2011 IEEE 17th International Conference on Parallel and Distributed Systems

Fast Snippet Generation Based On CPU-GPU Hybrid System

Ding Liu, Ruixuan Li*, Xiwu Gu, Kunmei Wen, Heng He, Guoqiang Gao
Intelligent and Distributed Computing Laboratory, School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan 430074, Hubei, P. R. China
E-mail: ldhust@smail.hust.edu.cn, {rxli, guxiwu, kmwen}@hust.edu.cn, {henghe, ggq}@smail.hust.edu.cn
* Corresponding author

***Abstract*—As an important part of searching result presentation, query-biased document snippet generation has become a popular method of search engines that makes the result list more informative to users. Generating a single snippet is a lightweight task. However, it will be a heavy workload to generate multiple snippets of multiple documents as the search engines need to process large amount of queries per second, and each result list usually contains several snippets. To deal with this heavy workload, we propose a new high-performance snippet generation approach based on CPU-GPU hybrid system. Our main contribution of this paper is to present a parallel processing stream for large-scale snippet generation tasks using GPU. We adopt a sliding document segmentation method in our approach which costs more computing resources but can avoid the common defect that the high relevant fragment may be cut off. The experimental results show that our approach gains a speedup of nearly 6 times in average process time compared with the baseline approach-Highlighter.**

Keywords: query-biased snippet generation; graphics processing unit; CPU-GPU hybrid system; parallel processing stream; sliding document segmentation

I. INTRODUCTION

Search engines have become the main way for users to locate and get effective information from the large-scale data collections. The modern search engines not only present the title, links and URL of a document, but also provide users with the document summary, which is called snippet in this paper. The snippet consists of one or more fragments which are extracted from the source document. Through this snippet, the query users can make a judgment of the relevance between the document and the query words.

There are several snippet generation approaches, which can be divided into two categories: one is query independent, and the other one is query-biased method. Query-biased approach consists of one or more document fragments which are associated with query. Compared to the former one, the query-biased snippet can provide user more meaningful information for them to make better decisions about which results are worthy of attention and which could be ignored. Almost all the modern search engines employ query-biased snippet generation approach.

Modern search engines usually have millions of daily visitations or even more, and each query may need to generate 10 or more snippets. As the search engines should generate different snippets for the same document with different queries, they cannot just cache and reuse the snippets based on the documents. That is, the snippet generation sub-system of search engines will consume considerable system resources. In a CPU derived system, mass of snippet generation tasks could be accomplished by using large number of CPU circles. However, this kind of system does not scale well and has limited computational and economical efficiency.

There are three main steps for snippet generation tasks. First, we need to segment the document into several fragments. The traditional segmentation methods which segment the texts in a truncation way have a common defect that the highly relevant fragment may be cut off, and it will lead to low snippet precision. Although solving this problem will cause high computational complexity, we believe that the emerging high-performance devices can provide some new way of solution. In our work, we propose a sliding document segmentation method which can avoid the truncation problem and achieve a preferable performance. Second, relevant fragments (with respect to the query) within the document should be chosen and ranked. This is referred to the "sentence selection" problem. The key issue of the problem is quantification of the relevance between the fragments and the query. This operation is called "scoring". We adopt a method of scoring based on vector space model (VSM) to compute the score of the relevance. The final step is to sort the fragments and construct the output snippets.

Generating a single snippet from a document-query pair is a lightweight task, but the number of this kind of task is extremely large in a search engine system. However, we find that the instructions of these tasks are nearly the same. Moreover, there is no interfering between the executions of any two tasks. That is, it is a typical single instruction multiple data (SIMD) paradigm. The common method is to use multiple CPUs with parallel process, but the efficiency of process is limited. Considering these features of snippet generation, we try to improve the performance and economical efficiency by using efficient parallelization method. In this paper, we proposed to use graphics processing unit (GPU) to solve this problem.

As the core graphic processor in the computer system, GPU has been developed into a highly parallelizable, multi-threading and multi-core processor. Originally, GPU

1521-9097/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPADS.2011.63

is designed for graphics applications. However, in fact, there are many non-graphical applications are also accelerated by GPU technology, such as signal processing, engineering simulation, and mathematical biology. Compared with the common method of using CPU, GPU can achieve a higher efficiency for snippet generation, because the scale of a single job is quite small, and the number of jobs is quite huge. It is more suitable for GPU process cluster to carry out this kind of tasks.

In this paper, we propose a CPU-GPU hybrid system, in which we design a process stream for snippet generation application of search engines. First, we analyze the computing characteristics of the snippet generation process using GPU. Then based on the analysis we design a cluster of data structures for the process which are suitable for the computing environment of GPU. As the response time is a key consideration for search engines, we employ three procedures to process the snippet generation based on the characteristics of the CPU-GPU hybrid system, and design a pipeline system for stream processing which can achieve a preferable response time.

To verify the effectiveness of our approach, we carry out a set of experiments to test the performance of the proposed system. First, we test the performance and applicability of the system to find out the internal and external performance indicators, mainly including throughput, response time, and processing time cost by each task. We present the bottlenecks and our targeted optimization measures. Second, we do some comparison experiments with baseline to test the performance and economical efficiency. We use a well-known snippet generation implementation in Lucene as the baseline, which is a famous project of Apache Software Foundation, and compare our approach with it to demonstrate the performance and economical advantage of our approach. Third, we carry out quality test experiments to illustrate that our approach can achieve a preferable snippet precision.

The rest of this paper is organized as follows. Section II presents the related work which mainly includes the snippet generation methods and general purpose computing on GPU. Section III discusses our snippet generation approach based on sliding document segmentation. Section IV shows how to use GPU to accelerate snippet generation. Experimental studies are presented in Section V. Section VI draws a conclusion.

II. RELATED WORK

The snippet generation has been addressed by many researchers since it is quite important for search engines. Kupiec, Pedersen and Chen [1] proposed a binary Naive Bayes classifier to solve the problem of fragment selection. However, this method was just designed for the kind of snippets that are generated without user query. It could be used to generate static snippets, which was namely query-independent approach. Considering the connection between documents and user query cannot be reflected by static snippet, Tombros and Sanderson [2] proposed a query-biased snippet generation method. The experimental results showed that, comparing with query-independent snippet, the query-biased snippet had advantages in both precision and recall. Meanwhile, users did not need to check the documents, such as webpages, and they would be able to judge the relevance of the documents and the query. After that, many scholars had accessed a series of achievements in this field with the vast majority query-independent methods. However, the snippet sub-engines of most of the large-scale search engines are query-biased. In this paper, we only focus on query-biased snippet generation.

Query-biased snippet generation approaches can be divided into two main branches. The most common approach is document-based, which splits a document into several fragments, and then calculates the relevance between the query terms and the fragments according to the features of each fragment. The fragments with high relevance are selected to construct the snippet as the output [2]. Highlighter is the snippet generation component of Lucene. It is an implementation of document-based method [3] that was widely used in enterprise search engine applications. However, there are two defects in this approach.

- It adopts a serial computational model and this may lead to low efficiency.
- The high relevant fragment may be cut off and will cause lower snippet precision.

Clarke and Cormack [4, 5] proposed another kind of query-biased approach called index-based method, and it was optimized by Gabriel Manolach [6]. The main idea of index-based method is to use the hit and position information of inverted index to calculate the most relevant fragment offsets.

Comparing with the query-independent snippet generation method, the query-biased snippet generation needs more computing resources, and it is more difficult to achieve high efficient cache. Therefore, there are many researches trying to improve its efficiency. A well-known method proposed by Turpin [7] uses a document compression method and makes a great improvement of efficiency in time and space. However, this method does not improve the performance of fragment selection process which is the key task. In order to accelerate this process, Manolach [6] proposed an improved index-based approach and can achieve a better performance in fragment selection. However, the index-based approach has recursive characteristic which means it is difficult to be converted to a parallel computing model. Therefore, in this paper, we adopt the document-based approach.

On the other hand, GPU vendors have started to offer better support for general-purpose computation on GPU [10]. In the field of information retrieval, some existing researches try to improve performance by using GPU. An important subject is using GPU for the inverted index related process. Ujaldon and Saltz [11] converted the process of retrieval into the pixel and texture process, and handled it through DirectX interface with GPU. Finally, they obtained an acceleration ratio up to 400%. Ding and He [12] applied Tesla GPU system of NVIDIA to

information retrieval and got an acceleration ratio during the retrieval process which is 2.37 times faster than before.

An increasing number of researchers are trying to using GPU for information retrieval to improve computational efficiency. However, to the best of our knowledge, there is no research trying to improve the efficiency of snippet generation of search engine by using GPU.

III. Snippet Generation Approach Based on Sliding Document Segmentation

A. Sliding Document Segmentation

Common document-based approaches divide the document into fragments in the first step [2, 3]. Considering that there is a linear relationship between the number of fragments and the computational complexity. To ensure necessary performance, the common approaches adopt truncation segmentation to split document. However, there will be a problem that the high relevant fragments may be cut off. Consider the following situation: a document has been parsed into a term vector: $DT = \{ dt_1, dt_2, dt_3, dt_4, dt_2, dt_1 \}$. Then, we split this document by a truncation method and get two fragments: $Frag_1 = \{ dt_1, dt_2, dt_3, \}$, $Frag_2 = \{ dt_4, dt_2, dt_1, \}$. If there is a user query which can be expressed as a term vector: $QT = \{dt_3, dt_4\}$, we can find that $QT \notin Frag_1$ and $QT \notin Frag_2$. In fact, the most relevant fragment has been cut off in the segmentation operation.

In order to solve this problem, we propose a sliding segmentation method that split a document by a sliding way. Sliding segmentation method can ensure that the high relevant fragments will not be cut off. However, in this way, the size of fragment set is much larger than the truncation way. In the former case, there will be $(\rho - \tau + 1)$ fragments created by the sliding method, while the number of truncation way is ρ/τ.

However, based on the belief that the number of terms which both occur in the document and query is much less than the length of the document, we believe that a large portion of fragments generated by the sliding method do not contain any query term. Therefore, in order to reduce the complexity, we can filter out the useless fragments before calculating the relevance scores.

B. Fragment Selection

After segmenting the document, we should decide which fragments need to be selected to construct the snippet, which determines the quality of snippet. Most of the existing approach [13, 14, 15] use the features of fragments to compute the final score.

In this work, we consider a fragment as a small document and the score of a fragment is the relevance between the small document and the query. To estimate the score, we present a model based on vector space model (VSM) [16] with the combination of fragment features. Given a query Q and a document F, the score function is:

$$Score(Q,F) = coord(Q,F) \times boost(F) \times \sum_{term\ t\ in\ Q} tf(t\ in\ F) \times idf(t)^2 \times boost(t) \quad (1)$$

where $coord(Q,F)$ is the occurrence number of the terms of Q in the document F, $boost(F)$ means the weight of F in the source document. $tf(t\ in\ F)$ is the frequency of the term t in fragment F, which means how many times the term t appears in F. $idf(t)$ is the inverse document frequency of term t which is a default weight factor. $boost(t)$ is an extension regulatory factor that is used to specify the weight of term t. When search engines or the searchers want to specify a different weight to the terms of a certain query, this factor will be used.

IV. Acclerate With GPU

A. Serial and Parallel Snippet Generation Algorithm

For a given document D and a query Q, the serial algorithm can be described as follows.

In the input step, a document D should be expressed as two vectors. One is a term's identity vector:

$$\text{DocTermIdV} = \{\text{Termid}(\text{dt}_1), \text{Termid}(\text{dt}_2), \ldots \text{Termid}(\text{dt}_\text{n})\}$$

where $Termid(dt_i)$ is the identification of term in the position i. The other is an IDF vector:

$$DocIdfV = \{idf(dt_1), idf(dt_2), \ldots idf(dt_n)\}$$

where $idf(dt_i)$ means the IDF value of term in the position i. The query Q is also expressed as two vectors. One is the identity vector of a term:

$$QueryTermIdV = \{Termid(qt_1), Termid(qt_2), \ldots Termid(qt_m)\}$$

which is similar to the $DocTermIdV$. The other is:

$$QueryBoostV = \{boost(qt_1), boost(qt_2), \ldots boost(qt_m)\}$$

where $boost(qt_i)$ indicates the boost factor of term in the position i. The $HitV$ here is a 0/1 vector which indicates the hits of document D with query Q. When $DocTermIdV_i \in QueryTermIdV$, $HitV$ can be assigned as the value 1. Otherwise, $HitV_i$=0. In a search engine, the snippet generation is to process the search results. In the search step, the search engine has already got hits information of the result documents. Thus, we assume that $HitV$ is an input but not an intermediate in the algorithm.

In the second step, the document will be split into fragment set $FRAG$ by the sliding method. If there are n terms in a document, the length of the set will be $(n - \tau + 1)$ in the condition that the maximal length of fragment is τ. The fragments that do not contain any query term in $FRAG$ will be filtered out. This step needs $HitV$. Then, we can use the scoring expression to calculate all the scores of fragments in $FRAG$. In the last, we can sort the scores and choose the top fragments to construct a snippet with a certain structure as output.

To parallelize the serial algorithm, we should find out the parallelizable part (s). As there is noninterference with each other among fragment scoring tasks, the process of computing the fragment scores can be parallelized. It is

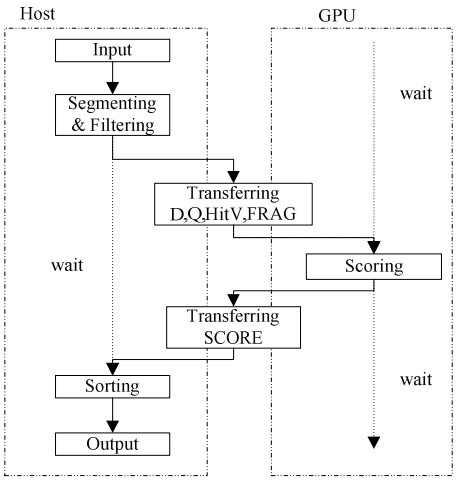


Figure 1. The executing flow of a single document-query pair task in CPU-GPU hybrid system

sensible that the process scoring of all fragments can be divided into smaller execution units. These units can be sent to many process units to execute. The Amdahl's law indicates that a parallel algorithm can obtain speedup ratio compared with the serial version as follows.

$$S(p) = \frac{p}{1 + (p-1)f} \tag{4}$$

where S is the speedup ratio, p is the number of processing units, and f indicates the proportion of serial parts in the parallel algorithm. This law states that the speedup ratio can be up to $1/f$ when $p \to \infty$.

Theoretically, in our algorithm, the factor f depends on the time cost by the following steps: initialization, filtering, task distribution, synchronization, sorting of scores and result outputting. These steps are hard to be parallelized.

B. Parallel Processing Stream Using GPU

In this section we will describe the parallel processing stream for batch document-query pairs.

GPU has its own arithmetic logic units (ALUs), controllers, memory and internal bus. However, as a peripheral, it must run on the host in the current system architecture. In a CPU-GPU hybrid system, GPU is used for parallel computing tasks, while CPU is used for the management, scheduling and input/output tasks. In this system, the data and instruction interactions between CPU and GPU are accomplished through the system bus. The processors inside GPU communicate with each other by the global memory, share memory and device bus.

We first introduce a single document-query pair task in this hybrid system, and find out the existing problem. Figure 1 shows the status of host, system bus and GPU in a task of single document-query pair. Although the scoring task of all the fragments has been paralleled in GPU, but only one of the three parts of the hybrid system, including host, system bus and GPU, is in working state while others are waiting. Therefore, there is a considerable waste of time in this algorithm.

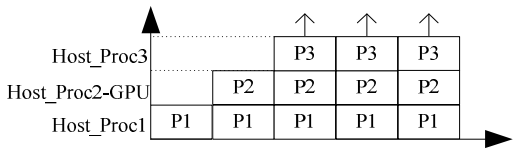


Figure 2. Pipeline architecture of processing stream in CPU-GPU hybrid system, where P1 is preprocessing, P2 is transferring and scoring, and P3 is sorting and outputting

To solve this problem, we design a pipeline system, which transforms the above process into three procedures.

Preprocessing. Firstly, the host gets the input data which contains a certain number of document-query pairs. In the next step, it splits the documents into fragment sets, and then filters out the irrelevance fragments. These three steps are all processed in CPU and host memory. We combine them to a procedure called preprocessing.

Transferring and scoring. A GPU device can only be used by a single host process at the same time. Transferring input data from host to GPU and transferring SCORE back from GPU to host rely on the system bus. Therefore, we need to consider GPU device and the system bus as the critical resources. We combine the following three steps into a procedure called transferring and scoring: transferring D, Q, HitV, FRAG to GPU; scoring in GPU; transferring SCORE back to host memory.

Sorting and outputting. Finally, the host will take the tasks that sort the score set and construct snippets as output. These steps will be considered as a procedure called sorting and outputting.

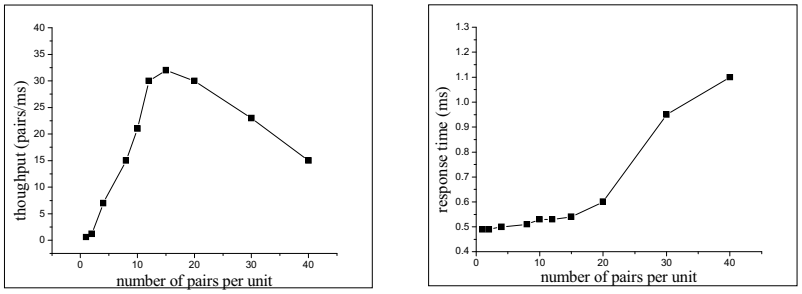
Based on the above abstraction, we present a pipeline architecture for CPU-GPU hybrid system, as shown in Figure 2. The preprocessing is handled by a host process unit, which is called Host_Proc1. The transferring and scoring procedure is handled by the system bus (with CPU instructions) and GPU, which is called Host_Proc2_GPU. The sorting and outputting procedure is handled by a host process unit called Host_Proc3.

In Figure 2, the process unit is a batch of document-query pairs, and the granularity of process unit is very important to the efficiency. In the section of experiments, we will present the influence of different granularities on the throughput and response time.

V. EXPERIMENTS

In this section we will evaluate our approach in performance, economic efficiency and quality, and compare it with the base-line.

All of our experiments in subsection A, B ran on a machine with an Intel Core2-Duo processors (2.2 GHz and 1 MB cache each) and 4 GB of main memory; and the GPU device is NVIDIA GTX200. All the codes are implemented in C++ and CUDA, which is an SDK for NVIDIA GPU, and compiled by NVCC, which is an integrated complier for C++ and CUDA. We took a data



(a) Throughput (b) Response time

Figure 3. Throughput and response time under different load levels

set containing 3,000,000 document-query pairs and generated every snippet constituting three fragments with respect to each pair. The corpus was selected from Reuters Corpus English Language 2006, 2007 and 2008. The queries were selected from 2006 TREC efficiency topics.

A. Granularity of Process Unit

The granularity of process unit indicates the maximum number of document-query pairs, which are contained in a process unit. This is a major factor that influences the performance of the processing stream. As it determines the time cost by the transferring and scoring procedure, it will bring a great impact on whether the processing stream can flow smoothly or not. We mainly tested the throughput and response time in a various process unit granularities, and the test result was shown in Figure 3. In this test, we kept the queue of waiting tasks full, which contained the tasks to be processed, so that the throughput would not be influenced by input.

Figure 3(a) shows that the throughput of the system varies with different unit size. From this figure we can find that the throughput of the system is low when the number of process unit is small. This is because the GPU resources cannot be effectively utilized, and the time is wasted on waiting and communications between host and GPU. With the increase of the unit size, the throughput also rises. It means that, with the growth of unit size, the GPU resources can be utilized more effectively. However, we can see the throughput dropping down when the unit size is more than 15. The reason is that, when the unit size goes beyond the optimum value, a large-size process unit will cost more time in GPU, while the host must spend more time on waiting.

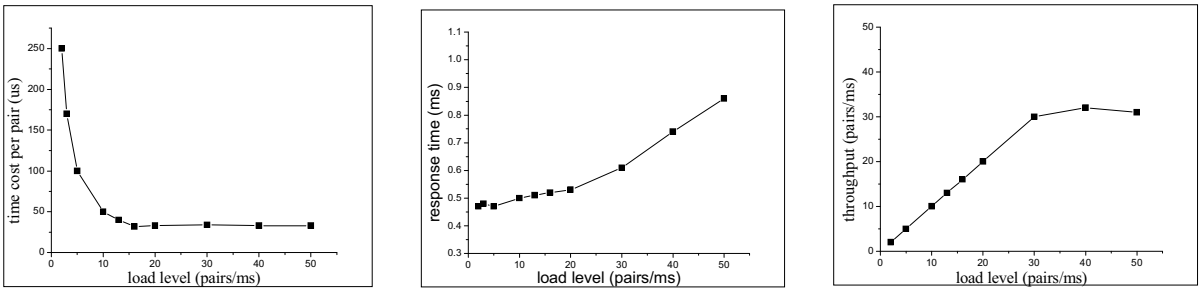
Figure 3(b) shows the response time of a snippet generation task. We can see that the response time increases with the growth of the unit size. This is because the larger unit size will cost more time in the transferring and scoring procedure. However, the slope of the curve becomes larger and tends to be constant. The reason is that, when the process unit size is small, time is mostly consumed in data transferring step, and when the unit size becomes large enough, the response time will have an approximate linear increase with the unit size. The process time of sorting step that has linear relationship with the unit size will account for the majority of the response time.

We adopt 15 as the value of granularity. Because our main purpose is to improve the throughput of system and the throughput achieves maximum while the response time in this granularity is also preferable.

B. Performance and Applicability Analysis

Compared with a serial approach, there are many steps added in our approach, including data distribution and transmission, interaction within and between the processing units. These additional parts will affect the efficiency of the system, especially when the system is under low workload. In order to find out the applicable conditions of our approach, we tested the system efficiency by measuring the time consuming of a document-query pair, response time on different load levels and throughput of the system on different workload.

Figure 4 shows the result of the test, from which we can find that when the workload is on a low level, the system is running with low efficiency. For example, when there are only 2 pairs inputted per second, the time that spends on generating a snippet is about 250 microseconds. This result mainly thanks to the internal feature of our approach. As mentioned above, the minimum process unit of our approach is a batch of document-query pairs, and in this test we use 15 as the size of process unit. When the workload is in a low level, for example, when there are only 5 document-query pairs inputted to the processing stream in a millisecond, the response time is about 0.5 milliseconds which we can find in the Figure 4(b). It means that the transferring and scoring procedure, which is handled by the GPU and system bus, can process a unit in about 0.5 milliseconds. Actually, it also means that the number of document-query pairs filled into a process unit is less than 5, and the process unit cannot be fully filled. When the system is running in this condition, the time cost in the scoring part is less than the situation that when the process unit can be filled fully. Nevertheless, the transferring part consumed roughly the same time. We can find that, when the workload is 30 pairs per millisecond, the time cost by a whole procedure is about 0.6 milliseconds, which is similar to the former situation that only 5 pairs inputted in a single millisecond. Therefore, the time cost to process a single pair using our approach has a wide disparity among different workloads. Compared with working on a high workload, the ratio of time that spent on the transferring parts will be much higher when our approach



(a) Time consuming of a pair (b) Response time measurement (c) Throughput measurement

Figure 4. Performance and applicability efficiency under different workload levels

running on a low level workload. As a result, the efficiency in this situation will be much lower. As the workload level increases, the efficiency becomes higher. When the load level becomes up to 15 pairs per millisecond, the process unit can be fully filled and the system efficiency gradually turns to be stabilized. The reason is that every process unit is fully filled and the ratio of time that spent on the transferring parts can achieve maximum value.

We can find in Figure 4 (b) that, when the workload is lower than 20 pairs per millisecond, with the increase of workload, the response time increases slightly. This is due to the system has a larger load margin. However, when the workload increases up to 30 pairs or higher, the response time will increase quickly, since the workload has reached the top capacity limit of the process stream.

In order to find out the bottleneck of our approach, we carried out a profiling test of the system. Transferring and scoring procedure which processed by the system bus and GPU is the key point of the processing stream. The transferring process has two steps. One is to transfer inputting data to the GPU global memory, and the data is consisted of document, query, fragment segment tag and other auxiliary data. The other step is to transfer the result score of each fragment back to the host. Obviously, the former part will cost more time as the size of data is much larger than the other one. We combine the two steps of data transferring processes as communication operation. This operation costs a part of process time and the rest time is consumed by the scoring operation, which should

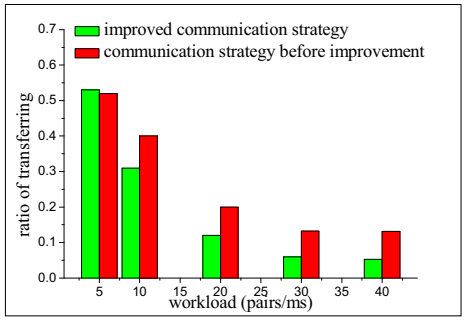


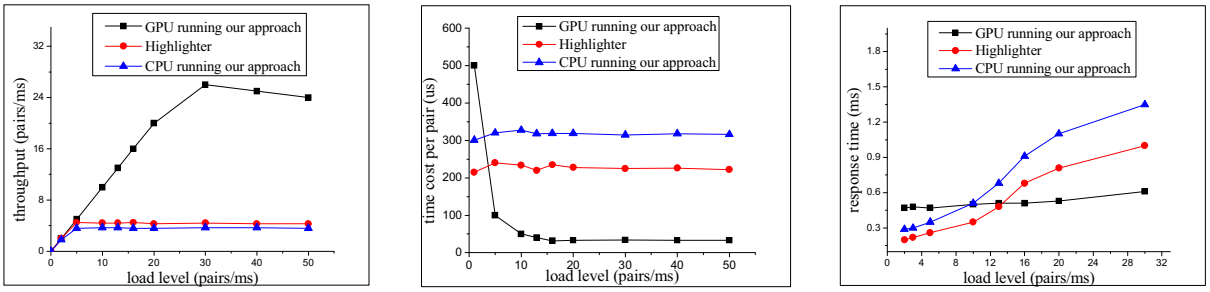
Figure 5. Comparison of time cost ratio of transferring between two methods

calculate score value of thousands of fragments. We measure the time cost of each part by using a profiler tool.

Figure 5 shows the results, in which the red columns indicate the ratio of time cost by communicate operation. Obviously, a lower ratio of time cost by transferring operation will lead to a more efficiency transferring and scoring procedure, as the process of real value is scoring. In the figure, we can find that, when the workload is in a low level, the ratio of time cost by communication is more than 50%. Time is mostly wasted, and the system is working in low efficiency in this case. With the growth of workload, this ratio decline and keep stable after it reached the point of 11.6%. In this condition, the throughput of system is about 20 pairs per millisecond. Through further analysis, we find that there is still room for improvement, as GPU has a large global memory. However, as each process unit only has no more than 15 document-query pairs, the size of data transferred to GPU is small. As a result, the utilization of GPU memory is low and the communication frequency is in a high level, which will cause time waste. In order to improve the performance in this situation, we adopt an improved strategy that, when the Host_Proc2-GPU process picks tasks from the queue of waiting tasks, it transfers the needed data of all the tasks in the queue to the GPU global memory, instead of the strategy that only transfers the needed data of tasks which form the next process unit. In this way, the frequency of data transferring will drop significantly.

In Figure 5, the green columns indicate the ratio of time cost by communication operation of the improved strategy. We can find that, when the workload level is low, the ratios of both strategies are nearly the same. Each time the Host_Proc2-GPU process picks tasks from the queue, the number of tasks in the queue is not larger or even smaller then the size of the process unit, so that the gap of frequency of communication between the two strategy is small. However, as the workload increases, this gap becomes larger. Finally, the ratio of time cost by communication of improved strategy is about 5.6% when the system is fully loaded, and the throughput in this situation is about 30 pairs per millisecond (see Figure 4(c)).

Figure 4 (c) shows the throughput of our approach on different workloads. From the figure we can find that, when the workload is 30 pairs per millisecond, the speed of input stream and output stream are the same. When the workload increases to more than 30, the throughput turns



(a) Throughput Measurement (b) Time consuming of a pair (c) Response time measurement

Figure 6. Comparison tests in performance with the baseline

TABLE I. F1 MEASURE RESULT

	GTX250 + Intel Core2 Due(x2)	Intel Core2 Due (x2)	Intel i5 2300 (x4)	AMD Opteron8 6134 (x6)
Speedup	5.8	1.8	3.7	5.5
Price($)	107+77	77	190	507

to be stable. In this condition, the throughput does not appear to reduce significantly. Because of the design of our architecture, the excess workload can be hidden by the process unit queue, and will not influence the transferring and scoring procedure which is the key procedure of the processing stream.

From above analysis, we can draw a conclusion that our approach is applicable under a high workload level. Compared with working in a low level, the time consumed for generating a single snippet will be shorter and the ratio of time cost by communication operation will be smaller. As a result, the system can achieve high efficiency. However, the response time will increase rapidly when the load level is too high, which means the system load should not exceed the full load threshold.

C. *Performance and Economical Efficiency Comparison*

In order to verify the performance of our approach, we carried out comparison tests. We measured three items which contained throughput, average time consuming of a document-query pair and response time. We contrasted our approach running on GPU, our approach running on CPU with single thread and Highlighter with single thread under a variety of load levels.

Figure 6 shows the results of these tests, from which we can see our approach running on CPU got a throughput less than Highlighter about 30%. This is because the sentence selection process of the former is more complex. However, our approach running on GPU got a throughput that is about 6 times more than Highlighter when the processing stream is under full load. The time consuming test indicates that the processing stream is running at full capacity, each snippet generation task would cost about 32 microseconds by using GPU while the Highlighter must spend 215 microseconds to do the same task. Therefore, we got a speedup of 6 by adopting our approach running in GPU. In the response time test, we got the result that the load level is lower than 30 pairs per millisecond. There would be little change in the response time by GPU. On the other side, as the throughput capabilities of Highlighter and our approach running on CPU are limited. When the input load level is exceeded, the response time will increase rapidly. When the load is in a high level, our approach running in CPU-GPU hybrid system can get a much lower response time. This result is attributed to the larger throughput of processing stream and the finite length of the task queue in the implementation.

To demonstrate the advantage of our approach on economical efficiency, we also carried out a comparison experiment between our approach running in CPU-GPU hybrid system and Highlighter running on different CPUs. We used three CPUs with 2, 4, 6 cores to run the same algorithm using 2, 4, 6 threads, and recorded the maximum throughput of each test.

Table I shows the results of the tests, in which each column indicates the throughput speedup of the different devices comparing with the throughput of Highlighter running in a single thread, and the average price of the devices in market. From the table we can find that, compared with the mainstream devices, to get similar speedups, our approach running in the hybrid system cost less than the multi-core CPUs.

D. *Snippet Quality*

Although our primary focus in this work is on efficiency, we briefly describe our test on the effectiveness of our approach and Highlighter. As there is no benchmark for evaluating the snippet quality, we performed a user study to evaluate the effectiveness of our approach contrastively. In this test, we took 160 document-query pairs and also chose 3 fragments from each pair. Ten graduate students who were not involved in our research were invited to participate in this study.

The most commonly way to measure retrieval effectiveness called F1 measure, which is the harmonic mean of precision and recall. It balances recall and precision in a way that gives them equal weight [17,18]: $F_1 = 2rp/(r+p)$, where $r = \alpha/(\alpha+\gamma)$ as recall and $p = \alpha/(\alpha+\varepsilon)$ as precision; here α indicates the number

TABLE II. F1 MEASURE RESULT

	r	p	$F1$
our approach	0.48	0.44	0.459
Highlighter	0.46	0.43	0.444

of fragments both selected by user and the testing approach; γ indicates the number of fragments in user result list, but does not appear in the testing approach; ε indicates the number of fragments in the result list of the testing approach, but not in the user result list. Table II shows the result of this test. We can find that our approach can get better snippet quality compared with Highlighter.

In summary, experimental evaluation shows that our snippet generation approach is applicable under a high workload level. Meaning while, compared with Highlighter, it can get a speedup of more than 6times in average process time, and achieve better snippet quality.

VI. CONCLUSION

In this research, we proposed to import GPU as a parallel coprocessor to deal with the large-scale query-biased document snippet generation tasks. Firstly, we pointed out the common problem that high relevant fragments may be cut off by the truncation segmentation. We adopted a sliding document segmentation that can avoid this defect. Then we presented a fragment selection method which adopted an improved SVM-based expression to score the relevance between fragment and query. Finally, we constructed a CPU-GPU hybrid system in which the GPU is used for parallel computing tasks while the CPU is responsible for the management, scheduling and input/output tasks. We classified the snippet generation process into three procedures according to the hybrid system architecture, and designed three-level processing stream based on this abstraction. The experimental results showed that our approach can gain a higher efficiency on high workload level compared to the baseline Highlighter and can also achieve a slightly better snippet quality.

ACKNOWLEDGEMENTS

We thank Dr. Zhao Zhang at Iowa State University, Dr. Zhichun Zhu at University of Illinois at Chicago and Dr. Weijun Xiao at University of Minnesota for their valuable advices and insightful comments. This work is partially supported by National Natural Science Foundation of China under grants 61173170 and 60873225, National High Technology Research and Development Program of China under grant 2007AA01Z403, Natural Science Foundation of Hubei Province under grant 2009CDB298, Wuhan Youth Science and Technology Chenguang Program under grant 200950431171, Open Foundation of State Key Laboratory of Software Engineering under grant SKLSE20080718, Innovation Fund of Huazhong University of Science and Technology under grants 2011TS135 and 2010MS068, and Graduate Innovation Fund of Huazhong University of Science and Technology under grant HF-08-03-2011-210.

REFERENCES

[1] J. Kupiec, J. Pedersen, and F. Chen, "A trainable document summarizer," in Proc. 18th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp.68–73, 1995.

[2] Anastasios Tombros and Mark Sanderson, "Advantages of query biased summaries in information retrieval," in 21st Conference on Research and Development in Information Retrieval (SIGIR'98), pp.2-10, 1998.

[3] D. Cutting. Lucene. http://lucene.apache.org.

[4] C. L. A. Clarke and G. V. Cormack, "Shortest-substring retrieval and ranking," in ACM Trans. Inf. Syst., pp.44-78, 2000.

[5] C. L. A. Clarke, G. V. Cormack, and F. J. Burkowski, "An algebra for structured text search and a framework for its implementation," in The Computer Journal, pp.43-56, 1995.

[6] Holger Bast, Marjan Celikik and Gabriel Manolache, "Efficient Index-Based Snippet Generation," in WWW'09, April 20-24, 2009.

[7] Andrew Turpin, Yohannes Tsegay, David Hawking, and Hugh E. Williams, "Fast generation of result snippets in web search," in 30th Conference on Research and Development in Information Retrieval (SIGIR'07), pp.127-134, 2007.

[8] Donald Metzler and Tapas Kanungo, "Machine Learned Sentence Selection Strategies for Query-Biased Summarization," in SIGIR 2008 Workshop on Learning to Rank for Information Retrieval (LR4IR 2008), pp.136-148, 2008.

[9] Ramakrishna Varadarajan and Vagelis Hristidis, "A system for query-specific document summarization," in 15th Conference on Information and Knowledge Management (CIKM'06), pp.622-631, 2006.

[10] General-Purpose Computation Using Graphics Hardware (GPGPU). http://www.gpgpu.org.

[11] Manuel Ujaldon and Joel Saltz, "The GPU as an indirection engine for a fast information retrieval," in Int. J. Electronic Business, Vol. 3, Nos. 3/4, 2005.

[12] Shuai Ding, Jinru He, Hao Yan and Torsten Suel, "Using Graphics Processors for High-Performance IR Query Processing," in WWW 2008, April 21–25, 2008.

[13] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell, "Summarizing text documents: sentence selection and evaluation metrics," in SIGIR99, pp.121–128, 1999.

[14] H.P. Luhn, "The automatic creation of literature abstracts," in IBM Journal, pp.159–165, April 1958.

[15] T. Sakai and K. Sparck-Jones, "Generic summaries for indexing in information retrieval," in SIGIR01, pp.190–198, 2001.

[16] Stephen Robertson, Hugo Zaragoza, "The Probabilistic Relevance Model: BM25 and beyond," in the 30th Annual International ACM SIGIR Conference 23-27 July 2007.

[17] Yiming Yang, "An Evaluation of Statistical Approaches to Text Categorization", in INFORMATION RETRIEVAL, Volume 1, pp.69-90, 1999.

[18] C. J. van Rijsbergen. Information Retireval. Butterworths, London, 1979.