share_{tid} \Leftarrow \mathrm{LDC}_{tid} - 0.5 \times D \times \mathrm{NU}_{tid}$
end if
スレッド間同期

(b)

for $k \Leftarrow 0$ to K **do**
  $\mathrm{GAUSS}_{n,k} \Leftarrow -0.5 \times (D \times \log(2 \times \pi) + (\ \mathrm{GAUSS}_{n,k} - (2 \times \mathrm{arrA}_{n,k} + \mathrm{arrC}_k)) + share_k$
end for

(c)

アードメモリ $share$ で実行され，$O(NK)$ の処理を，並列演算により，各スレッド辺り $O(K)$ で演算する．

このアルゴリズムは初めに (a) において n にブロック数 × スレッド数分独立なインデックス，つまり 1 から N までの独立なインデックスを取得し，tid にブロックごとに独立なインデックス 1 から T_1 を取得する．次に，(b) において，tid が K 未満，つまり K 以下の threadId を割り当てられたスレッドにおいて，シェアードメモリに次の計算で用いる値を格納している．この処理により次の処理でグローバルメモリへのアクセスを削減し，高速化している．最後に，(c) で，図 8 に示した配列 arrA 及び arrB，前述したシェアードメモリの値を用いて，配列 GAUSS の各要素を計算する．これらの計算によって，配列 GAUSS にガウス関数の対数確率密度を得る．

3.2 λ計算カーネル:LAMBDA

このカーネルでは，各要素が式 (6) で表される，配列 LAMBDA の計算する．計算には Algorithm2 で示されるカーネルを K/T_2 ブロック，T_2 スレッド及び，サイズ T_2 のシェアードメモリ $share$ を用いて演算を行う．

Algorithm1 は，データ数 N 並列で計算していたのに対し，このカーネルではクラスタ数 K 並列で計算するために，初めに (a) において k にブロック数 × スレッド数分独立なインデックス，つまり 1 から K までの独立なインデックスを取得し，tid にブロックごとに独立なインデックス 1 から T_2 を取得する．次に，(b) においてシェアードメモリに $\nu_k + 1$ の値を格納し，スレッドを同期する．最後に，

Algorithm 2 LAMBDA カーネル

$k = \text{blockId} \times \text{blockDim} + \text{threadId}$
$tid = \text{threadId}$

(a)

if $tid < K$ **then**
  $share_{tid} \Leftarrow NU_{tid} + 1$
end if
スレッド間同期

(b)

$tmp = 0$
for $d = 0$ to D **do**
  $tmp \Leftarrow tmp + digamma(share_k - d)$
end for
$\text{LAMBDA}_k \Leftarrow 0.5 * (D \times \log(2.0) + tmp - D/\text{BETA}_k)$

(c)

(c) により配列 LAMBDA(4) の値を得る.

3.3 重み付き対数確率計算カーネル:WLP

このカーネルでは GPU 上でカーネル GAUSS とカーネル LAMBDA によって計算された配列と，式 (5) により，CPU 上で計算された配列を計算を用いて，式 (19) に従い，重み付き対数確率を計算し，配列 WLP に格納する.

$$\text{WLP}_{(n,k)} = \text{WEIGHT}_k + \frac{\text{LAMBDA}_k}{2} + \text{GAUSS}_{(n,k)} \tag{19}$$

計算には Algorithm3 で示されるカーネルを N/T_1 ブロック，T_1 スレッド，及び，サイズ T_1 のシェアードメモリを用いて演算を行う.

この処理も Algorithm1 同様にデータ数 N 並列で計算を行うために，(a) において n にブロック数 × スレッド数分独立なインデックス，つまり 1 から N までの独立なインデックスを取得し，tid にブロックごとに独立なインデックス 1 から T_1 を取得する．次に，(b) において，各ブロックごとに K 個のスレッドで，$\text{WEIGHT}_k + \frac{\text{LAMBDA}_k}{2}$ の値をそれぞれ計算し，シェアードメモリに格納し，最後に (c) で，各スレッド $O(K)$ でシェアードメモリと GAUSS の値を足し合せる，

Algorithm 3 WLP カーネル

n = blockId × blockDim + threadId
tid = threadId
(a)
if $tid < K$ **then**
  $share[tid]$ ⇐LAMBDA$_k$/2+WEIGHT$_k$
end if
スレッド間同期
(b)
for $k \Leftarrow 0$ to K **do**
  WLP$_{n,k}$ ⇐ GAUSS$_{n,k}$ + $share_k$
end for
(c)

これらの計算によって，配列 WLP に式 (7) に示した，重み付き対数確率密度を得る．

3.4 対数負担率計算カーネル:RESP

変分 M ステップで，パラメータを更新するのに用いる負担率を計算し，配列 RESP に格納する．RESP の各要素は式 (20) で計算される．

$$\mathrm{RESP}_k \quad = \quad \mathrm{WLP}_k - \sum_{i=1}^{K}(\mathrm{WLP}_i) \tag{20}$$

ここで $\sum_{i=1}^{K}(\mathrm{WLP}_i)$ の計算には，桁落ちを避けるため，式 (21) で示される，logsumexp と呼ばれる手法を用いた．

$$\log\left(\sum_{i=1}^{N}\exp(x_i)\right)$$

$$
\begin{aligned}
&= \log\left\{\exp(x_{max})\sum_{i=1}^{N}\exp(x_i - x_{max})\right\} \\
&= \log\left\{\sum_{i=1}^{N}\exp(x_i - x_{max}) + x_{max}\right\} \qquad (21)
\end{aligned}
$$

計算には次のアルゴリズム 4 で示されるカーネルを N/T_1 ブロック，T_1 スレッド及び，サイズ T_1 のシェアードメモリを用いて演算を行う．

Algorithm 4 RESP カーネル

n $\Leftarrow$ blockId $\times$ blockDim + threadId
tid $\Leftarrow$ threadId
(a)
$share_{tid} \Leftarrow \mathrm{WLP}_n$;
for $k \Leftarrow 2$ to K **do**
  $share_{tid} \Leftarrow max(\mathrm{WLP}_{n,k}, share_{tid})$
end for
スレッド間同期
(b)
$sum \Leftarrow 0$;
for $k = 1$ to K **do**
  $sum \Leftarrow \exp(WLP_{n,k} + share_k)$
end for
$val \Leftarrow \log(sum) + share_k$
for $k = 1$ to K **do**
  RESP $_{n,k} \Leftarrow$RESP$_{n,k} - val$
end for
(c)

この処理も Algorithm1 及び，Algorithm3 同様にデータ数 N 並列で計算を行うために，(a) において n にブロック数 $\times$ スレッド数分独立なインデックス，つまり 1 から N までの独立なインデックスを取得し，tid にブロックごとに独立なインデックス，つまり 1 から T_1 を取得する．次に (b) で，N 個のデータごとに，K 次元の期待値の最大値を $share_{tid}$ に格納，その値を用いて (c) で，式 (21) に示した logsumexp を用いて，配列 RESP に式 (8) に示した負担率 r を得る．

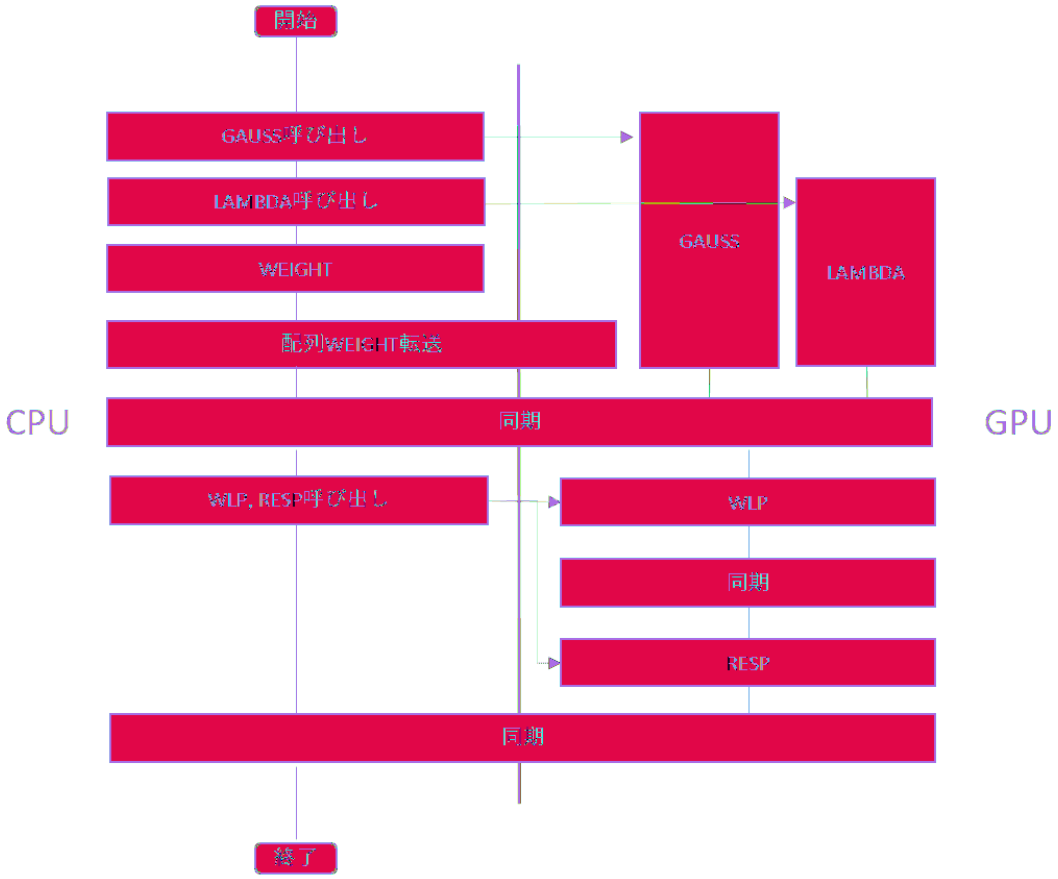

Figure 9　並列実装された変分 E ステップ

GPU 上で実行される GAUSS, LAMBDA, WLP, RESP の 4 つのカーネルと CPU 上で実行される WEIGHT カーネルで 2.2 節で述べた，変分 E ステップとなる．並列実装された変分 E ステップの処理の流れを図 9 に示す.

3.5　対数尤度の合計計算カーネル:SR

RESP の値を用いて変分 M ステップでパラメータを計算するに辺り，RESP を指数化しなければならないが，後に計算される変分下限の値の 1 つに指数化前の RESP の値を用いるものがあるので，その部分 (SR とする) を式 (22) に従い，先に計算する.

$$\mathrm{SR} = -\sum_{n=1}^{N}\sum_{k=1}^{K}(e^{RESP_{nk}} \times RESP_{nk}) \tag{22}$$

GPU を用いた加算において，スレッド間で書き込みアドレスが競合すると，正しい値が書き込めない．つまり，配列の要素の合計値を求める等の計算において，前述してきたような SIMD 的なアプローチを用いることはできない．競合を回避しつつ高速に演算するために，CPU 側で Algorithm5 に示す callSum カーネル，GPU 側で Harris の配列総和の算出の高速化手法 [26] を参考に実装した Algorithm6 に示す cpySum 及び foldSum カーネルを用いて実装を行っている．

foldSum,cpySum カーネルは，配列を入力とし，スレッド数に応じて部分ごとの合計を求めるカーネルである．例として要素数 128^2 の配列 arr を入力に 128 ブロック，128 スレッドで実行した場合，次の式に示す代入を行う．

$$arr_i \Leftarrow \sum_{j=i\times128\times(i-1)}^{i\times128\times(i-1)} arr_j \tag{23}$$

cpySum カーネルと foldSum カーネルの違いは代入配列であり，cpySum カーネルは求めた和を異なる格納に代入，foldSum カーネルは求めた和を入力した格納に代入する．SR カーネルにおいては，配列 RESP の値を次の変分 M ステップでも用いるため，cpySum カーネルで計算用の配列に加算・代入後，foldSum カーネルを繰り返し総和を求める．callSum カーネルは，前述した cpySum カーネルと fold カーネルを CPU 側から呼び出すカーネルであり，配列 $inputArr$ を入力とし，GPU 上で効率的に扱えるサイズごとに配列を切り分け，部分ごとの合計を cpySum，foldSum カーネルを用いて求め，足し合わせることで配列全体の合計値 $output$ を出力とするカーネルである． このカーネルはまず，(a) で配列先頭からどれだけの要素の合計を求めたかを示す変数 $offset$ の初期化，入力配列 $inputArr$ の大きさ $size$ の取得を行い，(b) で，$inputArr$ の大きさ $size$ に応じたブロック数，スレッド数で cpySum カーネルを呼び出し，足し合わせつつ，配列の合計を求めるための配列 $partArr$ に代入する．例として，$size$ が 256^3 未満 128^3 以上の場合，入力配列の $offset$ 番目から $offset + 128^3$ 個の要素の 128 個ずつの合計，つまり，$i = \{1...128^2\}$ において，

$$partArr_i \Leftarrow \sum_{j=offset+128\times(i-1)}^{offset+128\times(i-1)} inputArr_j \tag{24}$$

Algorithm 5 callSum カーネル

$size \Leftarrow$inputArr の長さ , $offset \Leftarrow 0$

(a)

while $size > 0$ **do**
  if $size >= 256^3$ **then**
    256^2 ブロック，256 スレッドで cpySum カーネルを実行
    $size \Leftarrow size - 256^3, offset \Leftarrow offset + 256^3, partSize \Leftarrow 256^2$
  else if $size >= 128^3$ **then**
    128^2 ブロック，128 スレッドで cpySum カーネルを実行
    $size \Leftarrow size - 128^3, offset \Leftarrow offset + 128^3, partSize \Leftarrow 128^2$
  else if $size >= 1024^2$ **then**
    1024 ブロック，1024 スレッドで cpySum カーネルを実行
    $size \Leftarrow size - 1024^2, offset \Leftarrow offset + 1024^2, partSize \Leftarrow 1024$
  else if $size >= 512^2$ **then**
    512 ブロック，512 スレッドで cpySum カーネルを実行
    $size \Leftarrow size - 512^2, offset \Leftarrow offset + 512^2, partSize \Leftarrow 512$
  else if $size >= 256^2$ **then**
    256 ブロック，256 スレッドで cpySum カーネルを実行
    $size \Leftarrow size - 256^2, offset \Leftarrow offset + 256^2, partSize \Leftarrow 256$
  else if $size >= 128^2$ **then**
    128 ブロック，128 スレッドで cpySum カーネルを実行
    $size \Leftarrow size - 128^2, offset \Leftarrow offset + 128^2, partSize \Leftarrow 128$
  else if $size >= 64^2$ **then**
    64 ブロック，64 スレッドで cpySum カーネルを実行
    $size \Leftarrow size - 64^2, offset \Leftarrow offset + 64^2, partSize \Leftarrow 64$
  else if $size >= 1024$ **then**
    1 ブロック 1024 スレッドで cpySum カーネルを実行
    $size \Leftarrow size - 1024, offset \Leftarrow offset + 1024, partSize \Leftarrow 1$
  else
    fromArray の offset から配列の最後の要素の合計値を求めるカーネルで値を取得し output に加算
  end if
end while

(b)

while $partSize > 1$ **do**
  foldSum カーネルを threadNum で実行
  $partSize \Leftarrow partSize/threadSize$
end while
output に $partArr$ の 0 番目の要素を加算するカーネルを実行

(c)

Algorithm 6 cpySum 及び foldSum カーネル

$i \Leftarrow$ blockId $\times$ blockDim^2 + threadId
$tid \Leftarrow$threadId
$mysum \Leftarrow$ inputArr$_i$ + inputArr$_{i+blockDim}$
$share_{tid} \Leftarrow mysum$
スレッド間同期
$idx \Leftarrow$ blockDim / 2
while $idx > 32$ **do**
  if $tid < idx$ **then**
    $share_{tid} \Leftarrow mysum \Leftarrow mysum + share_{tid+idx}$
    スレッド間同期
  end if
  idx を 1 ビット右にシフト
end while
if $tid < 32$ **then**
  if blockDim ¿= 64 **then**
    $mysum \Leftarrow share_{tid+32}$
  end if
  $offset \Leftarrow 32/2$
  while $offset > 0$ **do**
    $mysum \Leftarrow mysum+$ 同じワープ内の別のスレッドの $mysum$
  end while
end if
if $tid == 0$ **then**
  outputArr$_{blockId} \Leftarrow mysum$
end if

の代入が行われる．$partArr$ への代入後，(c) で，foldSum カーネルを用いて $partArr$ の合計を求める．先ほどの例の場合，2 回のループが行われ，$i = \{1...128\}$ 及び $i = 1$ の順に，

$$partArr_i \Leftarrow \sum_{j=128\times(i-1)}^{128\times(i-1)} partArr_j \tag{25}$$

と合計値の代入が行われ，$partArr_1$ に $inputArr$ の $offset$ 番目から $offset+128^3$ 番目までの要素の合計値が代入される．この合計値を output に加算，という手順を繰り返し，配列全体の合計値を求める．これらのカーネルを用いて演算を行うことで，$O(N)$ の処理を各スレッド $O(\log N)$ で演算している．

3.6 クラスタ毎の RESP の合計値計算カーネル:SEC

各クラスタ毎の RESP の合計値を格納する配列 SEC:$Sum\ of\ Each\ Cluster$ を式 (9) を用いて計算する．実装については，SR カーネルと同様に callSum, cpySum, foldSum カーネルを用いて実装を行い，各スレッド $O(\log N)$ で演算する．

3.7 クラスタ毎の平均計算カーネル:MEC

各クラスタの平均を格納する配列 MEC: $Mean\ of\ Each\ Cluster$ を式 (10) 計算する．この計算は式 (26) で置き換えることができ，

3.1 同様，行列積ライブラリ cublasSgemm を用いることができる．

$$\bar{x}_k = r^T X / N_k \tag{26}$$

3.8 クラスタ毎の分散計算カーネル:VEC

各クラスタの分散を格納する配列 VEC: $Variance\ of\ Each\ Cluster$ を式 (11) で計算する．この計算はこの GMM が共分散ではなく分散を用いている場合，式 (27) で置き換えることができる．

$$S_k = \frac{r \cdot X \circ X}{N_k} - 2\frac{\bar{x}_k \circ r \cdot X}{N_k} + \bar{x}_k \circ \bar{x}_k \tag{27}$$

このカーネルにおいても，3.1 同様，行列積は CUDA の数値計算ライブラリである，cuBLAS の cublassSgemm を用い，行列同士の足し算及び，行列同士の掛け算は T_1 スレッド，N/T_1 ブロックで計算する．

3.9 α, β, ν の計算

式 (12), (13) 及び (14) の計算コストは小さいため，CPU 上で計算を行う．

3.10 m の計算

式 (15) で示される，各クラスタの平均 m の事前確率を計算する．この計算も計算コストは小さいため，CPU 上で計算を行う．

3.11 精度行列計算カーネル:PC

精度行列の事前確率である W は，式 (16) で計算される．W の値は平方根を計算し，$K \times D$ の配列 PC に代入する．また，変分下限の計算のため，この配列を用いて各クラスタ毎の対数精度行列の行列式を計算する．このカーネルは K スレッドで実行される．

GPU 上で実行される SEC, MEC, CEC, PC の 4 つのカーネルと CPU 上で実行される ALPHA, BETA, NU, MEAN の 4 つカーネルで 2.2 節で述べた，変分 M ステップとなる．並列実装された変分 M ステップの処理の流れを図 10 に示す．

3.12 変分下限の計算

最後に式 (17) で示される変分下限を計算する．SR カーネルで計算した部分及び，PC カーネルで計算した対数精度行列の行列式の計算を除けば，計算コストは低いため，CPU 上で計算を行う．

以上のカーネルの実装により，変分推論法による GMM のパラメータ推定を並列実装した．

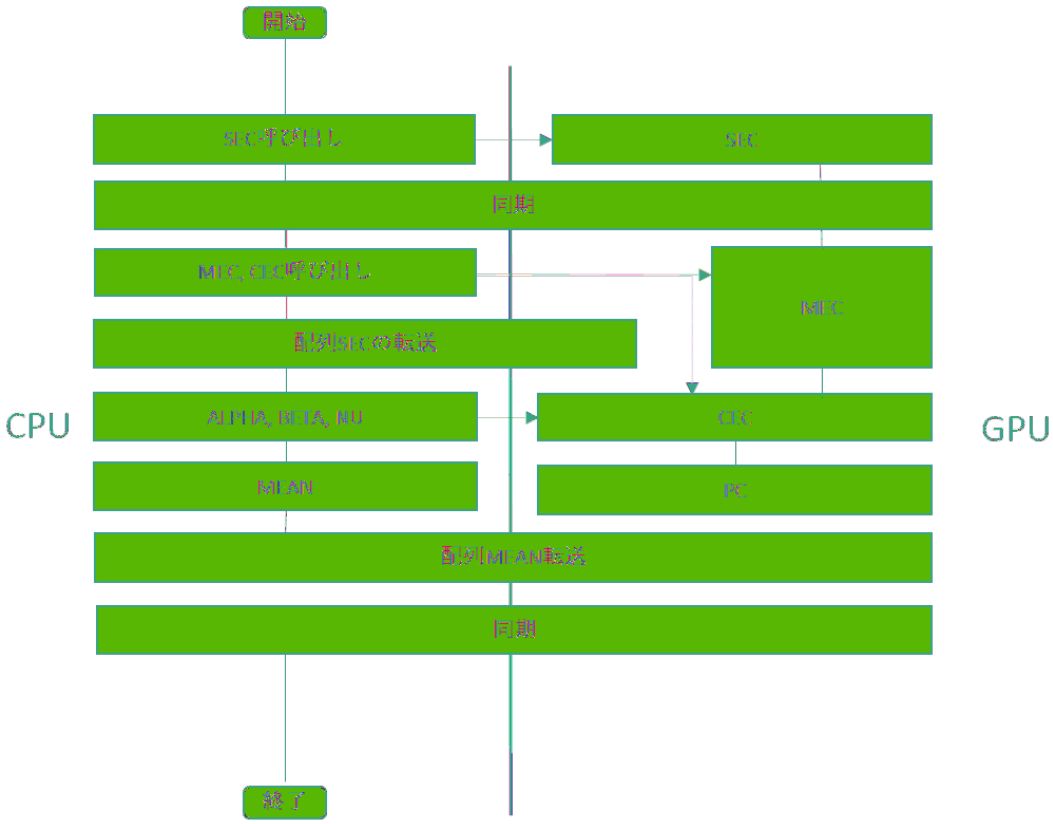

Figure 10 並列実装された変分 M ステップ

Table 3 評価環境

OS	CentOS Linux release 7.6.1810 (Core)
CPU	Intel(R) Xeon(R) Gold 6144 CPU @ 3.50GHz
GPGPU	Tesla V100 PCIe 16GB
memory	256GB
CUDA compiler	NVIDIA (R) Cuda compiler driver V10.0.130
scikit-learn	0.22.dev0

4. 評価

本実装の実行速度や精度の優位性を確かめるため，混合ガウスモデルの変分推論によるパラメータ推定モデルの CPU の実装として機械学習ライブラリ，scikit-learn の BayesianGaussianMixtureModels[16]，GPU を用いた混合ガウスモデルの最尤推定によるパラメータ推定モデルとして Corv の EM-GMM[27] の異なる 2 つの実装との比較を行った．実験には任意の個数のガウス分布からランダムにサンプリングし作成した人工データを用いた．実験環境を表 3 に示す．

また，混乱を避けるため本実装を VB-GPU，scikit-learn の変分推論によるパラメータ推定モデルを VB-CPU，EM アルゴリズムによるパラメータ推定モデルの GPGPU 実装を EM-GPU とする．

4.1 VB-CPU との比較

VB-CPU との比較評価として，次の 3 つの評価実験を行った．

- 次元数及びクラスタ数が等しく，データ数の異なる入力データを用いて実行速度と平均対数尤度を比較
- 次元数及びデータ数が等しく，クラスタ数の異なる入力データを用いて実行速度と平均対数尤度を比較
- 同じデータを用いて各カーネルの実行速度の比較

クラスタ数は対象データ作成に用いたガウス分布の個数の 2 倍とした．また，評価項目の平均対数尤度は式 (28) で計算される．

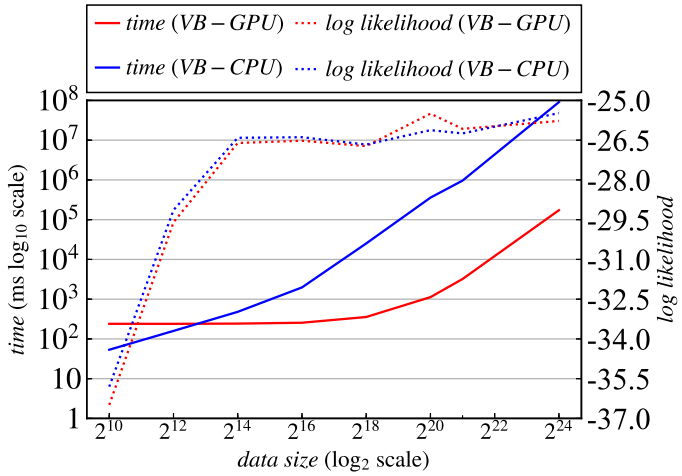

Figure 11 データ数の異なるデータでの VB-CPU との比較 (クラスタ数 16, 次元数 16)

$$\begin{aligned} &\textit{Average log likelihood} \\ &\qquad = \frac{\left\{-\sum_{i=1}^{N} \log\left\{\sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k)\right\}\right\}}{N} \end{aligned} \tag{28}$$

4.1.1 データ数の変更

次元数，対象データのクラスタ数を 16，データ数を 1024($= 2^{10}$) から 16777216($= 2^{24}$) のデータを用いて実行時間と平均対数尤度の比較を行った．この実験の評価結果を図 11 に示す．

図 11 より，データ数が大きい際，同程度の対数尤度を保ちながら VB-GPU が VB-CPU の実行速度を大きく上回っていることが分かる．データ数が 16777216$= 2^{24}$ のとき，VB-GPU 実行速度は VB-CPU の 524 倍で高速であった．

4.1.2 クラスタ数の変更

対象データの次元数を 16，データ数を 262144 に固定し，クラスタ数 8 から 256 のデータを用いて実行時間と平均対数尤度の比較を行った．この実験の評価結果

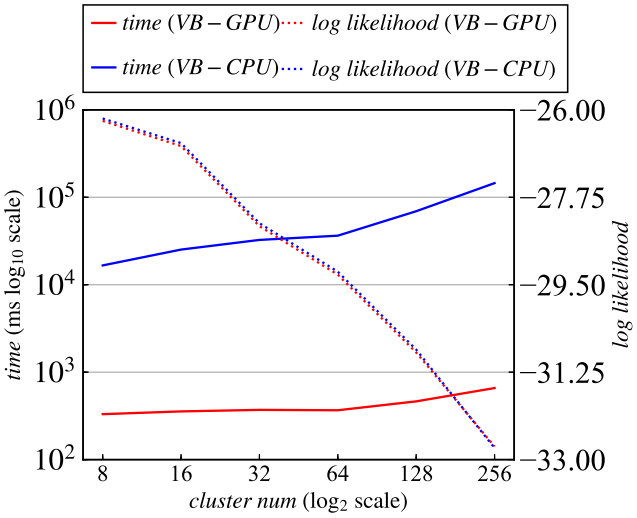

Figure 12　クラスタ数の異なるデータでの VB-CPU との比較 (データ数 2^{18}, 次元数 16)

を図 12 に示す. 図 12 より，クラスタ数に寄らず VB-GPU の実行速度が VB-CPU の実行速度より高速であることが分かる．クラスタ数が 256 のとき，VB-CPU の実行速度は VB-CPU の 220 倍高速であった．

4.1.3　各カーネルの実行時間

並列化の有効性を確認するため，3 で述べた主なカーネルの実行時間の比較を行った．VB-GPU の各カーネル実行時間の計測には GPGPU 向けプロファイラである NVIDIA Visual Profiler[28] 及び，C++標準ライブラリである Chrono ライブラリを用いた．VB-CPU の各カーネル実行時間の計測には Python 標準のプロファイラである cProfile 及び time 関数を用いた．評価に用いたデータはクラスタ数 16，次元数 16，データ数 1048576(= 2^{20}) のものを用いた．この実験の評価結果を，表 4 に示す．

また，ボトルネックの変化を調べるために各カーネルの処理時間の割り合いを比較した．図 13 に，実行時間が処理の全体実行時間の 1%以上を占める処理のそれぞれ割合の比較を示す．

表 4 より，特にボトルネックとなっていたカーネルの実行時間が並列処理により，高速されていることが分かり，並列処理が有効であったことが分かる．図 13 より，ボトルネックが VB-CPU における最大のボトルネックが RESP カーネル

Table 4 各カーネルの処理時間の比較

kernel	VB-CPU(ms)	VB-GPU(ms)	times
GAUSS	105986.234	399.990	×264.972
LAMBDA	21.284	2.805	×7.588
WEIGHT	13.134	1.535	×8.556
WLP	27494.603	76.623	×358.830
RESP	111091.948	146.225	×759.733
SR	75830.541	440.959	×171.967
EXP	66386.782	188.97	×351.309
SEC	3740.972	199.909	×18.713
MEC	15053.789	136.491	×110.291
VEC	36325.775	268.127	×135.480
MEAN	1.732	0.71	×2.439
PC	3.606	6.98	×0.517
others	570.263	122.758	×4.645

から，SR カーネルとなっていることが分かる．これは，他のカーネルと異なり，SR カーネルが桁落ち防止のため，倍精度を用いた実装を行っており，GPU が倍精度小数点による演算が浮動小数による演算に比べ低速であることが原因であると考えられる．

4.2 EM-GPU との比較

EM-GPU との比較評価として，次の 2 つの評価実験を行った．

- 次元数及びクラスタ数が等しく，データ数の異なる入力データを用いて実行速度と平均対数尤度を比較
- 次元数及びデータ数が等しく，クラスタ数の異なる入力データを用いて実行速度と平均対数尤度を比較

また，EM-GPU の実装では，1 度の試行で最適なクラスタ数を見つけることはできないため，クラスタ数を対象データ作成に用いたガウス分布の 2 倍から 2 までをハイパーパラメータとして与え試行し，その中で最も平均対数尤度の大きい

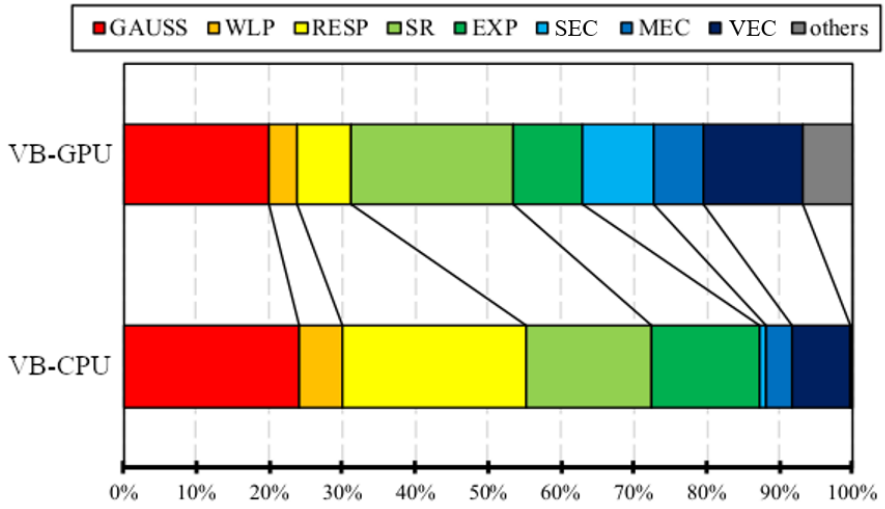

Figure 13 各カーネルの処理時間の割合の比較

ものを最適な混合数とした．VB-GPU は，4.1 同様，クラスタ数は対象データ作成に用いたガウス分布の個数の 2 倍とした．

4.2.1 データ数の変更

対象データのクラスタ数を 16，次元数を 8，データ数を 1024($= 2^{10}$) から 16777216($= 2^{24}$) のデータを用いて実行時間と平均対数尤度の比較を行った．この実験の評価結果を図 14 に示す．

図 14 より，データ数が小さい際，VB-GPU が EM-GPU より高速であることが分かるが，データ数が 2^{20} 以上になると，EM-GPU より低速であることが分かった．これはデータ数の増加に伴い，変分 EM アルゴリズムの収束に必要な回数が，EM アルゴリズムの収束に必要な回数よりも大きくなることが原因です．

4.2.2 クラスタ数を変更

対象データの次元数を 8，データ数を 262144 に固定し，クラスタ数 8 から 256 のデータを用いて実行時間と平均対数尤度の比較を行った．この実験の評価結果を図 15 に示す．図 15 より，どのクラスタ数においても，VB-GPU の実行時間は，EM-GPU の実行時間より小さいことが分かる．クラスタ数が 256 のとき，VB-CPU の実行時間のは EM-GPU の 1/70 であった．

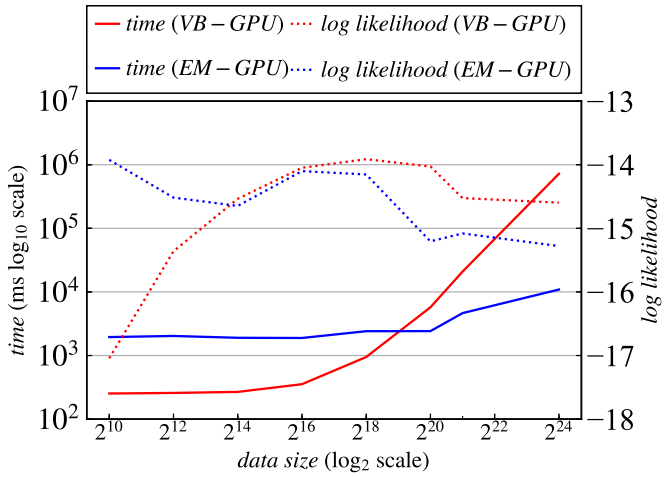

Figure 14　データ数の異なるデータでの EM-GPU との比較 (クラスタ数 16, 次元数 8)

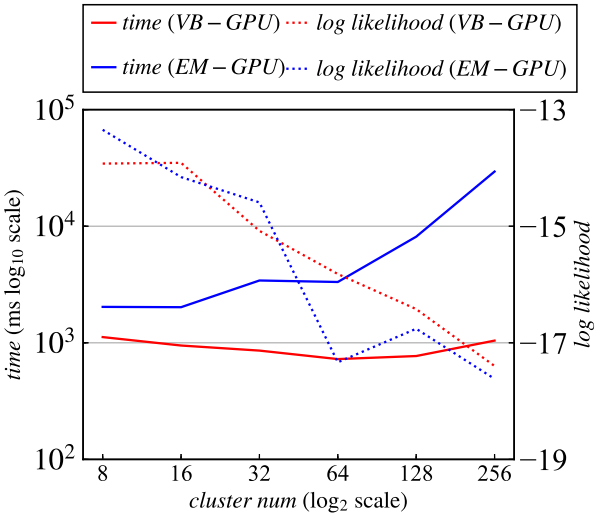

Figure 15　クラスタ数の異なるデータでの EM-GPU との比較 (データ数 2^{18}, 次元数 8)

5. おわりに

既存の最尤推定による混合ガウス分布のパラメータ推定の限界である過学習を防ぎ，最適な混合数の自動決定が可能な変分推論法を用いたモデルの高速化を行うために，GPGPU を用いて変分推論法による混合ガウスモデルのパラメータ推定の並列実装・評価を行った．提案手法は，同アルゴリズムを CPU で実装したものと比較して同じ精度を保ちながら，データ数が 16777216=2^{24} のとき 524 倍，クラスタ数が 256 のとき 220 倍それぞれ高速化できた．また，従来の EM アルゴリズムによる混合ガウスモデルのパラメータ推定 (GPGPU を使用) と比較して，クラスタ数が大きい際，70 倍高速化できた．結果より，クラスタ数が予測し難い大規模なデータのクラスタリングを効率的に行えることが期待できる．

謝辞

本論文の作成にあたり，終始適切な助言を賜り，また丁寧に指導して下さった中島康彦教授，中田尚准教授，木村睦教授，Renyuan Zhang 助教，Tran Thi Hong 助教に感謝します．また，発表に対して貴重なご意見を頂いた本学の池田和司教授，久保孝富准教授に深く感謝いたします．池田裕也くん，岩本淳くん，新谷隆太くんを始め，研究室のメンバーには常に刺激的な議論を頂き，精神的にも支えられました．ありがとうございます．

参考文献

[1] 総務省. “第 1 部 第 2 章 ビッグデータ利活用元年の到来”, 情報通信白書, 2017, pp. 52-105.

[2] Jiawei Han, Jian Pei, Micheline Kamber. “Data mining: concepts and techniques: Edition 3”, Morgan Kaufmann publisher, 2011, pp. 443-491.

[3] Chui-Yu Chiu, Yi-Feng Chen, I-Ting Kuo, He Chun Ku, “An intelligent market segmentation system using k-means and particle swarm optimization” Expert Systems with Applications, Volume 36, Issue 3, Part 1, 2009, pp. 4558-4565.

[4] Kang, Ha-Na & Yong, Hye-Ryeon & Hwang, Hyunseok. “Brand Clustering Based on Social Big Data: A Case Study” International Journal of Software Engineering and Its Applications, 10, 2016, pp. 27-36.

[5] Jing Wu and Zheng Lin. “Research on customer segmentation model by clustering” In Proceedings of the 7th international conference on Electronic commerce (ICEC ’05). Association for Computing Machinery, 2005, pp. 316-318.

[6] Gower, J.C. and Ross, G.J.S. “Minimum Spanning Trees and Single Linkage Cluster Analysis” Journal of the Royal Statistical Society: Series C (Applied Statistics), 18: 1969, pp. 54-64

[7] D. Defays. “An efficient algorithm for a complete link method” The Computer Journal, Volume 20, Issue 4, 1977, pp. 364–366.

[8] Johnson, Sally C. “Hierarchical clustering schemes.”Psychometrika, 32, 1967, pp. 241-254.

[9] S. Lloyd. “Least squares quantization in PCM,” in IEEE Transactions on Information Theory, vol. 28, no. 2, 1982 March, pp. 129-137.

[10] Banfield, Jeffrey D., and Adrian E. Raftery. “Model-Based Gaussian and Non-Gaussian Clustering.”Biometrics, vol. 49, no. 3, 1993, pp. 803–821.

[11] Abbas, Osama Abu. “Comparisons Between Data Clustering Algorithms.” Int. Arab J. Inf. Technol. 5. 2008, pp. 320-325.

[12] Dempster, A. P., Laird, N. M. & Rubin, D. B. “Maximum likelihood from incomplete data via the EM algorithm” Journal of the Royal Statistical Society, Series B, 39, 1977, pp. 1-38.

[13] Geoffrey J. McLachlan, Thriyambakam Krishnan. “The EM Algorithm and Extensions”, John Wiley & Sons publisher. 2007.

[14] C.M.Bishop(2008) “パターン認識と機械学習 下-ベイズ理論による統計的予測” 元田 浩, 栗田 多喜夫, 樋口 知之, 松本 裕治, 村田 昇監訳, 丸善出版, pp. 193-195.

[15] C.Adrian, C.M.Bishop, “Variational Bayesian Model Selection for Mixture Distributions” in Proceedings Eighth International Conference on Artificial Intelligence and Statistics. Morgan Kaufmann, 2001, pp. 27-34.

[16] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E., “Scikit-learn: Machine Learning in Python” Journal of Machine Learning Research, vol. 12 pp. 2825–2830, 2011

[17] scikit-learn, “Choosing the right estimator” [Online]. Available:`https://scikit-learn.org/stable/tutorial/machine\_learning\_map`

[18] D.A.Reynolds, T.F.Quatieri, and R.B.Dunn, “Speaker verification using adapted Gaussian mixture models” Digital Signal Process., vol.10, no.1, 2000, pp.19–41.

[19] t. toda, a. w. black, and k. tokuda, “voice conversion based on maximum likelihood estimation of spectral parameter trajectory” ieee trans. audio, speech, lang. process, vol. 15, no. 8, nov. 2007. pp. 2222–2235,

[20] C.Stauffer and W.Grimson, “Adaptive background mixture models for real-time tracking,” in IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1999, pp. 246–252, vol.2, IEEE.

[21] 藤田 迪, 梶 克彦, 河口 信夫, “Gaussian Mixture Model を用いた無線 LAN 位置推定手法” 情報処理学会論文誌, 1882-7764, 一般社団法人情報処理学会, 2011-03-15,52,3 pp. 1069-1081

[22] J. Nickolls and W. J. Dally, ”The GPU Computing Era,” in IEEE Micro, vol. 30, no. 2, March-April 2010, pp. 56-69. doi: 10.1109/MM.2010.41

[23] N. S. L. P. Kumar, S. Satoor and I. Buck, “Fast Parallel Expectation Maximization for Gaussian Mixture Models on GPUs Using CUDA,” 2009 11th IEEE International Conference on High Performance Computing and Communications, Seoul, 2009, pp. 103–109.

[24] Guo, Ce et al. “A fully-pipelined expectation-maximization engine for Gaussian Mixture Models.” 2012 International Conference on Field-Programmable Technology , 2012, pp. 182–189.

[25] He, Conghui et al. “A Fully-Pipelined Hardware Design for Gaussian Mixture Models.” IEEE Transactions on Computers 66 , 2017, pp. 1837–1850.

[26] M. Harris, “Optimizing parallel reduction in cuda” [Online]. Available:`http://developer.download.nvidia.com/compute/cuda/1\_1/Website/projects/reduction/doc/reduction.pdf`

[27] A. D. Pangborn, “Expectation maximization with a Gaussian mixture model using CUDA.” [Online]. Available: `https://github.com/Corv/CUDA-GMM-MultiGPU`

[28] NVIDIA Visual Profiler, “NVIDIA Visual Profiler”, [Online]. Available: https://developer.nvidia.com/nvidia-visual-profiler

発表リスト

[1] 西本宏樹, 中田尚, 中島康彦, “変分混合ガウスモデルアクセラレータ設計のための変分推論アルゴリズムの解析”, 信学技報, vol. 118, no. 334, VLD2018-62, pp. 155-160, 2018 年 12 月.
[2] 西本宏樹, 中田尚, 中島康彦, “GPGPU を用いた変分混合ガウスモデルのパラメータ推定高速化”, 信学技報, vol. 119, no. 76, CPSY2019-1, pp. 1-5, 2019 年 6 月.
[3] Hiroki Nishimoto, Takashi Nakada, Yasuhiko Nakashima, “GPGPU Implementation of Variational Bayesian Gaussian Mixture Models”, Proceedings of 2019 Seventh International Symposium on Computing and Networking (CANDAR), 6 pages (2019.11). The Seventh International Symposium on Computing and Networking (CANDAR’19)