A GPU-Based Accelerator for Chinese Word Segmentation

Xiwu Gu[1,*], Ruixuan Li[1], Kunmei Wen[1], Bei Peng[1], and Weijun Xiao[2]

[1] Intelligent and Distributed Computing Lab, College of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan 430074, P.R. China
{guxiwu,rxli,kmwen}@mail.hust.edu.cn, beipeng.peng@gmail.com
[2] Department of Electrical and Computer Engineering, University of Minnesota, twin cities
200 Union Street SE, Minneapolis, MN 55455, US
wxiao@umn.edu

Abstract. The task of Chinese word segmentation is to split sequence of Chinese characters into tokens so that the Chinese information can be more easily retrieved by web search engine. Due to the dramatic increase in the amount of Chinese literature in recent years, it becomes a big challenge for web search engines to analyze massive Chinese information in time. In this paper, we investigate a new approach to high-performance Chinese information processing. We propose a CPU-GPU collaboration model for Chinese word segmentation. In our novel model, a dictionary-based word segmentation approach is proposed to fit GPU architecture. Three basic word segmentation algorithms are applied to evaluate the performance of this model. In addition, we present several optimization strategies to fully exploit the potential computing power of GPU. Our experimental results show that our model can achieve significant performance speedups up to 3-fold compared with the implementations on CPU.

Keywords: Word Segmentation, GPU, CUDA.

1 Introduction

Chinese word segmentation is always the first task for Chinese information processing, and it is widely used in information extraction and information retrieval domain. As most web search engines locate the documents by keywords, the accuracy of word segmentation has a considerable effect on the quality of search results [1]. The objective of Chinese word segmentation aims to improve accuracy. However, Due to the explosive growth of Chinese literature online, Chinese information that needs to be processed by search engines becomes huge. The efficiency of segmentation thus becomes a prime factor affecting the performance of search engines [2]. Therefore, as an important part of information retrieval, word segmentation is expected to be more effective. To provide high throughput, search

* Corresponding author.

Q.Z. Sheng et al. (Eds.): APWeb 2012, LNCS 7235, pp. 231–242, 2012.

engines have to employ more CPUs to accelerate the processing of text analysis, index and search. In a distributed environment, the analysis tasks of Chinese texts are simultaneously launched on multiple CPU cores, and each core performs repetitive segmentation with different data set. As a result, the response time for processing massive data set is largely reduced. Though it is a straightforward solution to improve the performance of Chinese information processing by leveraging more computation resources, it will face a problem of increasing computing cost.

In this paper, we investigate a novel approach to enhancing the efficiency of Chinese word segmentation by using Graphic Processing Units (GPU). Generally, word segmentation is cutting a sentence into separate terms based on punctuation or similar rules. Each text is analyzed in the same way, so word segmentation can be regarded as a process sharing the instructions but with different data, which coincides with computing pattern of GPUs. Furthermore, segmentation on a large scale of texts will need huge repeated computation. Consequently, Chinese word segmentation becomes a potential application for GPU. GPU is designed for graphic rendering, and now it can perform a much higher degree of parallelism than CPUs, which offers many opportunities to exploit data parallelism. In recent years, lots of studies have attempted to use GPUs to accelerate the non-graphical applications, including scientific computing, analog emulation, and databases [3, 4, 5].

In our research, we introduce GPU to solve the performance bottle-neck problem of Chinese information processing for a large amount of data. We develop a CPU-GPU collaboration framework to analyze Chinese texts in parallel. Though GPU has provided a high-level programming interface, it is still a challenge task for developers to build a highly parallel data computation model. In addition, the utilization of streaming processors is another issue that will greatly affect the execution efficiency of GPU. In our framework, the GPU plays the role of a cooperator performing intensive, highly parallel computations rather than a rival to the CPU which is just responsible for scheduling and tasks distribution. By adding data buffer, the execution of word segmentation is launched asynchronously with text stream portability in a pipeline to overlap the I/O cost. In order to better suit to the GPU architecture, we design a parallel algorithm for the dictionary-based Chinese segmentation and optimize the instructions to delay memory access as much as possible. Furthermore, we design a cluster of data structures for GPUs and adopt several measures to cache the key data that is frequently read. Then, three Chinese word segmentation algorithms based on dictionary are performed on GPU to test the performance. We mainly evaluate the performance of this architecture in two aspects. First, we carry out a set of experiments to obtain the throughput in different situations, including parallel granularity and text size. The time cost in each phase of the task is also recorded. Then, our approach for parallel word segmentation is benchmarked against CPU, and several optimizations are carried out to improve the efficiency. The experiments indicate that our optimization strategy can well serve this new architecture, and GPU has advantages over CPU in data intensive computations.

The rest of the paper is organized as follows. We first review the related literatures of Chinese word segmentation and the studies of GPUs in different areas. Then, we introduce the CPU-GPU collaboration architecture and describe the implementation

of Chinese word segmentation on GPU. Moreover, some performance optimization technologies are integrated into this architecture for further acceleration. The experimental results are subsequently presented by the measurements of throughput and response time, and the paper concludes with a summary of parallel Chinese word segmentation on GPU.

2 Background and Related Work

2.1 Chinese Word Segmentation

Word segmentation is one of the most commonly researched areas in natural language processing (NLP), and it is always the basic work for information retrieval (IR). The texts should undergo a segmentation process to be broken up into smaller linguistic units before the words can be used to create index, only then texts can be searched by keywords. Therefore, word segmentation is such an important part that it has a great impact on the effectiveness of IR [1]. For languages like English, words are usually separated by space, and this explicit boundary is sufficient to divide a string of English into component words. However, some ideographic languages like Chinese do not mark a word in such a fashion, and it is a challenging task for computers to extract semantic words from a sentence automatically.

Chinese word segmentation has always been a very important research topic in Chinese text processing. Apart from Chinese information retrieval, it also plays a great role in other domains, such as Chinese input, speech recognition, machine translation and language understanding. In the past few years, most studies have looked into Chinese word segmentation for a better precision [6, 7]. The basic approaches involved in researches are mainly word-based, and they can roughly fall into two categories, namely the dictionary-based approach and the statistic-based approach. Most of the earlier work on Chinese word segmentation has concentrated on approach based on dictionary [8, 9], and it is commonly used in most current systems for coarse segmentation due to its simplicity and efficiency. For the dictionary-based approach, one of the most important components should be the dictionary which stores all possible words. When segmented, the chunk of strings can match against the dictionary to identify the words or phrases. However, when the dictionary becomes larger, it is difficult for CPU to locate a full word in serial mode. Therefore, the efficiency of searching dictionary turns to be a crucial aspect that influences the performance of word segmentation. To achieve quick prefix match, Fredkin [10] designed a digital search tree called Trie, whose time complexity is comparable to hash techniques. Aoe [11] simplified this prefix match tree into two parallel arrays for better space usage. The work in [12] introduced this structure to represent the dictionary for a high efficiency. The work in [13] designed a new structure to reduce space usage of the arrays. Nowadays, as the text set becomes larger, it is also a time-consuming job to process massive text data. So it is necessary to analyze the text sets in parallel besides improving dictionary structure. Multi-core device may be a good choice to improve the performance of word segmentation.

2.2 GPU

The graphic processing unit (GPU) is a dedicated device originally designed for rendering graphics. Compared to CPUs, modern GPUs are more efficient in manipulating computer graphics as a result of their highly parallel structure. Though GPU is a specialized circuit used for graphics processing, its high computing capability offers an opportunity to accelerate calculations for general-purpose computing. It is expected that GPUs can obtain a higher performance than CPUs when large blocks of data are processed in parallel. In the past few years, many engineers and scientists have increasingly exploited GPUs for non-graphical calculations, such as scientific computing and database [2, 3, 14]. Since NVIDIA launched the compute unified device architecture (CUDA), GPU becomes more programmable. In CUDA programming model, developers can transparently leverage the parallel engines on GPU with standard programming languages. Furthermore, data parallelism and task parallelism can be easily carried out with the help of thread groups. The job is partitioned into sub-tasks that are solved in parallel independently by blocks of threads, and then each sub-task is divided into finer pieces that are solved cooperatively by threads within the block [15]. Indeed, each block of threads is executed across the available processor cores concurrently or sequentially, and the kernel can be executed on any number of processor cores. So developers can just focus on the data flow rather than the traditional programming paradigm. Recently, the performance of modern GPU has increased at such a rapid pace that it is capable of outperforming current CPUs in some computation-intensive applications. More researchers have been exploiting GPU through CUDA for general-purpose computing [16, 17]. Several studies have been carried out to exploit GPUs to accelerate the computations involved in IR. The work in [18] addresses a problem of using GPUs to improve the efficiency of measuring document similarity. The work in [19] has implemented high-performance IR query processing on GPUs. They developed a set of parallel algorithms for index compression, index intersection, and ranking.

In the IR domain, indexing and searching always remain the focus as they are both the central parts for search engines. And recent works rarely present the issues on word segmentation because it seems to be in a secondary position. However, it is another case when search engines involve massive text processing. At that time, word segmentation will be a time-consuming process, and it will impose a big burden for CPUs. So it is desirable to use GPU to share the workload for search engines.

3 Implementation

GPU is specialized for intensive computations, so it can well address the problems that can be expressed as data-parallel. Large sets of data elements can be mapped to parallel threads, and each thread is executed for many data elements that have high arithmetic intensity. As a result, there is a lower requirement for sophisticated flow control. Many studies in non-graphic fields have adopted this data-parallel pattern to achieve better performance, such as physics simulation and computational finance. Word segmentation is also the one that fits the pattern. In general, the content that needs tokenizing is likely

to be a sentence, a paragraph or a piece of writing, and these individuals are almost isolated from each other anyway. By mapping independent text data to different threads, GPU can simultaneously launch all threads to start analyzing in parallel. However, it is not an easy task to perform parallel Chinese word segmentation on GPU. It requires frequent dictionary searching and big data transmission when the task is performed on GPU. All these movements are bound to generate large amounts of memory access operations. So, more optimizations are needed to make progress on GPU. Furthermore, many factors should be taken into account as a result of the limited resources on board. In this paper, we aim to solve these problems by designing the collaboration scheme and parallelism scheme. The collaboration scheme mainly coordinates the work between CPU and GPU, and further optimizes the data transmission. Parallelism scheme involves algorithm optimization to increase resources utilization. Both schemes play an important role in improving the overall performance of parallel word segmentation on GPU.

3.1 CPU-GPU Collaboration Architecture

The multi-core architecture of GPUs leads to a higher parallel computation capability than CPUs. However, the performance improvement would not be significant unless extra overhead is hidden. In this paper, we design CPU-GPU collaboration architecture to reduce the overhead and to make the best of GPU resource.

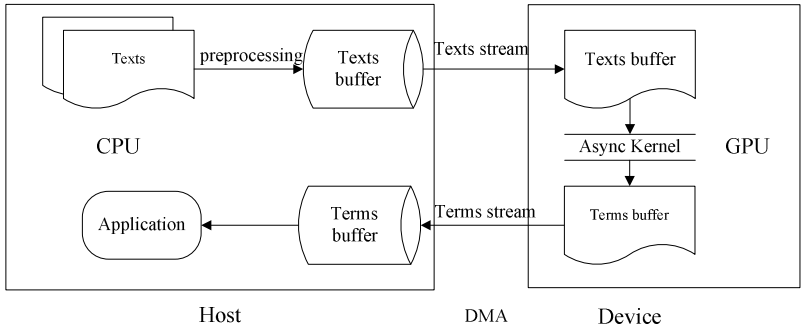


Fig. 1. CPU-GPU collaboration workflow

As shown in Fig. 1, the CPU is responsible for task scheduling while the calculation for word segmentation is offloaded to GPU. The task is carried out in batches which contain an amount of texts. First of all, raw texts are read from disk in batches incessantly and make a short stop in main memory. Before pushing into global memory, the batches should go through a pretreatment for load balance. On the host side, there is a text filter to convert Chinese texts to an appropriate format for GPU, and with the help of semantic separators, texts in a batch are divided into fragments with a full sense, so the workload can be controlled in fragments. Then the texts are pushed to texts buffer which maintains the texts to be analyzed by GPU. Once the device runs into idle time, the texts are immediately taken out from the texts buffer and transferred to global memory. In this case, CPU will be stuck at data

transmission phase in this case. For a better utilization of CPU, we asynchronously transfer data with the help of DMA. Furthermore, the data transmission task and calculation task are associated to work in a pipeline. As a result, streaming processors can keep running most of the time, and memory access could overlap parallel computation.

When the texts data is in place, each thread block is assigned one or more texts. To improve stream processor utilization, more tasks are distributed on GPU than there are enough cores. Then multiple threads on device side are launched concurrently to perform each task, and the terms extracted from the texts are pushed into terms buffer in a specified format. Finally, all terms are submitted in batches to host side asynchronously. In this cooperation model, GPU can continuously read data from texts buffer and put the results into the terms buffer without interference. In this case, streaming processors are always at full load, and some of the disk I/O time is hidden. Moreover, CPU could be extricated from the substantially repeatable computation with the assistance of GPU. The task for CPU is so limited that it can devote more to other applications. In a certain respect, this collaboration architecture can make better use of CPU and GPU to perform complex computations.

3.2 Parallel Segmentation on the GPU

Word segmentation based on dictionary is a typical task performed in serial. These operations during the segmentation procedure are bound to be closely linked with each other, so a finer-grained parallelism for dictionary-based segmentation is burdensome. Alternatively, we put the emphasis on data parallelism for a higher throughput. It has been proven that data parallelism can achieve a good performance on GPU. So data parallelism is supposed to be a promising way to increase efficiency of word segmentation. Though it seems a trivial task to achieve this kind of parallelism, it also takes a lot of efforts to perform an effective segmentation algorithm on GPU. For dictionary-based approach, dictionary usually contains lots of entries, which can occupy much space. Since the memory on GPU is limited, dictionary structure is of considerable importance. Besides, when the dictionary is extended, it will also cost a little more to find the right entry especially in global memory. So, parallel segmentation optimization is another important topic.

Dictionary construction faces a difficult choice. If a dictionary is small, many words will be missed during segmentation. Once the dictionary grows big, it will take up much valuable space. In our experiment, we select nearly 160 thousand Chinese words in common usage to make up the entries. In the dictionary, the percentage of two-word terms reaches half of the volume. Three-word terms and four-word terms share the rest with several terms exceeding ten words. Each character is hashed to a new value within two bytes so that dictionary space can be cut down. In general, linked list structure is flexible for dictionary update, but repeated node query in global memory may cause large latency due to the alignment problem. To better fit GPU architecture, all lexical information is compressed into two parallel arrays called Double Array [11]. As a result, the dictionary only takes up 3MB memory space and space utilization has exceeded 70%. Furthermore, we propose a state array to perform segmentation for different

demands in great ease. The extra array contains some auxiliary information such as transition information and plays a guidance role in word segmentation. Since the static dictionary no longer provides any update feature, we keep all the vocabulary information in files on host side. Then it is mapped to global memory when segmentation is required on device side. All dictionary data in global memory are aligned by 256 bytes in order that memory accesses within a warp can coalesce to one memory transaction. Even so, dictionary access is still a time-consuming task because the data for different threads are usually discontinuous in physical structure. To ease the pressure of memory access, we make use of the texture memory for data cache. In this case, some of the vocabulary information is located in texture memory. So it is unnecessary to access global memory every time when matching the dictionary. For segmentation in some specified field, it is likely to put this tiny dictionary in shared memory, which can largely reduce the memory access time.

The task of parallel word segmentation on GPU is illustrated in Fig. 2. The texts are first assigned to blocks, and then each block distributes the calculations across threads. When all texts data for each block is in place, all threads on device side are lunched concurrently to perform text analyzing. Each thread independently checks the character against dictionary one by one. When the transition information is not hit in texture memory, it then goes forward to the global memory. Once a term occurs, the thread put it into the terms buffer owned by each block. Since there is no synchronization overhead among threads in a block, this data-parallel pattern can achieve a good performance of word segmentation on GPU.

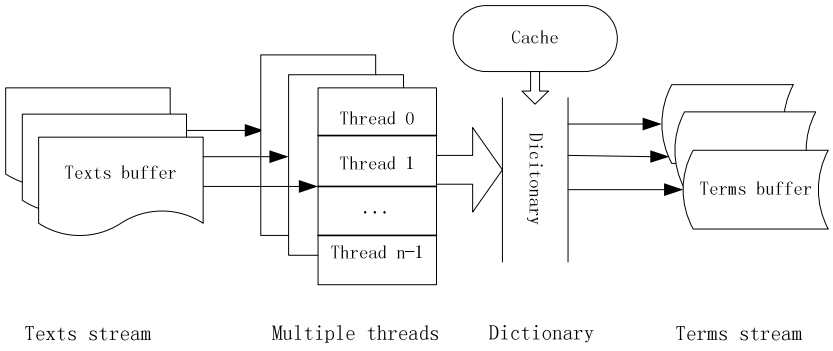


Fig. 2. Parallel word segmentation model

When the segmentation task is complete, another kernel is launched to count the frequency of each term. In order to keep load balancing, each block receives a block of terms and assigns a sub-block of equal size to each thread. Then each thread compares the terms to the current global term in sequence and changes the frequency value in frequency table. When all threads in a block complete the comparisons, the cursor of global terms moves to the next. Though this approach can achieve frequency counting for each term, it is inefficient on GPU as the threads should be synchronized in a block. Moreover, since the terms obtained by the segmentation kernel may be repeated in the memory, it will take extra time to remove the duplicate terms.

Therefore, we propose another approach to fulfill these two tasks. In this approach, the task of frequency counting is performed before the word segmentation task. The frequency of each term is immediately recorded when the term is extracted from the piece of text. In order to reduce the synchronizing operation among threads, we designed a global hash table for term mapping and a frequency table for each block based on the dictionary structure. Moreover, we use the final transition state of a term to identify the location in frequency table. During the segmentation phase, each thread only changes the frequency value in frequency table. When all threads finish the segmentation task, a frequency table has been already complete for each block. So it is straightforward for each thread to identify the terms that occur in the text by checking the frequency table. Then all threads can get the terms values from the hash table to fill the terms buffer. The improved algorithm executed by each thread is shown in Algorithm 1.

Algorithm 1. Kernel for word segmentation and frequency count

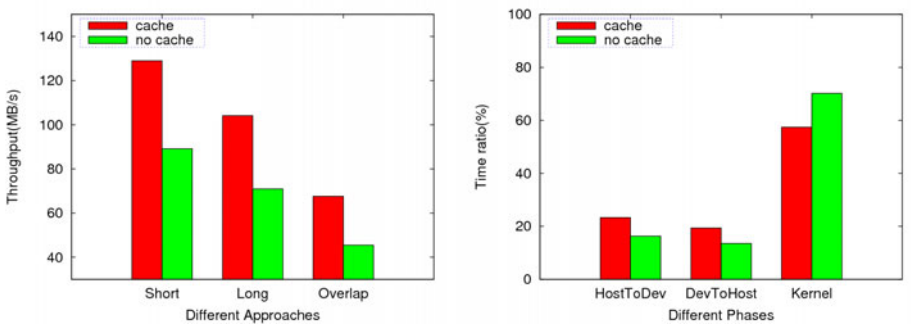
```
   Input: Texts set, Dictionary
   Output: Terms and frequency set
1  while more texts in texts buffer
2    string • get_piece(tid)
3    for each character c in string
4      s • match_dictionary_texture(c)
5      if a term appears then
6        block_freq(s) increases
7      else
8        Move to next character
9      end if
10   end for
11 end while
12 Locate terms in hash_table based on block_freq
13 Write terms to terms buffer
```

In this way, we first count the frequency of each term, and then obtain the terms of each text. As a result, all terms can be easily obtained in one time with the help of the hash table and frequency table, and each term is recorded only once in global memory. Furthermore, it leads to a decrease in the number of memory access as it is unnecessary to access the memory each time a term occurs during the segmentation phase. Moreover, it won't take too much synchronization overhead to perform the frequency statistics task in a block as the task of frequency counting is achieved by changing the frequency table instead of counting the duplicate terms. So it is more efficient for GPU to perform parallel word segmentation and term frequency statistics in this manner.

4 Experimental Results

During the experiment, the results obtained on GPU are benchmarked against CPU, and more experiments are carried out to evaluate the performance of this collaboration model. Our experiments are carried out on windows server 2003 with a

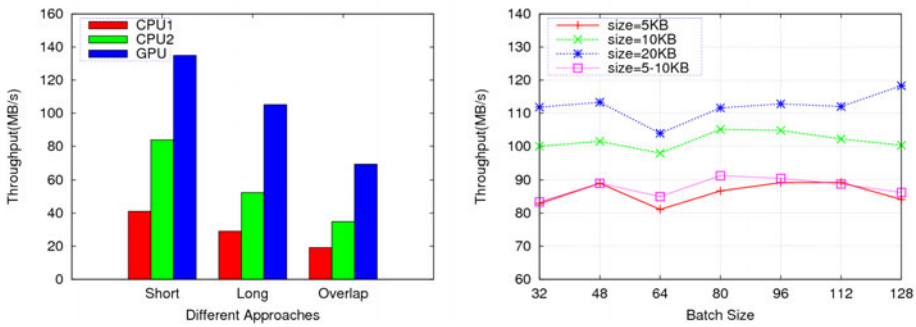
dual-core Intel CPU at a speed of 2.00GHz. The type of GPU is GeForce GTS 250 which has 128 CUDA cores. The Chinese texts are selected from Chinese law and regulation document set.



(a) Throughput of GPU with cache (b) Time ratio in different phases

Fig. 3. Performance of GPU with cache

For word segmentation, dictionary is an important resource that is accessed frequently. As it will suffer a great latency for each thread to access global memory, we propose to put some of the dictionary information into the cache of texture memory. So it is unnecessary for threads to match the dictionary in global memory each time. It is shown in Fig. 3(a) that the throughput is increased greatly with the help of cache. In Fig. 3(b), we can find that the time shared by the kernel is reduced while the time of I/O transmission is relatively increased. It is demonstrated that less time is cost to access global memory during segmentation when the dictionary is cached in texture memory. Therefore, stream processors can perform more tasks per unit time so that the overall time has been reduced.



(a) Throughput of CPU and GPU (b) Throughput in different text sizes

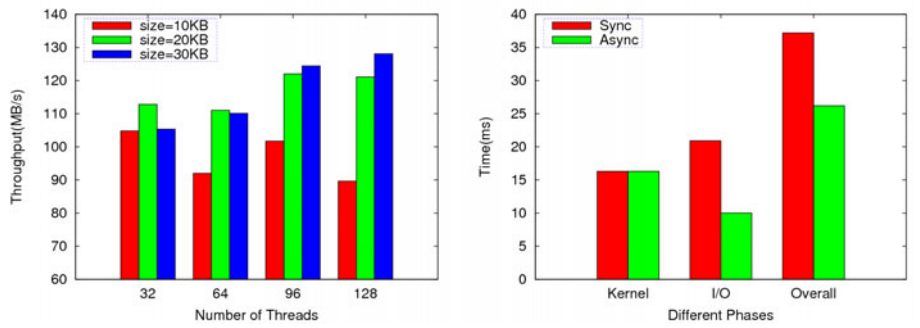
Fig. 4. Performance of GPU in different approaches and text sizes

In order to evaluate the performance of GPU, three Chinese word segmentation approaches are performed on CPU and GPU respectively. It is depicted in Fig. 4(a) that the throughput of GPU is two times more than that of single-core CPU and

almost twice as much as that of dual-core CPU. It can be inferred that GPU can reach the calculation capability of quad-core CPU in parallel segmentation. Moreover, since the shortest match approach usually divides a string into two-word terms, most of the threads can run in the same path with less thread falling into different branches. So the shortest match approach can achieve the maximum throughput while the longest match approach falls behind.

Furthermore, we carry out experiments on longest match approach to study the impact of task granularity on the efficiency of streaming processors. It is depicted in Fig. 4(b) that the throughput becomes higher as the text size of each batch gets larger. So, more calculations can improve the utilization of each stream processor before it is full-load. However, it is not the case when the size of texts in the same batch is different. Though the total size of batch grows the throughput also stays at a low level due to the unbalance load across the stream processors.

Besides load balance, parallelism granularity is an important aspect to improve the performance of GPU. It is depicted in Fig. 5(a) that more threads can better solve the problem of big data processing. So it can achieve a better performance to assign more threads to share the load when the text size grows. However, it will cause a side effect if the text is too small. Further study reveals that it has nothing to do with the workload assigned to each thread. In other words, heavier load for each thread may not necessarily cause higher throughput. It is the scheduling for multiple threads that influences the performance of stream processors. When the text size is rather too small, the overhead for managing additional threads exceeds the cost to perform the task. As a result, the resource on device side can't be fully utilized. However, once the text size gets larger, more threads are needed to share the workload so that stream processors are at full load. So, the throughput eventually grows in this case.



(a) Throughput in different threads counts (b) Time cost in different modes

Fig. 5. Throughput and response time of GPU

Since GPU is a coprocessor, the I/O overhead should be taken into account. More experiments are carried to study the time cost of different phases in parallel segmentation. It is shown in Fig. 5(b) that the I/O overhead contributes significantly to the overall time for a batch in synchronous fashion, and the I/O time unexpectedly exceeds the kernel time, which cannot be endured in efficiency. When the task of word segmentation is performed in asynchronous model, CPU can be free from data

transmission between main memory and global memory with the help of DMA. Meanwhile, stream processors are always running at full load on device side. As a result, the I/O time is cut by half in this case as the kernel execution overlaps the transmission operations.

The performance of GPU is further evaluated when it is used for both word segmentation and frequency statistics. It is shown in Table 1 that the statistics kernel costs most of the time to deal with a batch about 1MB, but the segmentation kernel is finished just in a while. The GPU puts up a poor performance in the task of frequency statistics, so we choose to count the term frequency before recording terms. With the help of a hash table and block wise frequency tables, the efficiency is largely enhanced. It is depicted in Table 1 that the response time of GPU is greatly reduced by combining word segmentation and frequency statistics into one kernel. Since the combined kernel consumes an amount of time, the time for I/O transmission becomes insignificant in this case. It is shown in Table 1 that the time for the task about 1GB in asynchronous mode is less than that of synchronous mode as the time for I/O transmission is well hidden by kernel execution.

Table 1. Execution time of GPU in different phases and modes

Approach	Time(ms)	Execution mode	Time(s)
Segmentation kernel	7.1	Sync mode	126.5
Statistics kernel	491.9	Async mode	118.7
Combined kernel	117.3		

5 Conclusions

In this paper, we present an implementation of parallel Chinese word segmentation using GPU. We design the CPU-GPU collaboration architecture and implement the segmentation algorithm. To exploit the computation capability of GPU, we optimize the task granularity and load balance from different perspectives. Though various inherent branches of segmentation algorithms have influenced the execution efficiency of GPU, we still achieve a 3-fold speed-up. It is well demonstrated that our approach is effective in parallel Chinese word segmentation, and GPU can be exploited to accelerate massive data processing.

Acknowledgments. This work is supported by National Natural Science Foundation of China under Grant 60873225, 60773191, 70771043, National High Technology Research and Development Program of China under Grant 2007AA01Z403.

References

1. Foo, S., Li, H.: Chinese word segmentation and its effect on information retrieval. J. Inf. Process. Manage. 40(1), 161–190 (2004)
2. Thompson, C.J., Hahn, S., Oskin, M.: Using modern graphics architectures for general-purpose computing: a framework and analysis. In: 35th International Symposium on Microarchitecture, pp. 306–317. IEEE Press, New York (2002)

3. Govindaraju, N.K., Lloyd, B., Wang, W., Lin, M.C., Manocha, D.: Fast computation of database operations using graphics processors. In: SIGMOD, pp. 215–226. ACM Press, New York (2004)
4. Preis, T., Virnau, P., Paul, W., Schneider, J.J.: GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model. J. Comput. Phys. 228(12), 4468–4477 (2009)
5. Fan, Z., Qiu, F., Kaufman, A.E., Yoakum-Stover, S.: GPU cluster for high performance computing. In: SC, p. 47 (2004)
6. Gao, J.F., Li, M., Wu, A., Huang, C.N.: Chinese word segmentation and named entity recognition: a pragmatic approach. J. Computational Linguistics 31, 574 (2005)
7. Wang, X.J., Qin, Y., Liu, W.: A search-based Chinese word segmentation method. In: 16th International World Wide Web Conference, pp. 1129–1130. ACM Press, New York (2007)
8. Yang, W., Ren, L.Y., Tang, R.: A dictionary mechanism for Chinese word segmentation based on the finite automata. In: International Conference on Asian Language Processing, pp. 39–42 (2010)
9. Liu, X.G., Luo, J., Xie, Z.: The Research of Chinese Automatic Word Segmentation In Hierarchical Model Dictionary Binary Tree. In: First International Workshop on Database Technology and Applications, pp. 321–324. IEEE Press, New York (2009)
10. Fredkin, E.: Trie memory. J. CACM 3(9), 490–499 (1960)
11. Aoe, J.I.: An efficient digital search algorithm by using a double-array structure. J. IEEE Trans. Software Eng. 15(9), 1066–1077 (1989)
12. Zheng, C., Zheng, Q.H., Zhou, Z., Tian, F.: A method for large cross-language lexicon management based on collaborative work of hash family and double-array trie. In: 14th International Conference on Computer Supported Cooperative Work in Design, pp. 658–663. IEEE Press, New York (2010)
13. Yata, S., Oono, M., Morita, K., Fuketa, M., Sumitomo, T., Aoe, J.I.: A compact static double-array keeping character codes. J. Inf. Process. Manage. 43(1), 237–247 (2007)
14. Krüger, J., Westermann, R.: Linear algebra operators for GPU implementation of numerical algorithms. J.ACM Trans. Graph. 22(3), 908–916 (2003)
15. NVIDIA CUDA Programming Guide (2010), `http://developer.download.nvidia.com/`
16. Ryoo, S., Rodrigues, C.I., Baghsorkhi, S.S., Stone, S.S., Kirk, D.B., Hwu, W.M.W.: Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 73–82. ACM Press, New York (2008)
17. He, B.S., Yang, K., Fang, R., Lu, M., Govindaraju, N.K., Luo, Q., Sander, P.V.: Relational joins on graphics processors. In: SIGMOD, pp. 511–524. ACM Press, New York (2008)
18. Zhang, Y.P., Mueller, F., Cui, X.H., Potok, T.: GPU-accelerated text mining. In: Workshop on Exploiting Parallelism using GPUs and other Hardware-Assisted Methods. ACM Press, New York (2009)
19. Ding, S., He, J.R., Yan, H., Suel, T.: Using graphics processors for high performance IR query processing. In: 18th International World Wide Web Conference, pp. 421–430. ACM Press, New York (2009)