GPGPUによるSVM分類器の高速化

Accelerating SVM Classifiers with GPGPUs

佐々木 裕　　長谷川 雄
Yutaka Sasaki　Yu Hasegawa
豊田工業大学
Toyota Technological Institute

1 Introduction

In practical applications of text classification, we frequently face a computational difficulties caused by huge data. Suppose that we need to classify millions of documents into tens of thousands of classes. We need to train tens of thousands of models based on millions of data samples when we take the one-against-the-rest binary classification approach. Not only training but also classification (*i.e.*, test) phase also becomes computationally challenging due to the overwhelming number of models.

NLM-NIH's MeSH (Medical Subject Headings) is a standard medical thesaurus that consists of more than 110,000 biomedical categories. MeSH has been used to classify more than 18 million biomedical literature. In this respect, when we need to classify biomedical documents into MeSH categories, it is inevitable to consider some kinds of effective accelerations in the training and classification phrases in the ML-based text classification. In this study, as the first step, we focus on acceleration of the classification phase.

Recently, GPGPUs (General-Purpose Graphics Processing Units) have been attracting strong attention in the massively-parallel computation field. Originally, GPUs were developed for image processing. Now, GPUs have been extended to be capable of general-purpose computations.

In this study, we target GPGPU-based accelerations of SVM document classification. Target data are produced from clinical trial protocols with 11,443 distinct MeSH categories. In this study, only 63 top-level categories were considered.

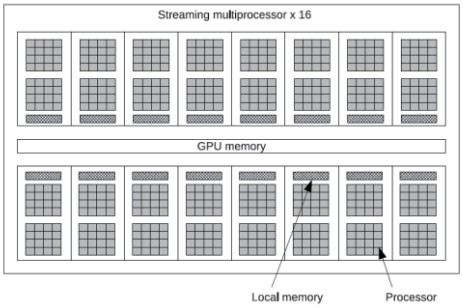

Figure 1 GPGPU architecture

2. Overview of GPGPU

We adopt NVIDIA's Tesla C2050 GPGPUs and a software development environment, NVIDIA's CUDA (Compute Unified Device Architecture). Figure 1 shows the architecture of GPGPUs. There are many Streaming Multiprocessors (SMs) inside of a GPGPU. Among them, there are the smallest processing units called CUDA cores. The number of cores is different GPGPU to GPGPU. Tesla C2050 incorporates with 32 cores for each of the 14 SMs, which can operate the total of 448 cores in parallel. The CUDA core clock itself is 1.47GHz, which is slower than state-of-the-art CPUs.

A GPGPU processor can handle the larger number of threads than the total number of its cores. Tesla C2050 can process up to 1024 threads simultaneously. The thread has been managed in a hierarchy where a grid is on the top of the hierarchy. The grid consists of blocks and the block is comprised with threads. The shared memory is allocated to each block. Tesla C2050's shared memory capacity is as small as

48 KB and it can be accessed for fast sharing data among threads in the same block. Access speed can be expected to be faster by utilizing GPGPUs.

Each thread is managed in the execution unit called *warp*. In CUDA, 32 warps are allowed for a thread. When a branch occurs, it slowdowns according to the number of warps. So, it is necessary to avoid divergent branches.

3. CTP corpus

3.1 Overview

We have created the Clinical Trial Protocol (CTP) corpus. [1] The overview of the corpus is as follows:

- # docs: 82,525
- Corpus size: 529 MB
- Ave. doc size: 6.4 KB
- # distinct MeSH categories: 11,443
- # top-level distinct categories: 63
- Ave. # categories/doc: 25.9
- Ave. # top-level categories/doc: 3.4

The corpus has been constructed in the following steps:

1. Downloaded all the clinical trial protocols in HTML from the clinicaltrials.gov (as of Dec. 11, 2009).[a]
2. At the same time, the MeSH category list for each protocol was downloaded in HTML.
3. The downloaded protocols and MeSH categories are combined into XML files.

Unfortunately, there are no common agreements in the format of clinical trial protocols. Therefore, we extracted unigram features form the following sections, regarding them as essential items in the clinical protocol.

- NCT ID: ID
- Brief Title: general title
- Official title: technical title
- Brief summary (Purpose): overview of the clinical trial
- Detailed description: detailed description of the clinical trial.
- Additional relevant MeSH terms: MeSH category list

3.2 MeSH Category

This section describes an overview of MeSH category. MeSH is a standard biomedical thesaurus defined by the National Library of Medicine at the National Institutes of Health. A unique MeSH Tree Number. is assigned to each MeSH category. The MeSH Tree Number represents a path from the root category to the category. For example, Human Influenza (C02.782.620.365) is categorized in the following path.

- Virus Diseases [C02]
  - RNA Virus Infections [C02.782]
    - Orthomyxoviridae Infections [C02.782.620]
      - Influenza, Human [C02.782.620.365]

Clinical trial protocols in clinicaltrials.gov are associated with MeSH categories in a inconsistent manner. Some include redundant categories, i.e., middle and leaf nodes in a path. Therefore, in this corpus, we expanded all the intermediate MeSH categories based on MeSH Tree Numbers.

4. SVM classification using GPGPUs

4.1 Dot product with the sparse index

Traditionally, sparse vector representation [2] is used to efficiently compute the dot product of sparse vectors. However, its sequential computation of the dot product is not suitable for parallelization. Then, we took a new approach called the sparse index approach [3].

In classification of a text, typically a weight vector is not sparse. Only an input vector is sparse. Therefore, we represent an input vector in the sparse vector format and the weight vector as a normal vector. This means that the sparse vector works as an index to retrieve elements in the normal weight vector. This drastically reduces the number of iterations that

[a] The site supports the XML format for downloading clinical trial protocols. However, sentences are segmented in the middle of the sentences by the line break in the XML. HTML pages do not include such a problem.

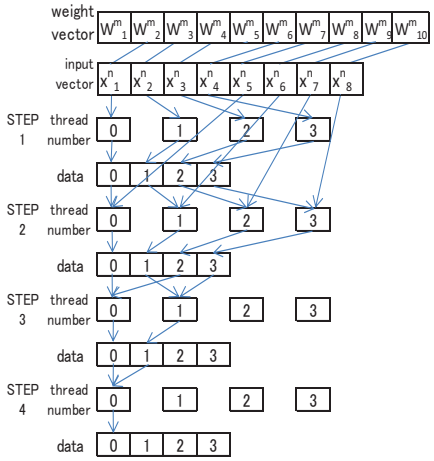

Figure 2 Parallel processing of a dot product computation

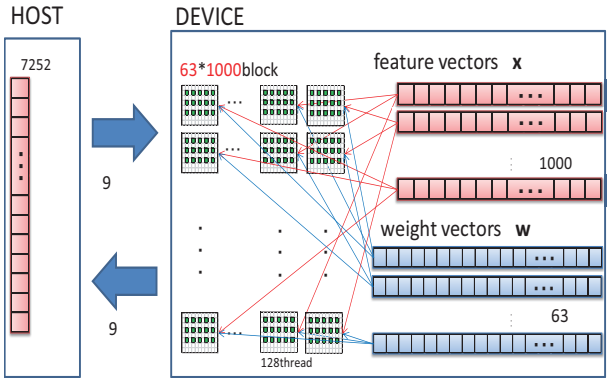

Figure 3 Category and data parallelization for a dot product computation

needs to compute a dot product. That is, the number of loop is limited to the number of elements in a sparse vector.

Moreover, the sparse index can be processed in parallel (details are described in the following section).

4.2 Feature vectors

In this study, we use feature vectors created from the CTP corpus. The number of documents to be classified is 8,252. The SVM classification model is generated from the rest of the data from the CTP corpus. Feature values represent the presence or absence of a word occurrence. The value is 0 if it is absent and 1 if appears.

Instead of directly computing the dot product, we can compute the sum of the weight vector elements whose corresponding feature value in a given input vector is 1.

As we use a linear kernel SVM, computing the dot product of feature vectors and weight vectors is the point that must be accelerated. In this study, to parallelize the computation of the dot product between the weight vector and input feature vector, the input vector is represented in the sparse vector and the weight vectors in a normal vector as described in the previous section.

4.3 Parallel implementation

The *dot product calculation parallelism* is employed using 128 threads. The dimensionality of the input vector is up to 11,496. The block diagram of a sample execution is shown in Figure 2. Due to space limitations, Figure 2 shows a parallel reduction using four threads but it can be scaled up to 128 threads. First, the weight vector elements corresponding to the input sparse vector index are copied from the GPGPU global memory to the shared memory with 128 parallelisms (Fig. 2, Step 1). Then, the parallel computation of the dot product is done using 128 threads (Fig. 2, Step 2). Then we calculate the sum of the elements locally copied from the weight vector using 128 threads in a tree format (Fig.2 Steps 3, 4) using a reduction method[1]. A weight vector may be less than the dimensionality of 128. In that case, 0s are filled with so as to avoid warp divergence caused by branching.

We also calculate the dot product with 63 *category parallelisms* for a single feature vector that is to calculate the dot product of an input vector and each of the 63 weight vectors at a time (corresponding to the number of categories). The overhead would still exist in data transfer in every DMA. Using the *data parallelism*, the optimization of data transfer can be achieved using the 63*1000 block for the dot products of 1000 feature vectors and 63 weight vectors at once, logically (Figure 3).

5. Experimental results

Table 1 shows the results. Thanks to data and dot product calculation parallelization, Table 1 (d) becomes 40 times faster in the speed of

Table 1 Experimental results

method	classification time	input vector loading time	weight vector loading time	GPGPU initialization	total time
(a) SVM-perf	–	–	–	–	319
(b) sparse index with 63 category parallelization (GPGPU)	5.59	0.75	0.96	4.36	11.66
(c) sparse index (CPU)	4.85	0.75	0.96	–	6.56
(d) sparse index with 63 category, dot product, and data parallelization (GPGPU)	0.14	0.75	0.96	4.36	6.21

(sec)

classification, compared to Table 1 (b).

Figure 4 shows the processing time along with the number of categories. The category, data, and dot product parallel approach to SVM classification (d) is much faster than other methods, especially when the number of categories is large enough. As the number of categories increases, there is no significant change in processing time unless they have enough memory and cores.

6. Conclusions

In this study, we successfully accelerated the SVM document classification with a GPGPU. Based on the sparse index method, category, dot product calculation, and data parallelisms are the most effect way to accelerate SVM classification. Parallelize the SVM training phase using GPGPUs is our future work.

References

[1] NVIDIA Corporation : Optimizing Parallel Reduction in CUDA, 2008. (http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf)

[2] John C. Platt: Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, Technical Report MSR-TR-98-14, 1998.

[3] Yutaka Sasaki, Massively Parallel Computing with GPGPUs for Text Classification: A Case study, Proc. of the Annual Meeting of Association for Natural Language Processing, 2011.

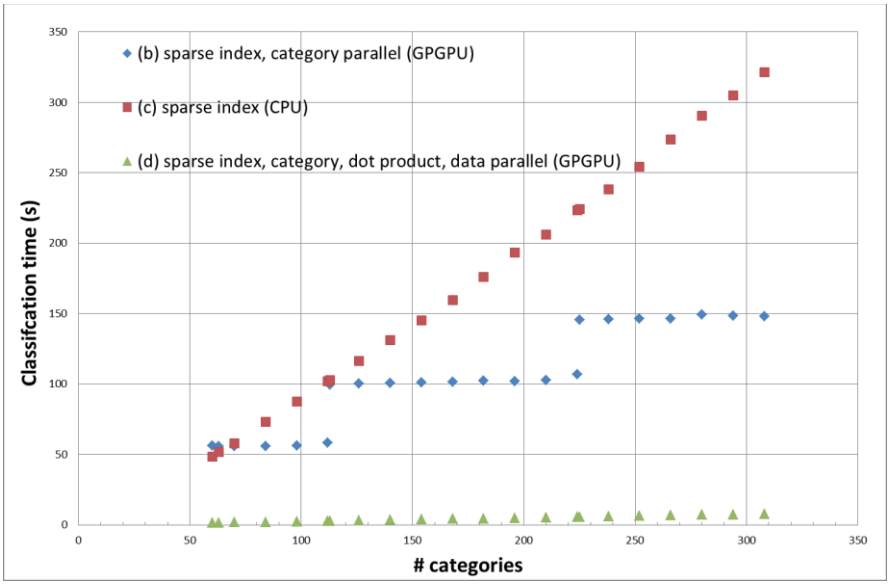

Figure 4 The number of categories vs. classification time