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

2008－DBS－146 (12)
2008／9／21

LDAのためのcollapsed変分ベイズ推定のGPUによる高速化

正田 備也† 濱田 剛† 柴田裕一郎† 小栗 清†

† 長崎大学工学部 〒852–8521 長崎県長崎市文教町 1–14
E-mail: †{masada,hamada,shibata,oguri}@cis.nagasaki-u.ac.jp

あらまし 本論文では，latent Dirichlet allocation (LDA) のための collapsed 変分ベイズ推定を Graphics Processing Unit (GPU) を用いて高速化する方法を提案する．LDA は，ベイズ理論に基づくマルチトピック文書モデルとして知られているが，Expectation-Maximization (EM) アルゴリズムが利用可能な probabilistic latent semantic indexing (PLSI) など他の文書モデルに比べ，パラメータ推定が複雑で膨大な計算を要する．そこで，LDA のための deterministic なパラメータ推定方法として優れている collapsed 変分ベイズ推定を，GPU を用いて高速化した．実験では約 500 万組の文書と単語のユニークなペアについて，1 つの Nvidia GeForce 8800 GT 上で collapsed 変分ベイズ推定を実行，20 Gflops の計算速度を得た．

キーワード GPU，テキスト・マイニング，latent Dirichlet allocation，collapsed 変分ベイズ推定，高速化

GPU Acceleration of collapsed variational Bayesian inference for LDA

Tomonari MASADA†, Tsuyoshi HAMADA†, Yuichiro SHIBATA†, and Kiyoshi OGURI†

† Faculty of Engineering, Nagasaki University Bunkyo-machi 1–14, Nagasaki-shi, Nagasaki, 852-8521 Japan
E-mail: †{masada,hamada,shibata,oguri}@cis.nagasaki-u.ac.jp

Abstract In this paper, we propose a method for executing collapsed variational Bayesian inference for latent Dirichlet allocation (LDA) on Graphics Processing Unit (GPU). While LDA is a well-known multi-topic document model based on Bayesian methods, it requires complicated inference, which leads to enormous computations in comparison with other document models, e.g. probabilistic latent semantic indexing (PLSI), to which Expectation-Maximization (EM) algorithm is applicable. Therefore, we accelerate collapsed variational Bayesian inference, known as an efficient deterministic inference method for LDA, by using GPU. In the experiments, we used about 5 million unique pairs of documents and words. We achieved 20 Gflops on a single Nvidia GeForce 8800 GT.

Key words GPU, text mining, latent Dirichlet allocation, collapsed variational Bayesian inference, acceleration

1. はじめに

本論文では，latent Dirichlet allocation (LDA) [3] のための collapsed 変分ベイズ推定 [11] を，Graphics Processing Units (GPU) を使って高速化する手法を論じる．

LDA は，元は文書モデルとして提案されたが，近年，文書データ以外へも盛んに適用されている [12] [13]．しかし，Dirichlet compound multinomial (DCM) [7] や probabilistic latent semantic indexing (PLSI) [6] など，Expectation-Maximization (EM) アルゴリズムが利用可能な文書モデルに比べ，モデルの構造が複雑で，大規模データへの適用には推定計算の高速化が必要と考えられる．EM アルゴリズム一般の高速化についてはすでに提案がある [4] が，本論文では，単語や文書のクラスタリング，情報検索などの応用に影響が出ない程度に，LDA のための推定計算に近似を導入し，高速化する方法を提案する．特に，変分ベイズ法 [3] よりも近似の程度が少なく，かつ，Gibbs サンプリング [5] とは異なり deterministic な方法である collapsed 変分ベイズ推定（以下，CVB 推定と略す）を採り上げる．高速化は，GPU を使って推定計算を並列化することにより実現する．具体的には，Nvidia GeForce 8800 GT（以下，8800GT と略す）のような膨大な数のマルチスレッドによる single instruction multiple data (SIMD) デバイスを利用し，LDA におけるトピック単位の計算を並列実行する．従来は，データを分割して並列に処理することで，LDA の推定計算を高速化していた [10] [8] [9]．本論文が提案する高速化は，この従来型のアプローチと併用することもできる．また，CVB 推定の高速化の提案は，我々の知るかぎり，これまでにない．

2. Nvidia CUDA について

提案手法では，Nvidia CUDA (Compute Unified Device Architecture) を利用して，LDA のための CVB 推定を高速化する．CUDA の詳細は関連のドキュメント [1] にゆずり，CUDA

互換の GPU について今回の実装に関係する点のみを論じる．

第一に，GPU カードにはメモリが搭載されている．以下，これをデバイス・メモリと呼び，CPU から使う通常のメモリをホスト・メモリと呼ぶ．デバイス・メモリの容量は，現在市販されている GPU カードで最大 1 GByte 程度であり，ホスト・メモリに比べ少ない．また，ホスト・メモリとデバイス・メモリ間のデータの転送は遅い．したがって，通常は，まず GPU 上で処理すべきデータをデバイス・メモリに転送し，GPU 上で計算を行った後に，必要な計算結果をホスト・メモリに戻す．このようにして，GPU 上で計算中にホスト・メモリを参照することは避ける．よって，数 GByte のメモリを使用するアプリケーションでは，データを分割して GPU 上で処理する．

第二に，GPU 上では膨大な数のスレッドを並列に実行させることができる．スレッド数は GPU により異なる．例えば，8800GT では，512 のスレッドがブロックと呼ばれる実行単位を形成し，さらに，複数のブロックをまとめたグリッドという上位の実行単位がある．GPU の性能を引き出すためには，できるだけ多くのスレッドを並列実行させる必要がある．また，スレッドはそれ自身の高速なメモリを持つが，デバイス・メモリに比べて容量が小さい（8800GT では GPU 全体で 64 KByte）．さらに，同じブロックに属するスレッド間で，同程度に高速なメモリを共有できるが，やはり，デバイス・メモリに比べて容量が小さい（8800GT ではブロック毎に 16 KByte）．これらの高速なメモリは，工夫して使う必要がある．

第三に，現在市販されているほとんどの GPU で，浮動小数点演算は単精度である．よって，計算が破綻しないよう，演算の順序や括弧のつけ方に注意してコーディングする必要がある．

第一の点については，データの量が多いと，ホスト・メモリとデバイス・メモリ間のデータ転送の回数が増え，高速化の利点が打ち消される．どの程度のデータ量が閾値になるかは，アプリケーションにより異なる．デバイス・メモリとホスト・メモリ間のデータ転送が頻繁であっても，データ量に対する計算量が相対的に多ければ，データ転送のオーバーヘッドは小さい．

第二の点については，デバイス・メモリへのアクセスをいかに減らすかが重要となる．頻繁に利用する変数は，各スレッドのメモリや，ブロック単位で共有されるメモリなど，高速なメモリに配置する．また，計算終了後，高速なメモリからデバイス・メモリに書き戻すデータも，少ないほうがよい．

第三の単精度演算に関しては，我々の実験の結果から，LDA のための CVB 推定については単精度演算で十分と思われる．また，本論文では扱わないが，LDA のための Gibbs サンプリング [5] も，単精度演算で十分と思われる．なお，LDA のような複雑な確率モデルの推定計算に関して，ガンマ関数など特殊関数の計算が必要とされる場合 [3] に，単精度計算で間に合うかについての検討は，今後の課題とする．

3. LDA のための CVB 推定の実装

3.1 通常の実装

LDA にはいくつかの推定方法があるが，本論文では，deterministic な推定方法では優れているとされる CVB 推定 [11] を採用し，GPU 上で実行する．式の導出など詳細は原論文にゆずり，ここでは実装上の問題を論じる．以下で使う記号は，原論文と一致させた．CVB 推定は，2 つのフェーズからなる．

- 変分パラメータ γ_{ijk} を更新するフェーズ．文書 j の i 番目の単語について，$\gamma_{ij1}, \ldots, \gamma_{ijK}$ という K 個の変分パラメータが，K 個あるトピックの各々に対応して定義される．なお，$\sum_k \gamma_{ijk} = 1$ が成り立つ．各文書について，出現しない単語にはパラメータは定義されない．また，文書 j の異なる位置 i と i' に現れる同じ単語については，γ_{ijk} と $\gamma_{i'jk}$ が等しいとする [11]．つまり，変分パラメータは文書 j と単語 w のユニークなペアついて定義される．以下，各ペアに対応する K 個のパラメータを $\gamma_{wj1}, \ldots, \gamma_{wjK}$ と書く．変分パラメータの個数は，文書と単語のユニークなペアの個数を M として，MK となる．
- 更新された変分パラメータに基づいて，各種の平均と分散を計算し直すフェーズ．平均は $E_{\hat{q}}[n_{jk\cdot}]$，$E_{\hat{q}}[n_{\cdot kw}]$，$E_{\hat{q}}[n_{\cdot k\cdot}]$，分散もこれらに対応して $\mathrm{Var}_{\hat{q}}[n_{jk\cdot}]$，$\mathrm{Var}_{\hat{q}}[n_{\cdot kw}]$，$\mathrm{Var}_{\hat{q}}[n_{\cdot k\cdot}]$ の 3 種類ずつある．正確は，これらの平均や分散は，ガウス分布による近似の結果として導入されている [11]．文書数を N，語彙数を W，トピック数を K とすると，3 種類の平均および分散は，それぞれ NK 個，KW 個，K 個ずつある．

通常の実装では，変分パラメータの更新と，平均および分散の再計算という 2 つのフェーズを，文書と単語のユニークなペアの各々について実行する．具体的には，文書 j に現れる単語 w について，まず，K 個の変分パラメータ $\gamma_{wj1}, \ldots, \gamma_{wjK}$ が更新され，次に，3 種類の平均と 3 種類の分散のうち j と w のペアに関係するもののみ再計算される．よって，文書集合全体をスキャンする 1 回の iteration の時間計算量は $O(MK)$ となる．

3.2 提案する実装

3.2.1 デバイス・メモリの使い方

本論文では，MK 個の変分パラメータ γ_{wjk} の一部しかデバイス・メモリに置くことができない状況も想定する．この場合，文書集合を分割し，各部分集合を逐次的に GPU 上で処理する．このように文書集合を分割する場合，そのつど，ホスト・メモリとデバイス・メモリ間のデータ転送が必要となる．文書集合のこのような分割を，本論文ではホスト上の分割と呼ぶ．ホスト上の分割で得られた複数の文書部分集合には，GPU とデバイス・メモリの組が複数あれば，従来型の並列化 [10] [8] [9] を使えるが，本論文では議論しない．その一方，3 種類の平均 $E_{\hat{q}}[n_{jk\cdot}]$，$E_{\hat{q}}[n_{\cdot kw}]$ $E_{\hat{q}}[n_{\cdot k\cdot}]$ と，3 種類の分散 $\mathrm{Var}_{\hat{q}}[n_{jk\cdot}]$，$\mathrm{Var}_{\hat{q}}[n_{\cdot kw}]$ $\mathrm{Var}_{\hat{q}}[n_{\cdot k\cdot}]$ については，すべてデバイス・メモリ上に保持できると仮定する．なぜなら，多くの場合，変分パラメータの空間計算量 $O(MK)$ が支配的だからである．

3.2.2 トピック単位での並列化

本論文では，以下 2 種類の K-way の計算が独立に実行できることに注目し，トピック単位での計算を並列実行することで，CVB 推定の高速化を実現する．

- 文書と単語のペア j, w の各々についての，K 個の変分パラメータ $\gamma_{wj1}, \ldots, \gamma_{wjK}$ の更新
- 3 種類の平均 $E_{\hat{q}}[n_{jk\cdot}]$，$E_{\hat{q}}[n_{\cdot k\cdot}]$，$E_{\hat{q}}[n_{\cdot kw}]$ と，3 種類の分散 $\mathrm{Var}_{\hat{q}}[n_{jk\cdot}]$，$\mathrm{Var}_{\hat{q}}[n_{\cdot k\cdot}]$，$\mathrm{Var}_{\hat{q}}[n_{\cdot kw}]$ それぞれの，異なるト

ピックに対応する K 通りの値の再計算

従来研究 [10] [8] [9] には，文書集合の分割による高速化は見られるが，上記のようなトピック単位の計算を同時に実行することによる高速化は見られない．提案手法では，上記の 2 種類の K-way の計算を，異なる K 個のスレッドで実行する．

ただし，本論文でも，計算をさらに高速化するため，文書集合の分割による高速化を，GPU 上での計算において併用する．文書集合を c 個の部分集合に分割する場合，同時に cK 個のスレッドが実行されることになる．しかし，ここで c 分割される文書集合は，すでにホスト・メモリからデバイス・メモリに転送されているデータであり，ホスト上でデバイス・メモリの容量に合わせて分割された後の文書部分集合かもしれない．そこで，デバイス・メモリにある文書集合の分割を，本論文ではデバイス上での分割と呼び，ホスト上での分割とは区別する．

3.2.3 GPU のための CVB 推定

ここで，文書 j と単語 w の各ペアに関する計算の詳細を示す．

（1） 現在の変分パラメータ γ_{wjk} の寄与分を，平均 $E_{\hat{q}}[n_{jk\cdot}]$，$E_{\hat{q}}[n_{\cdot kw}]$，$E_{\hat{q}}[n_{\cdot k\cdot}]$ と分散 $\mathrm{Var}_{\hat{q}}[n_{jk\cdot}]$，$\mathrm{Var}_{\hat{q}}[n_{\cdot kw}]$，$\mathrm{Var}_{\hat{q}}[n_{\cdot k\cdot}]$ から，減算する．

（2） 減算後の平均と分散を使い変分パラメータを更新する．

（3） 更新された変分パラメータを，$\sum_k \gamma_{wjk} = 1$ が満たされるように正規化する．

（4） 更新後の変分パラメータの寄与分を，上記の 3 種類の平均と 3 種類の分散に，加算する．

以上の一連の計算は，変分パラメータの正規化を除き，K 個のスレッドで並列実行できる．正規化も，総和計算の並列化により，時間計算量が $O(\log K)$ となる．よって，パラメータ推定計算のみの時間計算量は $O(\frac{M}{c}\log K)$ となる．ただし，デバイス・メモリとホスト・メモリ間のデータ転送のオーバーヘッドは考慮されていない．また，同時に実行可能なスレッド数には上限があるため，$T = cK$ は定数と考えるべきである．よって，時間計算量は $O(\frac{M}{T}K\log K)$ となる．

3.2.4 高速なメモリの使い方

ここでは，推定計算に現れる定数や変数のそれぞれについて，高速なメモリに配置できるかどうかを論じる．

CVB 推定では，各文書での各単語の出現頻度を用いる．このデータについては，文書と単語のユニークなペア j, w に関する K 個の変分パラメータ $\gamma_{wj1}, \ldots, \gamma_{wjK}$ を更新する，K 個のスレッドだけが，同じ値を共有する．したがって，ブロック単位で共有される高速なメモリに読み込んでおく．また，このデータは定数であり，デバイス・メモリへの書き戻す必要はない．

変分パラメータは，正規化されるとき，関係する K 個のスレッドが共有しなければならない．よって，ブロック単位で共有される高速なメモリ上に，必要な変分パラメータをデバイス・メモリから読み込んでおく．そして，更新された変分パラメータは，そのつどデバイス・メモリに書き戻す．

3 種類の平均と 3 種類の分散のうち，文書とトピックのペア j, k の各々に対応する平均 $E_{\hat{q}}[n_{jk\cdot}]$ と分散 $\mathrm{Var}_{\hat{q}}[n_{jk\cdot}]$ は，文書集合をデバイス上で c 個に分割し，同時に実行されるスレッドを増やしても，複数のスレッドで共有されはしない．よって，これらの値は高速なメモリに読み込んでおき，同じ文書について計算がおこなわれている間，使い続けることができる．こうして，遅いデバイス・メモリへのアクセスが回避できる．

しかし，デバイス上で文書集合を分割すると，K 個のトピックそれぞれに対応する平均 $E_{\hat{q}}[n_{\cdot k\cdot}]$ と分散 $\mathrm{Var}_{\hat{q}}[n_{\cdot k\cdot}]$ は，同じトピックに関わる全スレッドにより，再計算される．そこで，文書の c 個の部分集合ごとに別々の値を用いる．したがって，計算は近似となる．だが，この近似により，平均 $E_{\hat{q}}[n_{jk\cdot}]$ と分散 $\mathrm{Var}_{\hat{q}}[n_{jk\cdot}]$ の場合と同様，高速なメモリを利用できる．

最後に，単語とトピックの各ペア w, k に対応する平均 $E_{\hat{q}}[n_{\cdot kw}]$ と分散 $\mathrm{Var}_{\hat{q}}[n_{\cdot kw}]$ については，デバイス上での分割で得られる c 個の部分集合ごとに別々の値を用いると，近似が大きくなりすぎる．なぜなら，ホスト上とデバイス上とで 2 度の分割を経ることにより，各部分集合に含まれる文書数が少なくなっており，部分集合内での単語の頻度分布が，文書集合全体での頻度分布から大きく外れている可能性があるからである．したがって，これらの平均と分散については，遅いデバイス・メモリに配置し，並列して実行されているスレッド間で，ブロックを横断して共有することにする．だが，これら平均と分散の現在の値を読み出し，それをもとに変分パラメータを更新し，更新された変分パラメータによって平均と分散を再計算するまでの間に，別の文書を担当しているスレッドが，同じ単語について平均と分散の再計算をしている可能性がある．したがって，値の一貫性を保証するためには，排他制御をする必要がある．しかし，それでは高速化の効果が損なわれるため，提案手法では，排他制御をしないことにする．ところが，このように，平均 $E_{\hat{q}}[n_{\cdot kw}]$ と分散 $\mathrm{Var}_{\hat{q}}[n_{\cdot kw}]$ をデバイス・メモリに配置し，排他制御なしに多数のスレッドで同時に読み書きすると，スレッドを一定数以上に増やしたとき，変分パラメータの値が変わらないまま推定計算が進むようになった．そこで，1 ブロックのスレッド数を 256，ブロック数を 16 とすることで，総スレッド数 T を 1,024 に抑えた．

3.2.5 GPU 上での計算の後処理

提案手法では，GPU 上での計算が終わった後，3 種類の平均と 3 種類の分散は，ホスト・メモリに戻さないことにし，更新された変分パラメータだけをホスト・メモリに戻す．そして，3 種類の平均と 3 種類の分散を，変分パラメータの全体を使って，ホスト上で計算し直す．これは，デバイス・メモリとホスト・メモリ間のデータ転送のオーバーヘッドを最小限にするためである．また，ホスト上で文書集合を分割している場合は，ホスト・メモリに戻される変分パラメータが全体のうちの一部となり，変分パラメータの全体を使って平均や分散を計算すれば，近似の程度を緩和できる．

4. 実　　験

4.1 実験の設定

GPU 上での並列計算の効果を評価するため，2007 年 11 月 16 日から 2008 年 5 月 15 日に毎日新聞と朝日新聞の Web サイトからクローリングした計 56,755 件のニュース記事を用いる．MeCab [2] による形態素解析の後，高頻度語，出現が 5 回

表 1　テスト・データの perplexity．K はトピック数．10 回の試行の平均と標準偏差を示してある．

	$K=16$	$K=32$	$K=64$
GPU 未使用	1933.7 ± 19.7	1762.0 ± 14.0	2019.0 ± 27.6
GPU 使用	1947.5 ± 18.1	1785.0 ± 19.6	2068.5 ± 34.4

未満の語，半角文字からなる語などを除去し，40,158 の語彙数を得た．文書と単語のユニークなペアの数は 5,053,978 である．LDA のトピック数 K は 16，32，64 の 3 通りを用いる．推定結果は perplexity で評価する．各文書に現れる単語のトークンを 2 等分し，半分を推定計算に，残り半分を perplexity を求めるために，それぞれ用いた．各文書で 1 回だけ現れる単語は，推定計算に使う．その結果，文書と単語のユニークなペア 5,053,978 個のうち，推定計算に使われた個数 M は 3,387,822 個だった．CVB 推定の時間計算量は，この個数とトピック数の積に比例する．CVB 推定は，変分パラメータをランダムに初期化した後，開始する．10 通りのランダムな初期値から始め，文書集合全体の 1 回の処理を 1 回の iteration と定義し，この iteration を 100 回実行する．ただし，perplexity に 0.02% 未満の変動しか見られなかった区間を省くことで，本論文では 64 回目までの iteration の結果だけを提示する．

実験には，Intel Core2Quad Q9450 2.66GHz 搭載の PC に，ASUS EN8800GT/HTDP/1G という GPU カードを 1 枚挿した環境を利用した．この GPU カードのメモリの容量は 1 GByte である．ただし，1 GByte 全体を使おうとすると，ホスト・メモリからデバイス・メモリへのデータ転送をおこなう際にエラーが発生したため，デバイス・メモリは 3/4 程度の容量だけを使った．今回の文書データでは，$K=16$ の場合のみ，ホスト上での文書集合の分割はおこなう必要がなく，デバイス・メモリにすべてのデータを配置できている．$K=32, 64$ の場合，それぞれ，全体の 85%, 43% 程度のデータしかデバイス・メモリに配置できない．これは，変分パラメータの個数が MK 個であり，必要なメモリ量がトピック数に比例して増えるためである．したがって，$K=32, 64$ の場合，1 回の iteration あたり，それぞれ約 1.2 回，約 2.3 回，デバイス・メモリとホスト・メモリ間のデータ転送が生じる．一般に，この回数が多くなるほど，計算の高速化の効果は打ち消される．

4.2　実験の結果と考察

表 1 に，64 回の iteration が終了した後の perplexity を示す．トピック数が 16，32，64 の各々について，GPU を用いない場合と GPU を用いて近似並列計算した場合に，10 回の推定が与える perplexity の平均値と標準偏差を示してある．全体として，GPU を用いる場合の平均，標準偏差がともにやや大きい．だが，どのトピック数でも，GPU を用いない場合と用いる場合とで，標準偏差の ±1 倍の区間は交わっている．文書モデルとしての LDA に着目すると，今回の実験では，トピック数が 32 のときに最も小さな perplexity が得られた．

次に，提案手法による GPU 上での近似並列計算をおこなう場合と，そうでない場合とで，アプリケーションのレベルで違いがあるかどうかを，表 2 と表 3 により直感的に確認する．それぞれ，トピック数 64 の場合の，10 の実行結果のうちの 1 つにおいて，各トピック k について，単語を $\sum_j \gamma_{wjk}$ の降順にソートした後，$\sum_j \gamma_{wjk} / \sum_{k,j} \gamma_{wjk} \geqq 0.8$ を満たす単語を，$\sum_j \gamma_{wjk}$ が大きい順に上位から 8 個を示している．太字は $\sum_j \gamma_{wjk} / \sum_{k,j} \gamma_{wjk} \geqq 0.9$ を満たす単語である．空行は，$\sum_j \gamma_{wjk} / \sum_{k,j} \gamma_{wjk} \geqq 0.8$ となる単語を持たないトピックに対応する．つまり，空行に対応するトピックは，そのトピックに強く偏って出現する単語がなかった，あまり特徴のないトピックと解釈できる．これらの表の結果は，LDA を利用した単語クラスタリングと言える．2 つの表を比較して，一方に現れている単語クラスタが，他方の結果で対応するクラスタを持たないことがある．しかし，これは，GPU を用いない複数の推定結果の間でも起こることである．異クラスタ間での単語の意味的な分離の度合いや，クラスタ内での単語の意味的な関連の度合いについて，GPU を使う場合と使わない場合とで目立った違いはないように思われる．

最後に，図 1，2，3 に，64 回の iteration の各段階での perplexity と実行時間との関係を示す．本論文では，LDA のための推定計算という具体的なアプリケーションに GPU を用いるため，flops による評価ではなく，実際の実行時間を示す．今回の実験では，GPU 上で実行されている計算について，約 20 Gflops の性能を得ているが，現実的には，デバイス・メモリとホスト・メモリ間のデータ転送のオーバーヘッドを含め，他の時間消費を考慮すべきである．そのため，time 関数で計った実行時間を横軸として，perplexity の変化を図に示した．

図 1，2，3 は，それぞれ，トピック数 K が 16，32，64 の場合であり，10 通りのランダムな初期値から始めた推定計算のすべてについて，実行時間と perplexity の関係を表している．いずれの図においても，左 10 個のグラフが GPU を使用した場合であり，右 10 個のグラフが GPU を使用しない場合である．十字のマーカは，64 回の iteration を表している．GPU を使用しない場合の実行時間は，平均 34.0 分，67.3 分，134.3 分と，トピック数にほぼ比例している．その一方，GPU を使用する場合の実行時間は，平均 4.2 分，9.9 分，28.3 分と，トピック数に対して超線型に増えている．その理由は次の通りである．

提案手法による推定計算では，トピック単位での並列化をおこなっている．したがって，$O(\log K)$ の時間計算量で変分パラメータの正規化をおこなう部分が，計算時間の中では支配的となる．今回，3. 2. 4 節の最後に示した理由により，最大のスレッド数を一定値 1,024 に固定しているため，変分パラメータの正規化の時間計算量は $\frac{K}{1024} \times \log K$ に比例する．つまり，3. 2. 2 節で述べた，デバイス上での文書集合の分割数 c が，同時に実行できる正規化計算の個数であり，この値は $c = \frac{1024}{K}$ という式を満たすように決められる．そのため，トピック数の増加とともに，同時に実行できる正規化計算の個数が減少し，トピック数の比率以上の比率で，実行時間が増大する．さらに，デバイス・メモリとホスト・メモリ間のデータ転送のオーバーヘッドが，K とともに増加することも加わって，GPU 使用時には，トピック数に対して超線型に実行時間が増えている．

なお，デバイス・メモリ上のデータに，複数回の計算を続け

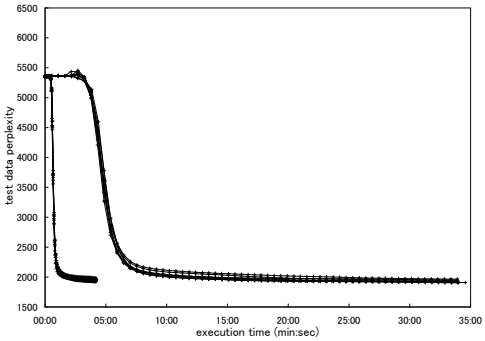

図 1　実行時間とテストデータ perplexity の関係．トピック数 16.

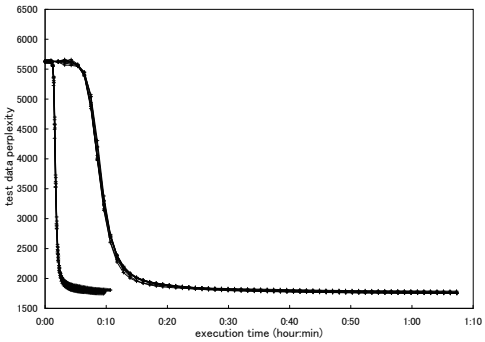

図 2　実行時間とテストデータ perplexity の関係．トピック数 32.

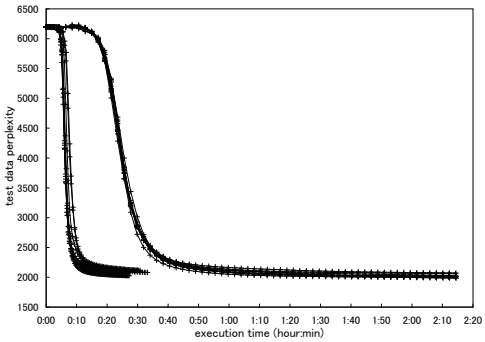

図 3　実行時間とテストデータ perplexity の関係．トピック数 64.

て実行すれば，さらに計算時間が縮まる．なぜなら，flops 値は変わらないが，デバイス・メモリとホスト・メモリ間のデータ転送の iteration 毎の回数が減るためである．しかし，近似計算の影響が蓄積され，収束時の perplexity が悪くなりうる．特に，ホスト上で文書集合を分割している場合，部分集合に対してパラメータを強くフィットさせることになる．よって，デバイス・メモリ上のデータに，複数回の計算を連続して実行する場合は，近似の影響を適切に見積もることが重要である．

5. おわりに

本論文では，Nvidia CUDA を利用して，LDA のための collapsed 変分ベイズ推定を高速化する手法を提案した．実験は，文書データ以外のデータについてもおこなっているが，詳細は別の機会に論じたい．ひとつ指摘するとすれば，今回使ったデータの約 2 倍の量の文書データを，512 MByte という，今回の半分のデバイス・メモリしか搭載しない GPU カードで処理すると，トピック数 64 のとき，GPU を使用しない場合とほぼ同じ実行時間となった．これは，デバイス・メモリとホスト・メモリ間のデータ転送のオーバーヘッドが，GPU による高速化の効果を打ち消したためである．このような状況では，従来のデータ分割による高速化を併用する必要がある．

本論文では，GPU 上での並列計算による効果が見えやすい実験結果を示した．この結果から，従来のデータ分割型の並列化とは異なる，細かい粒度の並列化の有効性が明らかとなった．細粒度の並列化とは，LDA の場合で言えば，トピック単位の計算の並列実行である．他の機械学習手法についても，データの分割により得られる並列性だけでなく，モデルの構造に由来する並列性を活用すれば，計算のさらなる高速化が可能になると思われる．これら 2 つの高速化を組み合わせれば，今まで現実的な時間で処理しきれなかったサイズのデータに，LDA などの高度なデータ・モデルを適用できるようになると予想される．

謝　辞

この研究は，長崎大学における，平成 19 年度文部科学省科学技術振興調整費（若手研究者の自立的研究環境整備促進），採択課題「地方総合大学における若手人材育成戦略」，課題名「リアルタイム情報処理による技術融合」により整備された研究環境を利用して実施されました．

文　献

[1] NVIDIA CUDA `http://www.nvidia.com/cuda`
[2] MeCab `http://mecab.sourceforge.net/`
[3] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *JMLR* Vol. 3, pp. 993-1022, 2003.
[4] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun. Map-Reduce for Machine Learning on Multicore. *NIPS 19*, pp. 281-288, 2007.
[5] T. Griffiths and M. Steyvers. Finding Scientific Topics. *PNAS*, Vol. 101 (suppl. 1), pp. 5228-5235, 2004.
[6] T. Hofmann. Probabilistic latent semantic indexing. *SIGIR'99*, pp. 50-57, 1999.
[7] R. Madsen, D. Kauchak, and C. Elkan. Modeling word burstiness using the Dirichlet distribution. *ICML2005*, pp. 545-552, 2005.
[8] R. Nallapati, W. Cohen, and J. Lafferty. Parallelized Variational EM for Latent Dirichlet Allocation: An experimental evaluation of speed and scalability. *ICDM workshop on high*

表 2　単語クラスタ: GPU を用いない collapsed 変分ベイズ推定

細胞, ｉＰＳ, 遺伝子, インフルエンザ, 幹, ヒト, マウス, 神経,
福祉, 離婚, 出産, 戸籍, アンケート, 認知, 喫煙, 禁煙,
病院, 医師, 診療, 医, 入院, 救急, 手術, 看護,
チベット, ミャンマー, 聖火, リレー, 被災, 暴動, デモ, 四川,
選手, 試合, 出場, 優勝, 野球, サッカー, プロ, 選手権,
ブラジル, フィリピン, 日系, 誘拐, マニラ, ジャカルタ, スハルト, 在留,
階, 建て, 消防, 出火, スキー, 木造, 水素, 全焼,
マニフェスト, 英夫, 東国, 国原, 徴兵, 新語, ＩＫＫＯ, 正恭,
死去, 葬儀, 賞, 喪主, 訃報, 受賞, 曲, 近親,
最終, 更新,
詐欺, インサイダー, サロン, 商法, 警視, 偽造, だまし取る, 架空,
ぼけ,
排出, 温暖, 目標, 温室, サミット, 気候, 変動, 枠組み,
州, 投票, クリントン, オバマ, 上院, 指名, 予備, プーチン,
無職, 同習, 刺す, 強盗, 盗む, 殴る, 署員, 窃盗,
銃, 佐世保, 暴力団, 発砲, 乱射, 馬込, 射殺, 撃つ,
公取委, 残業, 日雇い, グッドウィル, 罰則, 公文書, ＥＤＩＮＥＴ, 荷役,
沖縄, 基地, 米兵, 路, 滑走, ＪＡＬ, 海兵, 日本航空,
イラク, イスラエル, パキスタン, 治安, パレスチナ, イスラム, 武装, ブット,
乗用車, 飲酒, 車線, 致死傷, バイク, 交差点, 後部, 追突,
学校, 教育, 生徒, 教委, 小学校, 授業, 校長, 学習,
値上げ, トヨタ, リットル, コメ, 灯油, ＧＭ, 穀物, 小麦,
思う, 言う, 私, 何, 今, ん, 自分, いい,
原告, 救済, 和解, 薬害, 肝炎, Ｃ型肝炎, 石綿, 提訴,
国交, 戸, 柴, ダム, 宿舎, 特会, 河川, 鉄三,
記事, 検索, 入力, キーワード,
増, 前年, 減, 減速, 物価, 伸び, 同月, 好調,
北朝鮮, 核, 首脳, 主席, 外交, 外相, 胡, コソボ,
石原, 橋下, 徹, 府知事, 慎太郎, 都議会, 都知事, 府民,
古森, 福地, 正弘, 重隆, 二橋, 篤郎, 若返り, 勝俣,
修理, リコール, 不具合, 発火, ボルト, 電動, エスカレーター, 破断,
買収, 株主, ヤフー, ＭＳ, 労組, 賃上げ, 春闘, パワー,
駅, 線, ＪＲ, 鉄道, 電車, 新幹線, 列車, 見合わせる,
新, 現, 古賀, 擁立, 市議, 県連, 共, 平岡,
名人, 段, 羽生, 将棋, 森内, 対局, 大夫, 王将,
商標, アイホン,
さま, 皇太子, 陛下, 登頂, 文化財, 愛子, エベレスト, 宮内庁,
イベント, 展示, 帳, 雑記, プレゼント, 入場, キャラクター, チョコ,
保険, 年金, 料, 保, 加入, 後期, 受給, 徴収,
艦, 船, 海上, イージス, あたご, 石破, 漁船, 清,
親方, 力士, 青龍, 大相撲, 横綱, 時津, 兄弟子, 鵬,
被告, 判決, 裁判, 罪, 死刑, 審, 懲役, 検察,
食品, ギョーザ, 製, 冷凍, 中毒, 検出, 袋, 混入,
守屋, 洋行, 入札, 接待, 汚職, 収賄, 秋山, 談合,
国会, 総裁, 法案, 参院, 日銀, 可決, 小沢, 同意,
放送, 番組, 新聞, 出版, 上映, 視聴, 靖国, 週刊,
金融, 銀行, 損失, サブ, プライム, 決算, 銀, 融資,
ゲーム, 東芝, ソニー, 液晶, シャープ, 薄型, 松下, 松下電器産業,
ネット, サイト, インターネット, カード, ＮＴＴ, 接続, 端末, 回線,
大学, 学生, 試験, 合格, 受験, 学長, 学部, 入試,
ドル, 銭, 安, 前日, 終値, 株価, ニューヨーク, 相場,
原発, 古紙, 配合, 製紙, 炉, ごみ, リサイクル, 用紙,
動物, 園, 震度, 気温, 気象庁, 羽, 氷, 気象,
宇宙, 飛行, 土井, 打ち上げる, 衛星, ぼう, ＩＳＳ, ステーション,
店, 店舗, 吉兆, 船場, 店長, 本店, 百貨店, マクドナルド,
年度, 税, 交付, 税制, 補助, 税収, 配分, 課税,

表 3　単語クラスタ: GPU を用いた collapsed 変分ベイズ推定

総裁, 人事, 日銀, 同意, 武藤, 白川, 後任, 昇格,
映画, 賞, 作品, 受賞, 出版, 作家, 上映, 芸術,
イラク, イラン, ミャンマー, 軍事, シリア, アフガン, 駐留, アフガニスタン,
名人, 段, 公邸, 羽生, 将棋, 森内, 文楽, 対局,
保険, 年金, 料, 労働, 高齢, 診療, 加入, 介護,
飛鳥, 槙, 古墳, 法隆寺, 本堂, 上島, 明日香, 壁画,
北朝鮮, 首脳, 核, 力国, 外相, 両国, 韓, 拉致,
買収, 株主, ヤフー, 提携, 古紙, 配合, ＭＳ, 製紙,
イベント, 曲, 応募, 演奏, 帳, 歌, プレゼント, コンサート,
北京, チベット, 主席, 胡, 聖火, リレー, 暴動, ダライ,
農業, 農家, コメ, 収穫, 実習, 農地, 耕作, 畜産,
南大門, 聯合, 慰安, ウォン, 特攻隊, 酒川, 平島, 楼閣,
艦, 船, 海上, イージス, あたご, 石破, 漁船, 清,
最終, 更新,
イスラエル, コソボ, セルビア, パレスチナ, 爆弾, ガザ, 和平, トルコ,
州, クリントン, オバマ, 上院, 予備, マケイン, 共和党, 争い,
税, 整備, 交付, 税制, 会計, 税収, 省庁, 課税,
選手, 試合, 出場, 優勝, 野球, プロ, サッカー, 選手権,
思う, 言う, 私, 何, 今, 自分, ん, いい,
ダム, 新入,ッ, 利水,
金融, 銀行, 損失, サブ, プライム, 金利, ローン, 銀,
ギョーザ, 製, 回収, 冷凍, 中毒, 検出, 袋, 混入,
宇宙, 航空, 空港, 飛行, 土井, 打ち上げる, 衛星, ぼう,
価格, 増, 前年, 減, 値上げ, 高騰, トヨタ, 物価,
原告, 救済, 和解, 薬害, 肝炎, Ｃ型肝炎, 石綿, 製剤,
処分, 不正, 入札, 監査, インサイダー, 発注, 談合, 告発,
被告, 判決, 裁判, 地裁, 罪, 死刑, 審, 懲役,
放送, 携帯, ネット, サイト, インターネット, パソコン, ソフト, 配信,
国会, 衆院, 参院, 法案, 自民党, 可決, 小沢, 幹事,
記事, 検索, 入力, キーワード,
学校, 教育, 大学, 生徒, 学生, 校, 教委, 授業,
プーチン, 議席, 下院, ブット, 総統, モスクワ, 国民党, ムシャラフ,
市長, 橋下, 新, 推薦, 現, 徹, 府知事, 県知事,
容疑, 男, 銃, 刺す, 強盗, 佐世保, 所持, 盗む,
警察庁, 取り調べ, 録画, 録音, 遠山, 冤罪, 試行, 取り調べる,
乗用車, 消防, 出火, 飲酒, 水素, 全焼, 木造, 硫化,
死去, 葬儀, 喪主, 訃報, 近親, 済ませる, 別れ, 肺炎,
桜, 親方, 力士, 青龍, 大相撲, 横綱, 時津, 兄弟子,
沖縄, 守屋, 洋行, 米兵, 接待, 収賄, 秋山, 装備,
地震, 被災, 原発, 四川, 震度, 災害, 震源, 耐震,
サロン, サイバン, ロス, 警視, 移送, 和義, 霊感, 市警,
駅, 線, ＪＲ, 鉄道, 電車, 新幹線, 気温, 列車,
動物, 園, インフルエンザ, 鳥, 羽, 氷, 匹, 飼育,
ドル, 銭, 安, 前日, 終値, 株価, 相場, 為替,
病院, 手術, 臓器, 禁煙, 喫煙, たばこ, 外科, 脳死,
都, 千代田, 石原, 世田谷, 慎太郎, 江東, 豊島, 日教組,
研究, 細胞, 病, ｉＰＳ, 遺伝子, 幹, ヒト, 糖尿,
店, 店舗, 料理, 牛, 吉兆, 牛肉, 船場, 本店,
環境, 削減, 排出, 温暖, ガス, 目標, 地球, 温室,

performance data mining, 2007.

[9] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed Inference for Latent Dirichlet Allocation. *NIPS 20*, pp.1081-1088, 2008.

[10] D. Newman, P. Smyth, and M. Steyvers. Scalable Parallel Topic Models. *Journal of Intelligence Community Research and Development*, 2006.

[11] Y. W. Teh, D. Newman, and M. Welling. A Collapsed Variational Bayesian Inference Algorithm for Latent Dirichlet Allocation. *NIPS 18*, pp. 1353-1360, 2006.

[12] X. Wang and E. Grimson. Spatial Latent Dirchlet Allocation. *NIPS 20*, pp.1577-1584, 2008.

[13] D. Xing and M. Girolami. Employing Latent Dirichlet Allocation for Fraud Detection in Telecommunications. *Pattern Recognition Letters* 28, pp.1727-1734, 2007.