GPGPUによる超並列処理の文書分類への適用

Massively Parallel Computing with GPGPUs for Text Classification: A Case Study

佐々木 裕
Yutaka Sasaki
豊田工業大学
Toyota Technological Institute

yutaka.sasaki@toyota-ti.ac.jp

1　はじめに

文書分類を様々な分野の現実の問題に適用する場合のボトルネックとなっている点に、処理速度の問題がある。従来のように10カテゴリ程度への分類問題であれば、実用時間内に解けるが、現実の問題では、しばしば数千〜数十万カテゴリへの分類が必要となる場合がある。実際、国際医療コードICD-10は、68,000の診断コード(diagnosis code)と87,000の治療コード(procedure code)からなる。また、米国立衛生研究所(NIH)の国立医学図書館(NLM)が定める生物・医学用語のシソーラスであるMeSH（Medical Subject Headings) [3]は、110,000以上のカテゴリからなり、生物・医学系の1800万件以上の論文に対する世界標準の分類キーワードとして使われている。

各医療文書を忠実にすべてのICD-10やMeSHといったコードに分類するためには、何かの実装的な工夫が必要となる。ひとつの有力な方法は、千台規模の大量のコンピュータを導入し、並列処理による高速化を行なうことである。典型的にはone-against-the-rest法[7]による2値分類法によりNカテゴリに文書を分類する場合は、N並列処理によりN倍の高速化が達成できる。しかしながら、大量のコンピュータの導入には、コストや設置スペース、消費電力の問題が伴う。

そこで、本論文では、汎用グラフィックプロセッサ(GPGPU: General-Purpose Graphics Processing Unit)による並列処理を用いた文書分類の高速化の可能性について探求する。

表1: CTPコーパスの概要

項目	値
文書数	82,525
コーパスサイズ	529MB
平均文書サイズ	6.4KB
カテゴリ総異なり数	11,443
トップレベルカテゴリ異なり数	63
平均カテゴリ付与数	25.9
平均トップレベルカテゴリ付与数	3.4

2　CTPコーパス

GPGPUによる高速化を試みる対象の文書分類データとして、Clinical Trial Protocol (CTP)コーパスを用いる。GPGPUへの実装法は、そのGPGPU機構の特異性により、対象となるデータやモデルのサイズに大きく依存する。そこで、本節ではCTPコーパスの概要について簡単に紹介する。

臨床試験（Clinical Trial）とは、新薬などの新しい治療法の開発にあたって、その治療法が人体に与える効果や影響を医学的に調査するための試験である。臨床試験を行なう際には、臨床試験を実施する組織が臨床試験計画書（Clinical Trial Protocol）を作成する。表1にコーパスの概要を示す。全体で11,443種類のカテゴリが付与されている。本研究の長期的なゴールは、これらのすべてのカテゴリへの臨床試験計画書の分類を実現することであるが、本論文では、パイロットスタディとして、トップレベルのカテゴリ63種類を対象とする。

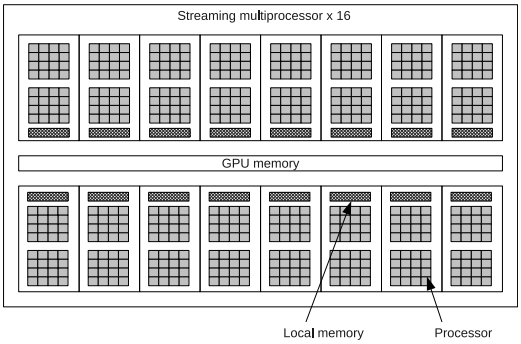

図 1: GPGPU Architecture

3 GPGPU の概要

本論文では、GPGPU を利用するための環境として、NVIDIA 社の CUDA (Compute Unified Device Architecture) 環境を採用する。GPGPU の構成を図 1 に示す。GPGPU はいくつかの Streaming Multiprocessor からなる。Streaming Multiprocessor には、8〜32 のコアプロセッサ (CUDA Core) とこれらのコア間で共有されるローカルメモリ (Shared Memory) が与えられている。また、各 GPGPU ボードにはすべてのコアからアクセス可能な汎用メモリが与えられている。機種によるが、GPGPU には、数〜十数個の Streaming Multiprocessor が搭載されているため、1 枚の GPGPU ボードで合計数十〜数百個のコアが並列に動作する。GPGPU ボードを複数枚搭載すれば、1000 コア以上の並列処理でも比較的安価に実現できる。

GPGPU を使った処理の高速化を実現するためには、いくつかの GPGPU ボードの特性を考慮する必要がある。

1. GPGPU ボードに搭載されているメモリは 1〜3 GB 程度であり、CPU のメインメモリとは異なり、巨大なデータを GPGPU ボードのメモリに一度に載せることはできない。たとえば、情報検索の高速化のために、10GB 以上のインデックスを GPGPU ボードのメモリに一度に載せるといった使い方はできない。

2. 各コアのクロックは 1GHz 前後であるため、順序的な制約等の影響で、GPGPU 内で実質的な並列度があまり上がらない場合は、メイン CPU で処理する方が早くなる場合がある。

3. 各コアは、命令の先読みをしないため、条件分岐やループなどの命令の実行が CPU より遅くなる。

4. 共有メモリのサイズが 16〜32KB であるため、比較的小さなデータを処理する部分を並列化する必要がある。

5. 各コア及び共有メモリと GPGPU ボードの汎用メモリとの間のメモリアクセスが非常に遅い。

6. GPGPU のコア上で共通して動作するコード（カーネルと呼ばれる）を生成する CUDA コンパイラの最適化は CPU 用のコードよりも劣る。

7. GPGPU の浮動小数点演算は単精度 (float) である。倍精度 (double) も使えるが、単精度より計算が大幅に遅くなる。

4 GPGPU による文書の分類

本節では、SVM による文書の分類において、GPGPU を用いる方法について述べる。経験的に文書分類に適したカーネルとして、本論文では線形カーネルを用いる。このため、文書の分類の主要な処理は、素性ベクトルと重みベクトルの内積 (dot product) を計算する部分となる。以下、従来の内積計算法、及び GPGPU に特化した内積計算法を与える。

Sparse Vector による方法　まず始めに従来手法 [5] を紹介する。各文書は、素性抽出により素性ベクトルに変換される。素性には単語の N グラムを含むため、文書分類における素性ベクトルは数万〜数十万次元のベクトルとなるが、非ゼロの要素が数百〜数千程度であり、非常にスパースな素性ベクトルとなる。

そこで、素性ベクトルの実装に Sparse Vector 表現が用いられる。まず、素性ベクトル $\boldsymbol{x}^\ell$ に対し、ベクトルの要素の添字を格納する配列と値を格納する配列の 2 つの用意する。ここれではそれぞれ $x^\ell_{index}[\ ]$ と $x^\ell_{value}[\ ]$ とする。素性ベクトル $\boldsymbol{x}^\ell = (v_1, v_2, ..., v_n)$ から、非ゼロ要素 v_i を全て取

り出し、添字の順番に整列させる。整列順 k 番目の v_j を次のように 2 つ配列を用いて表現する。

$$x^{\ell}_{index}[k] = j, x^{\ell}_{value}[k] = v_j$$

2 つの sparse vector $\boldsymbol{x}^1$ と $\boldsymbol{x}^2$ 間の内積の計算は、以下のような処理により $O(x^1_{index}$のサイズ$+$ x^2_{index}のサイズ$)$ で計算できる。$i = j = 0$ から i、j または i と j をインクリメントしながら、添字の値 $x^1_{index}[i]$ と $x^2_{index}[j]$ を比較していき、$x^1_{index}[i] = x^2_{index}[j]$ の場合に、$x^1_{value}[i] * x^2_{value}[j]$ を内積の累計に足し込んでいけば良い。

配列と Sparse Vector による方法　上記の従来方法は逐次処理において計算効率が良いが、順序的な処理であるため並列計算には不向きである。線型 SVM による文書の分類は、基本的に重みベクトルと文書の素性ベクトルとの内積計算となることを利用して、以下のような効率化が可能である。

重みベクトルはサポートベクトル $\boldsymbol{x}^i$ とそのラベル y^i、学習時に推定されたラクランジュ乗数 α_i から以下のように求められる。

$$\boldsymbol{w} = \sum_{i \in SV} \alpha_i y^i \boldsymbol{x}^i$$

このとき文書分類では、$\boldsymbol{w}$ はスパースなベクトルではなく数百万次元の多くに値が入ったベクトルとなる。そのため、先のような内積の計算をすると百万回以上のループを回ることになる。GPGPU コアはメモリアクセスが非常に遅く、クロック数も低いことから、並列計算による効率化よりも各 GPU コアの速度低下の影響の方が大きくなる。

そこで、GPGPU の特性に合わせて、重みベクトルは通常の配列に格納し、文書から生成された素性ベクトル $\boldsymbol{x}^{\ell}$ は Sparse Vector 形式で格納することにする。このようにすれば、重みベクトルの配列の $x^{\ell}_{index}[k]$ 番目の要素の値を順に取り出して、内積の累計の変数に足し込む操作を行なえば良い。その回数は、$\boldsymbol{x}^{\ell}$ の非ゼロ要素の個数回（約 1000 回程度）に抑えられる。重みベクトルの配列は GPU 汎用メモリに載せるので、配列のサイズは 1GB 以下である必要があるが、文書分類データでは、素性空間の次元は数百万次元であり、重みベクトルは各カテゴリについて数十 MB 程度に収まるので問題ない。一方、SVM 学習時の内積計算には多数の素性ベクトル間の内積の計算をする必要があるため、訓練データが数万件ある今回の場合には、学習時のベクトルの表現方法として通常の配列による方法は適さない。

5　実験と評価結果

文献 [6] の通り、CTP コーパスから “Brief Title”, および “Official Title”, “Brief Summary”, “Detailed Description” の 4 つの記述項目に現れる単語（unigram）を素性として用いた。単語はすべて小文字による表現に変換した。学習アルゴリズムには SVM^{perf} [2] を用い、SVM^{perf} を使って学習されたモデルを用いて測定をおこなった。10 分割交差検定における 1 つのラウンドにおける、テストデータ 8,523 文書を分類する速度を測定した。

本論文では、CPU として Core i7 (3.3GHz)、GPGPU として、GeForce GT230 (1GB メモリ、96 CUDA Core) を用いて実験を行なった。計算精度は単精度とした。

実験結果を表 5 に示す。なお、素性抽出の時間はどの手法でも同じであるため含まない。SVM^{perf} を用いて 8,523 データを 63 カテゴリに分類した場合は 319 秒であった (a)。Sparse Vector による逐次的な効率化は、CUDA 環境には向かないため、CUDA 環境では、Sparse Vector の 63 並列による内積計算に 1,080 秒を要した (b)。一方、Sparse Vector による内積計算を独自に実装し、CPU 側で実行すると 255 秒に短縮された (c)。

ここで、前節の配列と Sparse Vector による方法により CUDA 環境において 63 並列処理により内積の計算を行なうと、21 秒ですべての分類が完了し、10 倍以上の高速化を達成した (d)。しかしながら、CUDA 向けの高速化は CPU にとっても効率が良い処理であることから、CPU 側で測定したところ 15 秒で分類が完了した (e)。ただし、カテゴリ数の増加につれ、CPU による処理は線形に所要時間が伸びていく。一方、コア数の多い高機能な GPGPU の導入すれば、カテゴリ数が増加してもコア数の範囲であれば、分類時間が 7 秒と不変であるため、本実験により GPGPU の利用の効果が裏付けられたと言える。

表 2: 63 カテゴリへの分類実験結果

手法	分類時間	モデル読込	CUDA オーバヘッド	所要時間（全体）
(a) SVM-perf	-	-	-	319 sec
(b) Sparse Vector (CUDA)	1,066 sec	10 sec	4 sec	1,080 sec
(c) Sparse Vector (CPU)	245 sec	10 sec	-	255 sec
(d) 配列& Sparse Vector (CUDA)	7 sec	10 sec	4 sec	21 sec
(e) 配列& Sparse Vector (CPU)	5 sec	10 sec	-	15 sec

6 GPGPU による文書分類の学習

本研究では、データの並列ではなくカテゴリの並列による高速化をテーマにしている。学習のフェーズにおいても、数万カテゴリへの分類モデルを効率的に学習する方法についても検討を進めている。

すでに、CUDA 環境で動作する SVM は cuSVM [1] と multisvm [4] が存在する。今後これらの実装法を参考にしながら、以下のアプローチで大規模カテゴリへの分類の学習アルゴリズムを実装していく。ポイントは、大規模カテゴリの分類学習において、同じ訓練データを共有しながら数万のカテゴリを平行して学習ができれば、数万回独立に学習するよりも効率的である。特に、各データ間の内積の計算を 1 度行い、数万カテゴリの学習ではその内積を共有しながら並列に学習することができる。

7 まとめ

GPGPU を用いて、文書分類を高速化する方法について述べた。臨床試験計画書の MeSH カテゴリーへの自動分類に関する CTP コーパスを用いて、カテゴリ階層のトップレベル階層に位置する 63 カテゴリへの分類実験を通して、SVM による文書分類時における GPGPU による高速化について計測した。従来、8,523 文書の 63 カテゴリへの分類に 319 秒を要していたが、GPGPU により 21 秒に短縮することができた。カテゴリ数が GPCPU のコア数の範囲であれば、カテゴリ数が増加しても分類時間は不変である。

今後は、GPGPU による、より大規模なクラスについての評価を行なうことが課題である。また、分類側だけでなく、学習側での GPGPU の利用にして検討及び実装も課題である。さらに、大量のカテゴリの分類において、カテゴリが階層化されている場合に、階層情報を使うことのメリットも興味深い問題である。

参考文献

[1] http://patternsonascreen.net/cuSVM.html

[2] Thorsten Joachims, A Support Vector Method for Multivariate Performance Measures, *Proc. of the International Conference on Machine Learning (ICML-05)*, pp. 377–384, 2005.

[3] Medical Subject Headings, http://www.nlm.nih.gov/mesh/

[4] http://code.google.com/p/multisvm/

[5] John Platt, 1998. Fast Training of Support Vector Machines using Sequential Minimal Optimization,

[6] 佐々木裕, 臨床試験計画書の MeSH カテゴリへの自動分類, 言語処理学会年次大会, 2010.

[7] Jason Weston and Chris Watkins, Support Vector Machines for Multiclass Pattern Recognition, *Proc. of the Seventh European Symposium on Artificial Neural Networks*, Brussels, pp. 219-224, 1999.