The Hong Kong Polytechnic University

Department of Computing

GPU Accelerated Hot Term Extraction from User Generated Content

CHENG MING FUNG

A Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Philosophy

December 2011

CERTIFICATE OF ORIGINALTIY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signature)

Cheng Ming Fung _____(Name)

Abstract

This thesis aims at developing and investigating an efficient approach to hot term extraction. In the Web 2.0, the user generated content (UGC) is increased dramatically in different Consumer Generated Media (CGM) such as forums and blogs. People easily search their knowledge and opinions in CGM as well as generate Word Of Mouth (WOM) in different online channels. Facing the huge amount of data, it is not easy to find the useful information even using a search engine. Having a good hot term extraction algorithm can reveal hidden information to users and also provide an indicator in the search results, so that users can easily know which terms are popular in the search results.

In this thesis, a GPU based hot term extraction algorithm is presented. Graphics Processing Units (GPUs) is designed for data-parallel computations. Comparing to running a single program with multiple data in CPU, GPU can have faster execution. The hot term is defined as a word that appears frequently in the search result. We assume that the greater the frequency of appearance of a term, the more the relevancy of the term to the users. As there are lots of terms in the searched results, processing them is time-consuming. The proposed GPU based hot term extraction algorithm can achieve a fast performance and works well in real-time applications.

Acknowledgement

I would like to express my gratitude and thanks for the support and advice given to me by my supervisor, Dr. Chung Fu-lai (Korris), without which it would have been far more difficult to bring my project to a successful conclusion.

I also thank Dr. Fu Tak Chung and those management team of Well Synergy Limited to give me a chance to participate this Teaching Company Scheme.

My special thanks are extended to my supervisor in Well Synergy Limited, Dr. Chuang Siu Nam, for spending much time on the research guidance and technical support. Without those invaluable supports, I would never have finished my research.

Finally, I would like to thank my parents and friends who have provided me lots of encouragements and supports in tangible and intangible ways.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem and Motivation

With the rapid development of the Web, people have a convenient and effective platform to share things in a public environment. For example, sharing knowledge in Wikipedia, sharing videos in YouTube and sharing opinions in blogs. These kinds of channels can be grouped as social media.

Nowadays, social media has been increasing dramatically. This fast growing trend has provided resourceful ways for information collection. It is important for people to discover hidden knowledge by mining through plain textual data. Thus, it offers unprecedented opportunities and challenges to researchers of many different work sectors. The social media textual data analysis has a unique characteristic that the content is written in free format with human readable language, which is also called user generated content (UGC). If a customer uses search engines to find some opinions on a certain brand of mobile phone, the search engines cannot answer questions like on what features of mobile phone that consumers have discussed the most. This is because the free format textual data cannot be processed easily by machines. In general, it is impossible for people to find out the exact information they want every time by browsing different websites. In some cases, the information they want is even hidden. This is the reason why text mining on web applications has become popular.

In order to explore more useful information from the search results by using search engines, scholars have proposed heuristics algorithms from fields like information retrieval, text mining and machine learning. In text mining, Feldman et al. [1] used association rule mining to find the patterns across terms in documents. The relationships of terms are found and then the documents can be represented by a set of terms. Furthermore, the documents can be organized in a hierarchical taxonomy structure.

Apart from using association rule mining in text mining, Larsen and Aone [2] presented a document clustering algorithm that implements the k-means algorithm and determines a good initial clustering. By clustering text documents, documents about similar topics will be grouped together. Typically, a search engine returns a long list of searched documents. However, users are limited to view all the documents. Document clustering effectively categorizes the documents so that users can easily find the interested information in those clusters.

There is a clustering search engine called “Carrot Search” which is implemented by Weiss and Osinski [3]. The search results of Carrot Search are clustered with different topics. With the fancy web interface, users can easily select the clusters with corresponding topics to view the search results. The clustering algorithm helps search engines effectively list out the search results for users and hence, users can easily find those clusters they are interested in.

With the concept of text clustering, finding relevant keywords in searched results is beneficial to users. According to the query in search engine, the searched results

must contain the searched keyword and users may also be interested in other terms that appear frequently in the searched results.

To analyze hot terms which are defined as the words that appear frequently in the searched results, one of the easiest ways is to directly convert the document into a bag of words (BOW) representation and then count the frequency of each term. We assume that the greater the frequency of appearance of a term, the more the relevancy of the term to the users. There are different algorithms for finding the frequent terms in document collections. For example, Ahonen [4] presented an algorithm to find the frequent word sequences. Given a document collection and frequency threshold σ, if a sequence appears in more than σ times, the sequence is considered as frequent. This algorithm is to find the word sequences rather than only limited to a single word. However, finding word sequence is time consuming, it is not fast enough in real time applications. The details will be discussed in Chapter 4.

Hot term extraction is an important feature of social media tools in the social media industry. K-Matrix CI [5, 6] is one of the social media tools that provide a monitoring platform for customers to discover the word of mouth (WOM) knowledge in the CGM. Users can type the query and select the time period, and the K-Matrix CI can real-time return the related results which come from CGM sources. For example, searching “pccw” from 2010-01-01 to 2010-12-31, there are more than 100K posts returned, the search report is shown in Appendix A. One of the analyses in the report is hot topic listing among the searched results. The hot topic listing shows the topics that consist of the most matched posts. However, the hot topic listing is not deep enough to mention what terms are popular as well as lots of discussion on them. The

hot term extraction can enhance this limitation. This is the reason why the hot term extraction is important in the social media tools.

The hot term extraction is also useful in academic area. PubMed [7] is a database which collects millions of biomedical publications. PubMed is also a part of National Center for Biotechnology Information (NCBI) which provides a search function to search documents from PubMed database. Kou et al. [8] developed an application called PubCloud whose function is to summarize the search results from PubMed. An example of PubCloud is shown in Figure 1.1 [9] which shows the terms that are extracted from the top 200 results of "genes" query. Font size and font color indicates frequency and recency of the results respectively.

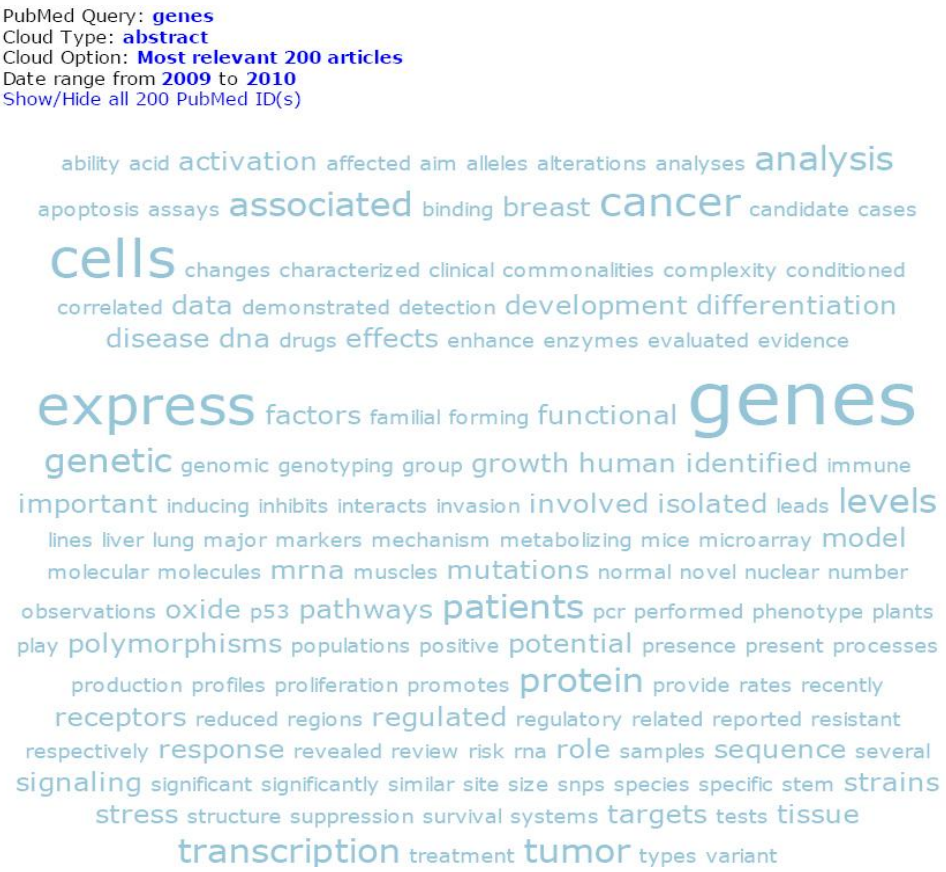


Figure 1.1 – Example of PubCloud [9]

In the algorithms of text mining [1, 2, 4], they are not capable of executing in real time application as the processing time is not fast enough. Although the Carrot Search [3] runs very fast with the processing time is around one second, it just processes 200 documents each time. Also, the PubCloud can only process at most 200 documents each time. The quantity of documents is not large enough in the current practical usage.

From the documents collected by the K-Matrix Digital Intelligence Ltd., each document contains on average 40 unique words. If there are 50K documents in the searched results, it contains around 2 million words. This is a challenge in processing hot term extraction. As there are lots of terms in the searched results, finding the most frequent terms is time consuming and thus a fast algorithm is needed.

To achieve a fast algorithm for hot term extraction, we propose to implement the algorithm with Graphic Processing Unit (GPU). Generally, GPUs can be regarded as high performance many-core processors with 10x faster computation than CPUs. GPUs was designed for games and graphics applications, but now GPUs can be used as a general-purpose parallel processors with support programming languages such as CUDA [10]. Therefore, it is possible for the developers who lack of the knowledge of the graphics rendering pipeline but could write GPU programs properly.

However, there are at least two challenges in developing GPU based hot term extraction.

1) As hot terms are terms that appear at a certain frequency, key-value sorting is needed for extracting the most frequent terms where the key is term id and the value is frequency. In view of the large number of terms in the

searched results, term id is a long type integer (64-bit). However, up to now, not all the programming libraries support GPU based sorting algorithm for sorting 64-bit key-value pairs.

2) A hot term extraction algorithm has some core functions such as sorting, filtering and counting. As GPU programming lacks support for some basic functions, it is a difficult task to design a GPU based framework for hot term extraction.

1.2 Objectives

In view of the two problems mentioned in section 1.1, the objectives of this project are set as follows.

(i) To develop a flexible GPU based sorting algorithm for processing key-value pairs whose key and value are not restricted to 32-bit integer type.

Recently, the latest version of CUDA supports GPU based sorting algorithm for sorting 64-bit key-value pairs. However, the JAVA binding library which is called JCUDA has not yet supported 64-bit. Both the key and the value are restricted to 32-bit integer. We make an attempt to address this issue and propose a flexible way that the key or value can be more than 32-bit.

(ii) To propose a GPU based hot term extraction framework that all core functions are implemented with GPU and the extraction task is efficient in our current practice.

In accomplishing this objective, we can help users to find some hidden information

by extracting hot terms in the searched results. Comparing the performance with that of CPU, GPU based hot term extraction framework could run at faster speed.

1.3 Outline of the Thesis

In this thesis, a GPU based hot term extraction framework is proposed. The thesis is divided into five chapters. In addition to the introduction in Chapter 1, a literature review on GPU computing is provided in Chapter 2. Chapter 3 introduces a flexible sorting algorithm called Borrow-Bit Sorting. Chapter 4 introduces a GPU based hot term extraction algorithm and Chapter 5 concludes the results and summarizes recommendations for future study.

Chapter 2

Parallel and GPU Computing

2.1 Introduction

Traditionally, most of the applications are implemented as sequential programs with a processor executing the instructions one by one. As the speed of sequential programs running on a single processor is limited, in order to raise the execution speed of the software applications, software developers now rely on the advances in the corresponding hardware and hence they have more and more expectations on each new generation of processors. However, the enhancement of the speed of processors faces different challenges due to the energy-consumption and heat-dissipation issues. Thus, processor vendors have switched to develop other models where multiple processor units, referred to as processor cores, are used in each chip to increase the processing power [11].

Multi-core processors have led to a tremendous impact on the IT industry. Multiple instructions can be parallel executed in multi-core processors, so that the multi-core processors are much faster than a single processor. The multi-core model began as two-core processors, and up to now, Intel Core i7 has four processor cores and supports hyper-threading with two hardware threads designed to maximize the execution speed of sequential programs.

Besides the development of multi-core processors to increase the execution speed, the graphics processing unit (GPU) is a new trend for software developers to write parallel programs for achieving higher execution speed. GPU is the processor

typically embedded on the graphics card in a computer. The GPU was originally developed to enhance the performance of 3D computation in game or video. Recently, some software developers have turned to use the GPU for general-purpose computation (GPGPU), in order to take the advantage of GPU which is characterized by Single Instruction Multiple Data (SIMD) architecture, i.e. data can be processed in parallel. The GPU is a fast and parallel processor and it can support parallel programs for artistic renderings and mathematical calculations.

2.2 GPU Performance and Hardware Architecture

Nowadays, the market is seeking for real-time processing, high-definition 3D graphics, and GPU has evolved into a highly parallel, multi-threaded, many-core processor model with very high computational power and memory bandwidth. Comparing the performance of CPU, GPUs have better performance in floating-point calculation as shown in Figure 2.1 and Figure 2.2 [12]. With the exploitation of GPU in general-purpose computation, the performance of GPU has been shown with large performance improvement. For example in 2009 as in Figure 2.1, the ratio between NVIDIA GPUs and Intel multi-core CPUs for peak floating-point calculation throughput can be about 10 to 1.

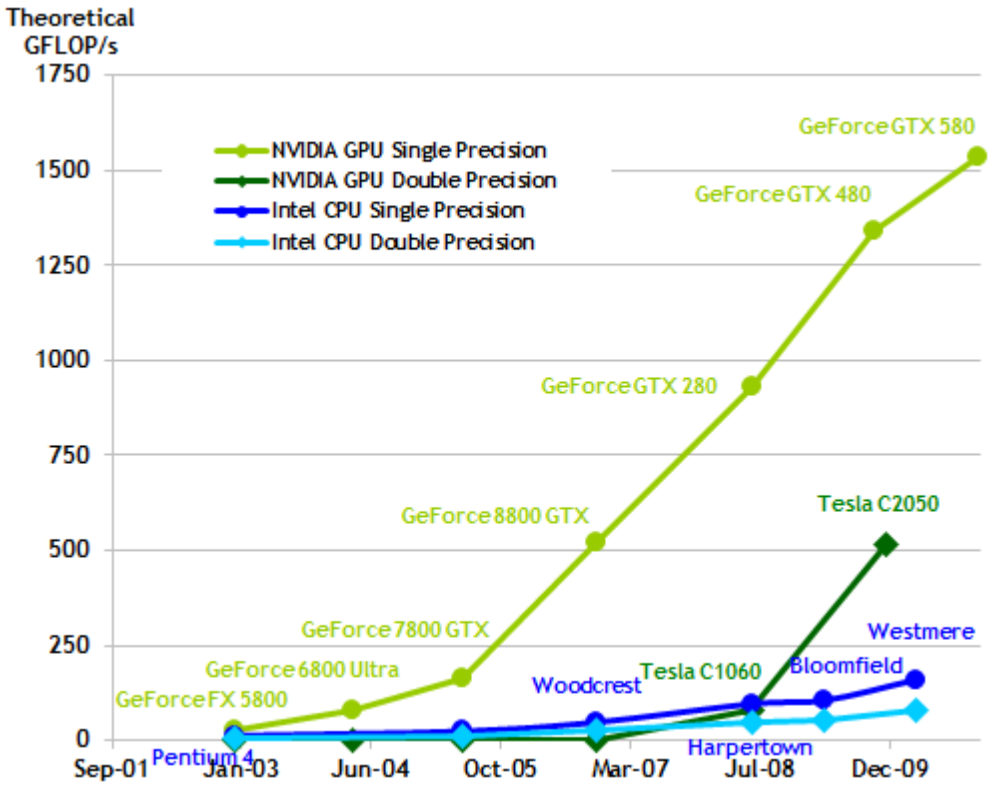


Figure 2.1 - Floating-Point Operations per Second for CPU and GPU [12]

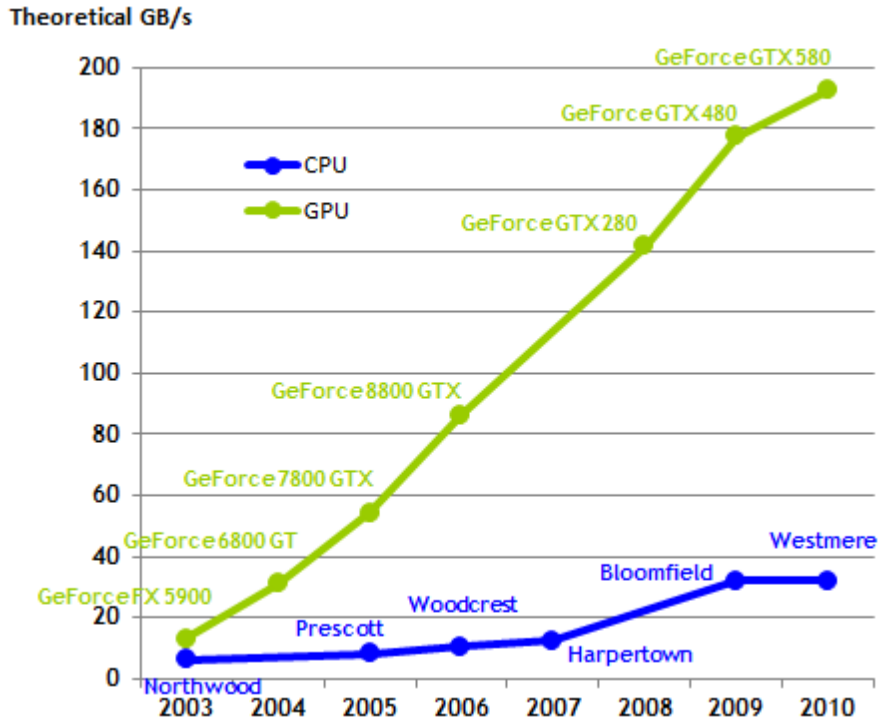


Figure 2.2 - Memory Bandwidth per Second for CPU and GPU [12]

The architecture of CPU and GPU can be compared with the help of Figure 2.3 [12]. GPU has more transistors devoted to data processing while CPU has more transistors dedicated to data caching and flow control. The architecture of CPU was designed to optimize sequential code's performance. It makes use of control logic to allow instructions from a single thread of execution to execute in parallel or even out of their sequential order while maintaining the appearance of sequential execution [11]. To maintain the high speed of execution, large amount of cache memories are used to reduce the instruction and data access latencies in large complex applications. On the other hand, the GPU was originally designed for graphics rendering and it has evolved as a highly compute intensive and highly parallel computation system. As the GPU is designed for data-parallel computations, a single program can be executed on many data elements in parallel, and hence more transistors are placed into arithmetic operations rather than memory operation. With a concept of SIMD, every ALU processor must execute the same instruction at the same time, and only the data may different.

In parallel programs, data elements can be mapped into multiple threads, thus facilitating different threads cooperating with each other to complete the work faster. When applications have to process large data sets, using a data-parallel program can speed up the computations.

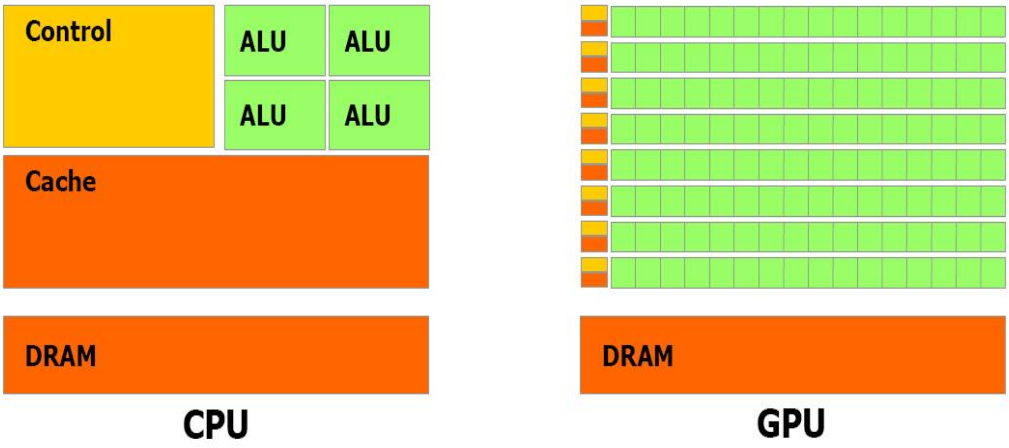


Figure 2.3 - The GPU Devotes More Transistors to Data Processing [12]

2.3 CUDA for GPU Computing

2.3.1 Introduction

CUDA is defined as “compute unified device architecture”. It was introduced by NVIDIA at the end of 2006. CUDA is a general purpose parallel computing architecture. It allows users to use extensions to the C and C++ programming languages to develop applications for parallel computing on GPUs.

In NVIDIA G80 and its successor chips, all of them support CUDA which provides general-purpose parallel programming interface. Figure 2.4 shows the architecture of a NVIDIA Tesla GPU with 112 Streaming Processor Cores [13]. It is organized into a number of highly threaded streaming multiprocessors (SMs) and there has a number of streaming processors (SPs) in each SM. The SM is mainly responsible for managing, scheduling and executing threads in groups of 32 parallel threads called warps. And the main function of SP is to share control logic and instruction cache. In NVIDIA GPUs, different GPUs have different number of SMs and SPs in a building block.

The memory in Figure 2.4 is the DRAM of GPU but not the DRAM of CPU. In CUDA, the GPU is called device while the CPU is called host. The device cannot access the host's memory and it can only access the memory located on the device itself. Therefore, data have to move from the host's memory to the device's global memory when executing a program in device. The data is then split into smaller parts for GPU's multiprocessors to process them in parallel.

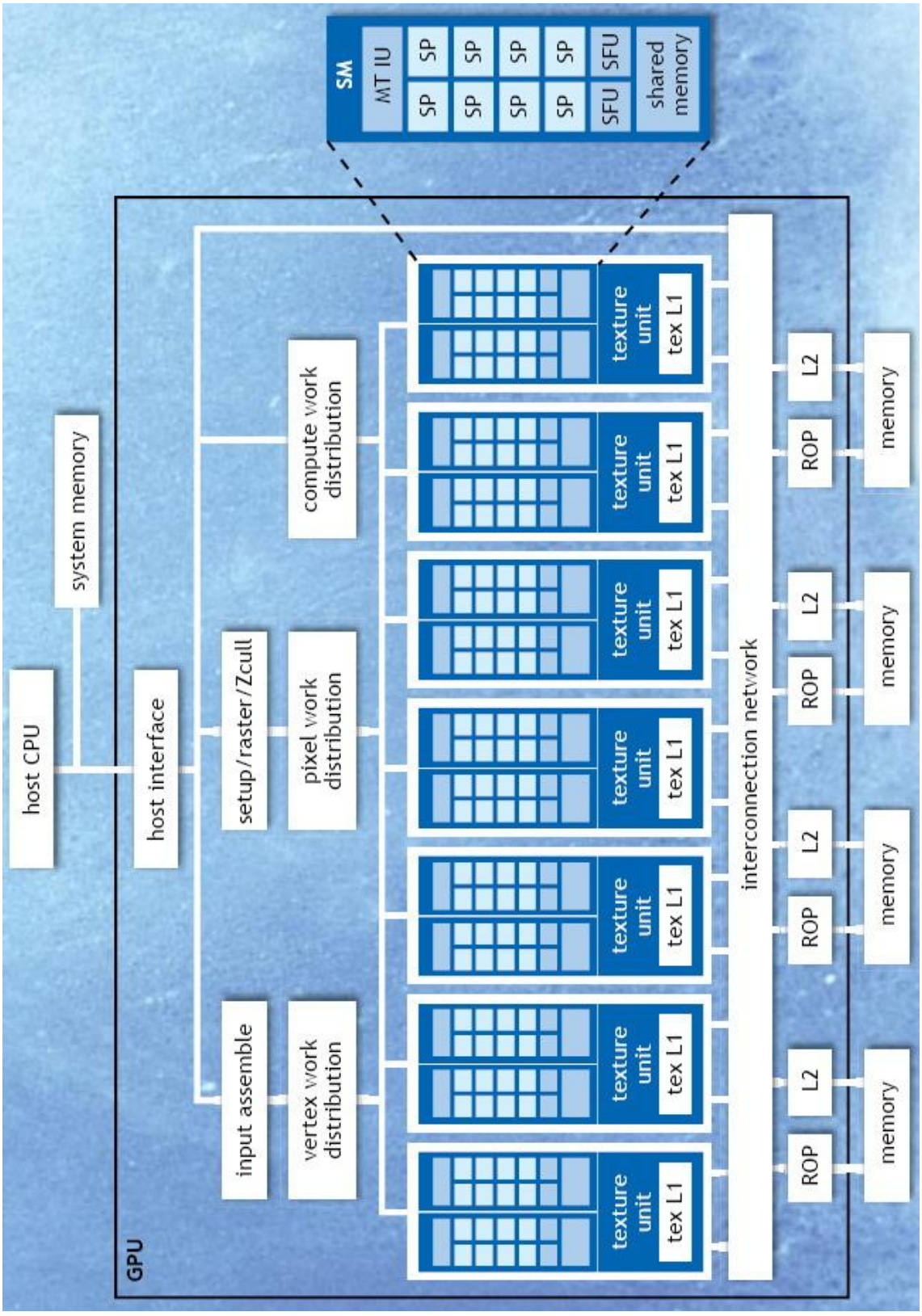


Figure 2.4 - NVIDIA Tesla GPU with 112 Streaming Processor Cores [13]

2.3.2 CUDA Programming Model

The CUDA programming environment allows programmers to write a C function to be executed in the device, and it is called kernel. A kernel is executed as single program multiple data model (SPMD) and it is executed N times in parallel by N different CUDA threads, where the number of threads (N) can be defined by users. The execution of a CUDA program is illustrated in Figure 2.5 [11]. The CUDA programming model separates the memory between host and device, therefore the execution starts with host (CPU) execution. Since, the host and device have separate memory spaces, the data in the host memory have to transfer to the device memory. When a kernel is launched, the execution is moved to the device. According to how the user defines the number of threads in the host, the threads are generated when the kernel is launched and collected into a grid. The execution of two grids of threads is shown in Figure 2.5. When all threads in a grid complete their executions, the grid terminates, and the execution is moved to the host until another kernel is launched.

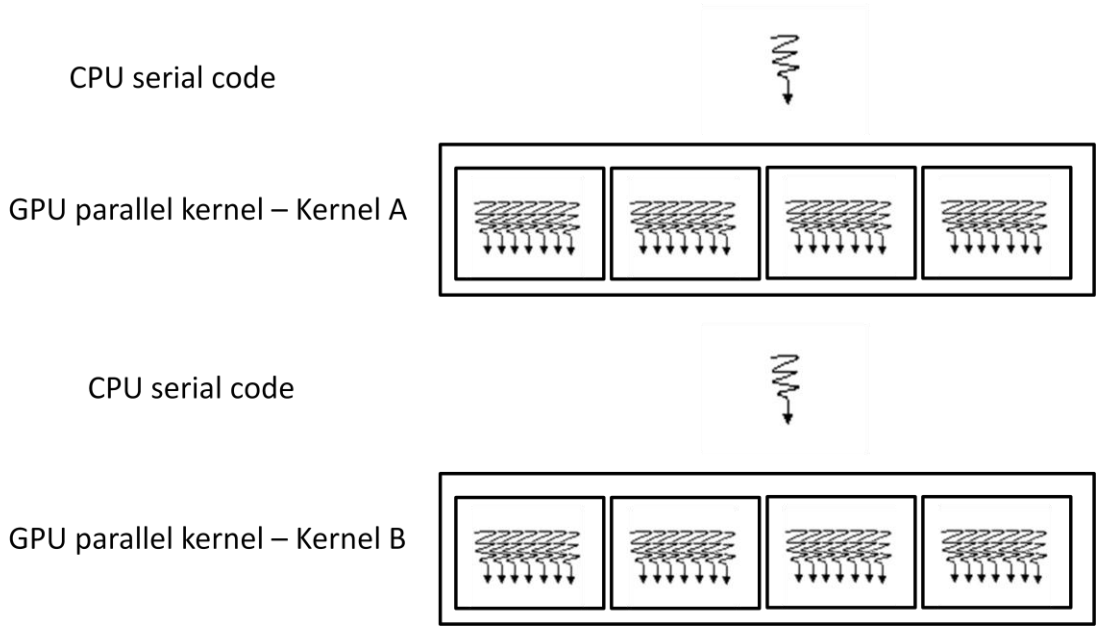


Figure 2.5 – Execution of a CUDA program [11]

Figure 2.6 [11] and Figure 2.7 [11] show the hierarchical structure of CUDA threads. Threads can be organized to form a one-dimensional, two-dimensional or three-dimensional thread block. Multiple thread blocks can be organized to form a one-dimensional or two-dimensional grid. During the launching of a kernel, the CUDA runtime system generates the corresponding grid of threads and all of the threads are executed in the same kernel. In order to distinguish between one and other, a unique ID is assigned and it also helps identifying the proper portion of the processing data. The unique ID is represented by coordinates which combine with block index and thread index. For a three-dimensional thread block with size (D_x, D_y, D_z), a thread ID is defined as (x, y, z), and using the size of a block can retrieve its thread index .In this example, the index of thread ID (x, y, z) is ($x + y\ D_x + z\ D_x\ D_y$). Since all threads of a block share the limited memory, there is a limitation to the number of threads per block. The maximum size of a block, $D_x \times D_y \times D_z$, is limited to 1024 threads in the latest GPU specification.

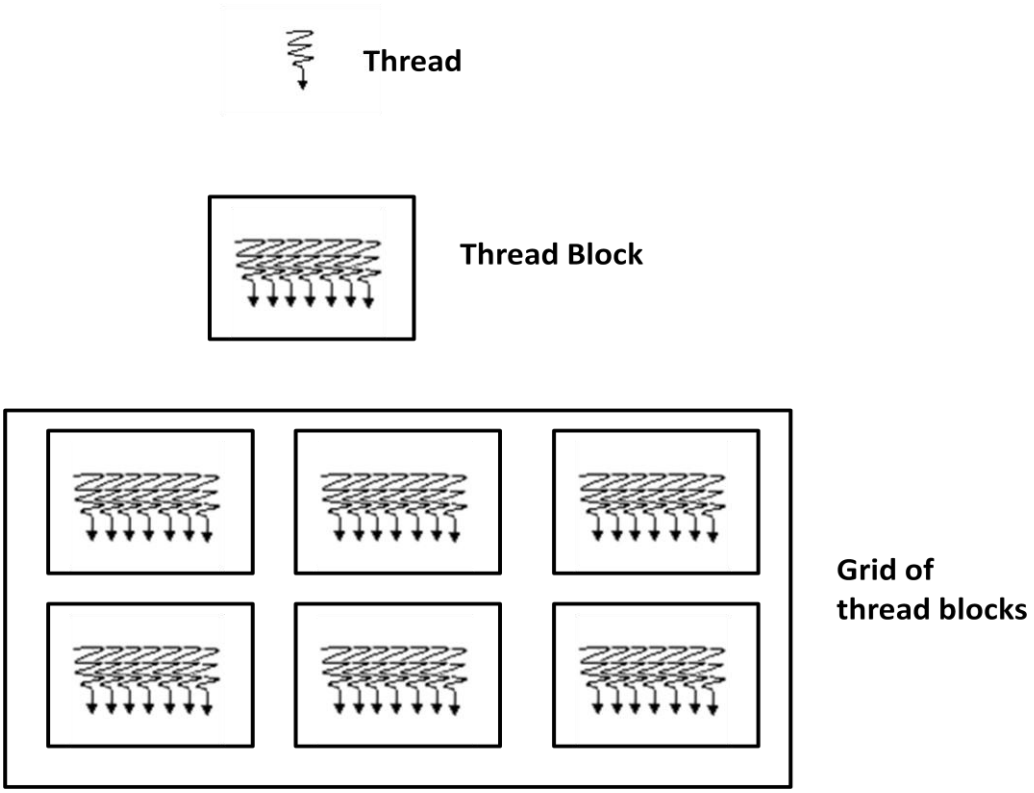


Figure 2.6 – The hierarchy structure of CUDA thread [11]

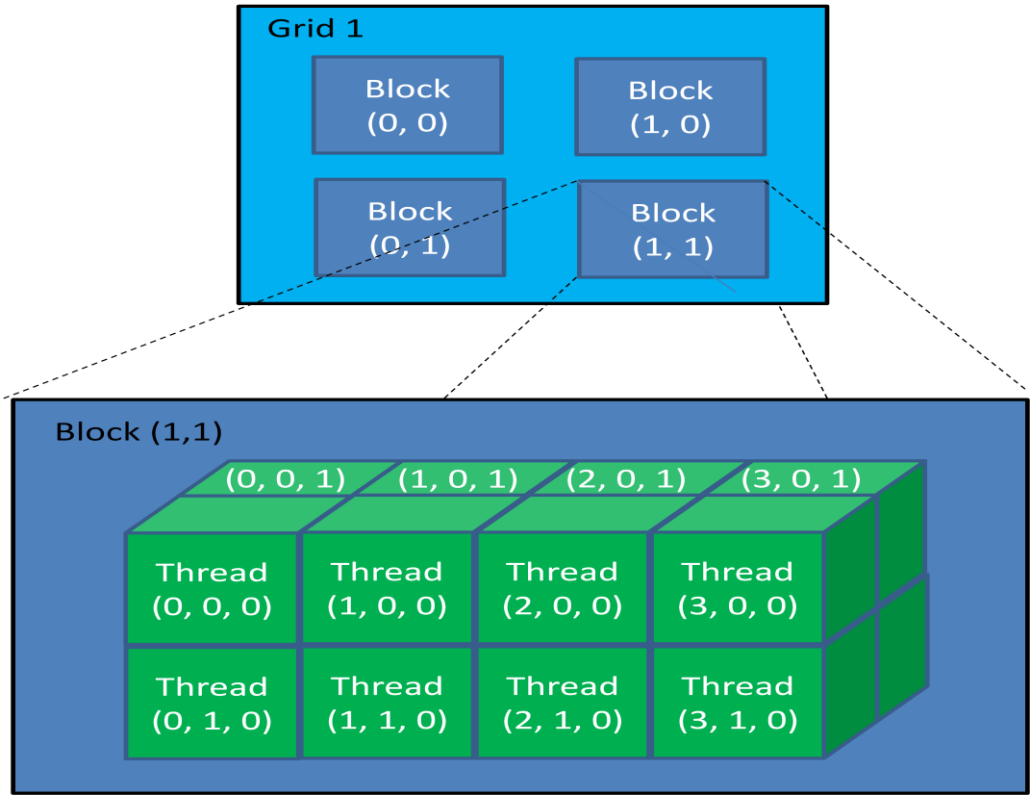


Figure 2.7 – The hierarchy structure of CUDA thread with two-dimensional block and three-dimensional thread [11]

There is a resource limitation on thread assignment in different GPUs. For example, NVIDIA GeForce GTX 470 has 14 streaming multiprocessors (SMs), up to 32 blocks can be assigned to each SM. The maximum number of threads limited on each SM is 1024. Therefore, the thread assignment can be in the form of 4 blocks of 256 threads each, 8 blocks of 128 threads each, etc. Since there are 14 SMs in GTX 470, up to 14,336 threads can be simultaneously residing in the SMs for execution.

After defining the number of threads in a block, the block is assigned to a SM. The block is further divided into 32-thread units called warp, it is the unit of thread scheduling in SMs. If each block has 1024 threads, there are 1024/32 = 32 warps per SM. A warp executes a common instruction at a time, so peak performance is reached when all 32 threads of a warp agree on the instruction path. If one thread of a warp wants to execute a branch instruction, the warp serially executes the diverged instruction and disables the threads which are not on that path. When all paths complete, the thread converges back to the same instruction path [12].

2.3.3 CUDA Memory Model

Before executing a kernel by a large amount of threads, the data have to transfer from the host to the device memory. In CUDA memory model, there are several types of memory that can be used to achieve high execution speeds in the kernel. The overview of CUDA device memories is shown in Figure 2.8 [11]. At the bottom of the figure, there are global memory and constant memory. These are the memories that the host code can transfer data to and from the device. Constant memory allows read-only access by the device code. Threads on the device have their local memory called register, and each thread can only access its own registers. Shared memory is allocated to thread blocks, and each block can only access its own shared memory.

Threads in the same block can cooperate by sharing their input data and the intermediate results of the work. However, threads cannot cooperate with different blocks. Variables that reside in register and shared memory can be accessed at a very high speed at the same time.

In summary, CUDA has a good design of programming model and memory model, so that kernel can be executed at high speed and in a parallel manner. Software programmers can specifically define the size of block and grid, and skillfully use different types of device memory to archive high performance.

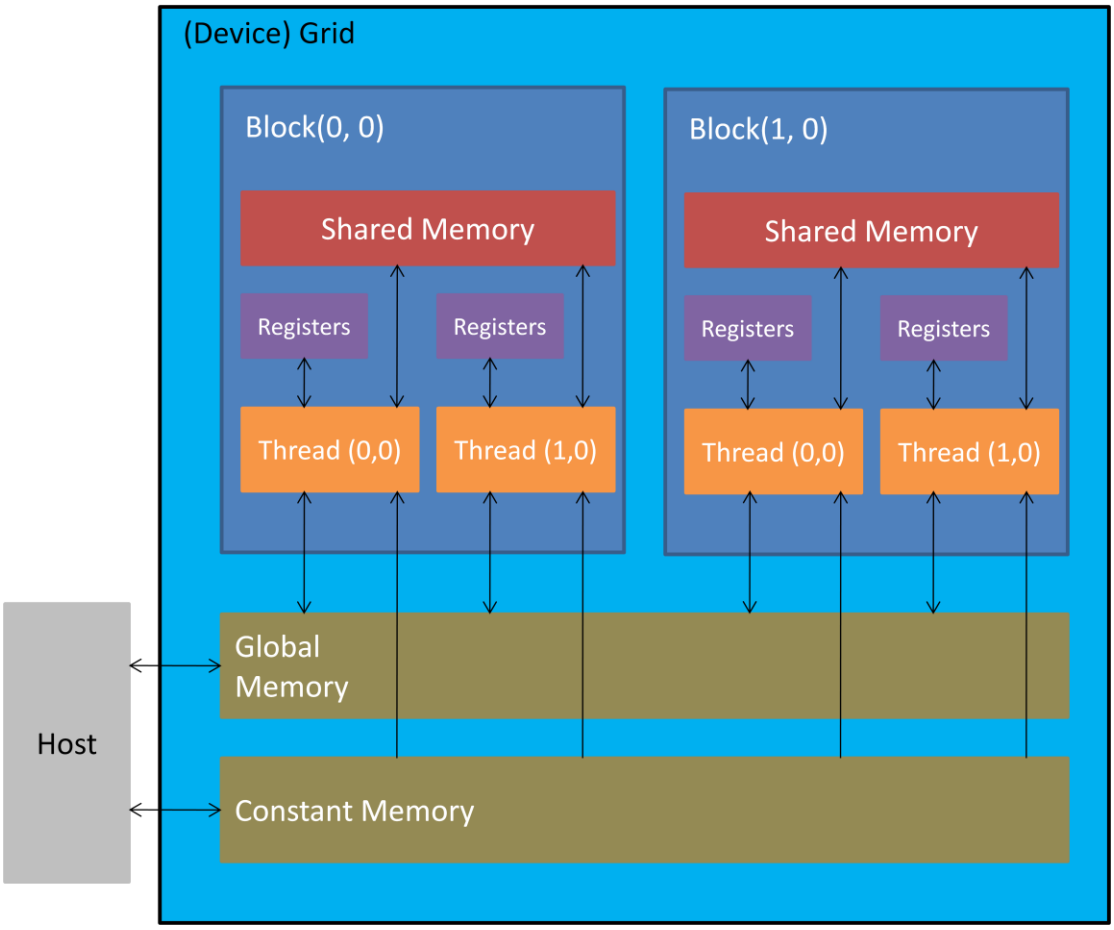


Figure 2.8 – Overview of the CUDA device memory model [11]

2.4 Review of GPU Computing

Since the GPGPU developed, a wide variety of applications implemented with GPGPU have obtained significant performance speedup. The following gives a review of GPU computing in different areas.

Linear algebra: Krüger and Westermann [14] presented a framework for solving linear algebra problems on GPU, which proposed a stream model for arithmetic operations on vectors and matrices. Gallapo et al. [15] presented an algorithm to efficiently solve dense linear systems using GPU. They stored the data as two-dimensional representation corresponding to the data layout on the GPU, and also supporting parallel data transfer of row and column swapping. With the fast development of GPU, developers can build a multiple GPUs for implementing the software applications. Quintana-Ortí et al. [16] implemented highly efficient matrix factorizations by multiple GPUs systems. Their experimental results showed the performance speedup by using four NVIDIA G80 GPUs.

Graph algorithm: Harish and Narayanan [17] used the CUDA to implement parallel algorithm for accelerating a large graph algorithm which involves millions of vertices. Buluç et al. [18] implemented a recursive block Floyd-Warshall algorithm on GPU for solving path problems. Another computationally expensive operation such as graph component labeling is also implemented with CUDA to get the advantage of parallel model [19, 20].

Pattern analysis: Vincent and Eric [21] proposed a speedy and parallel k nearest neighbor (kNN) search implementation using a GPU. Traditionally, the kNN search is

slow because it requires many operations on calculating the distance between two points. They implemented the algorithm with CUDA and showed the computation time being much decreased, up to 120x faster than without CUDA implementation. Catanzaro et al. [22] used the GPU for Support Vector Machines (SVM) classification and training which archives up to 138x speedup over LIBSVM. Recently, Chiosa and Kolb [23] proposed a GPU based framework for solving clustering problems. They presented an efficient parallel algorithm, Multilevel mesh clustering, implemented on GPU and provided high quality clustering results.

Sequence alignment: In genomics research, sequence alignment is used to analyze genes and genomes. Trapnell et al. [24] implemented MUMmerGPU, a parallel sequence alignment program, on GPU. They showed that using GPU has a better performance than CPU in memory-intensive applications. After that, Trapnell and Schatz [25] implemented MUMmerGPU 2.0 which used a suffix tree based algorithm on GPUs and archived 13x performance speedup compared with the CPU counterpart.

Other applications have also shown significant performance speedup, such as differential equations [26, 27, 28], molecular modeling and simulation [29, 30], biomedical image analysis [31] and AES encryption [32]. A detail GPU survey has been done by Owens et al. [33] in 2007.

Recently, Senguta et al. [34] focused on developing sets of parallel primitives for the development of GPGPU applications. They developed a set of scan primitives implemented with CUDA. Using the scan primitives, they showed the performance speedup of quicksort and spare matrix-vector multiply. Later, they provided a

powerful parallel primitives library called CUDPP [35]. As the implementation of GPU based hot term extraction is mainly based on CUDPP. The following subsection shows the details of scan operation and stream compaction and the other CUDPP details will be discussed in the next chapter.

2.4.1 Parallel Scan Operation

The prefix-sum operation, also known as scan operation, is widely used in parallel applications with complex access requirements. There are two types of scan, namely, inclusive scan and exclusive scan. A new array is generated by the inclusive scan where each element j is the sum of all elements up to and including j; whereas the exclusive scan generates a new array where right shifting the inclusive scan result by one element and inserting the identity for the first element.

Definition: The inclusive scan operation takes a binary associative operator $\oplus$, and an array of n elements

$$[a_0, a_1, \ldots, a_{n-1}],$$

and returns the array

$$[a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1})].$$

For example, if $\oplus$ is addition, then the inclusive scan operation on the array

$$[2\ 0\ 5\ 1\ 3\ 8\ 2],$$

returns

$$[2\ 2\ 7\ 8\ 11\ 19\ 21].$$

Definition: The exclusive scan operation takes a binary associative operator $\oplus$ with

identity I, and an array of n elements

$$[a_0, a_1, \ldots, a_{n-1}],$$

and returns the array

$$[I, a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-2})].$$

For example, if $\oplus$ is addition, then the exclusive scan operation on the array

$$[2\ 0\ 5\ 1\ 3\ 8\ 2],$$

returns

$$[0\ 2\ 2\ 7\ 8\ 11\ 19].$$

On the GPU, the earliest implementation of scan was used for “non-uniform stream compaction” [36] which is a part of a collision detection application. And then, Hensley et al. [37] improved the implementations and applied to summed-area table generation, but the time complexity was O(nlogn). Recently, Harris et al. [38] introduced CUDA based implementation of scan and then Sengupta et al. [34] developed the CUDA based implementation of segmented scan by reduce and down-sweep steps with the overall work complexity as O(n).

Beside the scan operation, segmented scan plays an important role in GPU based hot term extraction framework. Segmented scan operates as parallel scans on arbitrary partitions (“segments”) of the input vector. Considering a stream with 1/0 flag where 1 represents the first element of a segment, and then the scan is performed in each segment. An example is shown in Figure 2.9, an inclusive opertaion is selected to perform in the segmented scan.

Input Vector

2	3	5	3	4	2	1	1

Segment Flag

1	0	1	0	0	0	1	0

Output

2	5	5	8	12	14	1	2

Figure 2.9 – Example of inclusive segmented scan

2.4.2 Stream Compaction

Stream compaction, also known as stream reduction, is an important parallel primitive in a variety of general-purpose applications, such as GPU-based collision detection [39] and sparse matrix compression. Stream compaction can be used to remove unwanted elements in sparse data. It allows highly parallel algorithms to maintain performance over further processing steps and reduces overall memory usage.

In single processor machine, the implementation of stream compaction is a sequential algorithm. The algorithm is shown in Table 2.1, with the valid element of input moved to output.

```
Int j=0;
for (int i=0; i<N; i++){
        if (input[i] is valid){
                output[j] = input[i];
                j++;
        }
}
```

Table 2.1 – Coding of stream compaction in sequential algorithm

The efficiency of parallel algorithms is much better than sequential algorithms. The implementation of parallel algorithms is based on performing a parallel exclusive prefix sum operation [40]. Considering a stream with 1/0 flag where 1 represents a valid input and 0 represents an invalid input, the prefix sum is performed on the stream and the result is used to move each valid input to the new location of output. The detail is shown in Figure 2.10.

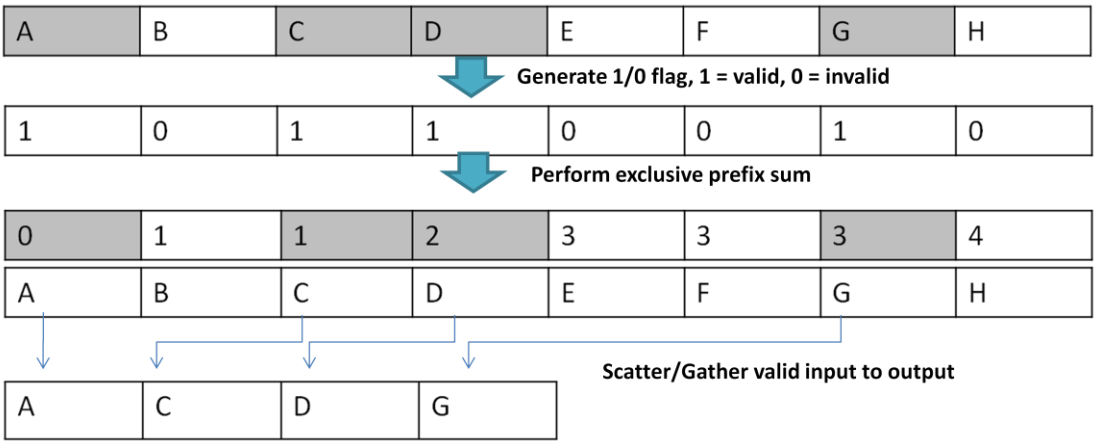


Figure 2.10 – Example of stream compaction

Horn [36] implemented stream compactions without scatter steps, since the GPUs in 2005 lack the support for random write access to memory (scattering). Therefore, a solution was to use gathering where a binary search is performed to find the valid input corresponding to output. The gathering operation was fairly expensive and required more memory usage, making the overall complexity as O(N logN).

Recently, GPUs support scattering, so that it can be used to replace the gathering operation [36]. CUDPP library provides the implementation of this approach and the stream compaction can achieve the overall complexity of O(N).

2.5 Conclusion

In this chapter, the graphics processing unit (GPU) is reviewed. GPU is designed as a parallel architecture for executing a huge amount of data. As the parallel architecture brings a significant performance speedup compared with CPU, many researchers successfully attained higher performance by implementing different algorithms on GPU. Moreover, NVIDIA introduced the CUDA which has a good design of programming model and memory model. Programmers can easily write CUDA programs and define the size of block and grid for running kernel on the GPU. On the other hand, an open source – CUDPP provides a powerful parallel primitives library. Programmers can easily use the library to perform scan operation, stream compaction and other operations.

Chapter 3

Borrow-Bit Sorting Algorithm

3.1 Introduction

Sorting is a general research problem in computer science. It is a core part of numerous algorithms whose performance depends on the efficiency of the sorting algorithm. For example, some of the clustering algorithms use sorting for finding the closest k^{th} elements. Sorting has been used in different areas of computer applications and it is an internal database operation. Therefore, any application can use the database benefiting from an efficient sorting algorithm. Other applications such as computer graphics, search engine, computational biology also involve sorting.

In this chapter, a Borrow-Bit sorting approach is introduced. It aims at speeding up the processing time of sorting key-value pairs by using the function provided in Java called Arrays.sort. Also, it has a flexible design for sorting key-value pair. Both keys and values are not limited to unsigned integer type (32-bit). In other words, either keys or values can be more than 32-bit. The principle of this approach is that the sum of bits in each pair of key and value must be within 64-bit.

3.1.1 Practical Background in GPU Sorting

NVIDIA has provided different CUDA toolkit version for developers. From CUDA 3.0 version, it supports CUDPP version 1.1.1 which was released in April 2010. However, the radix sort [41] in CUDPP version 1.1.1 only supports 32-bit key-value pairs sorting. The programming language in CUDPP and CUDA uses C/C++, and JCUDA [42] which is

a Java binding for NVIDIA CUDA and CUDPP. Therefore, java programmers can use JCUDA to implement the program in GPU.

In May 2011, CUDA 4.0 was released but it does not support CUDPP anymore. Instead, another high-level interface for GPU programming called Thrust [43] was used. Thrust is also written in C++ programming language. Moreover, the radix sort [44] implemented in Thrust is much faster than the radix sort in CUDPP. The CUDPP was updated to version 2.0 in August 2011. CUDPP now uses the radix sort of Thrust and it can also support 64-bit key-value pairs sorting. However, up till now, the JUDA has not been updated. Therefore, the JCUDA users are still not able to use the enhanced GPU sorting.

In the next chapter, the GPU based hot term extraction algorithm will be introduced. The algorithm is implemented with JCUDA. Since the JCUDA does not support 64-bit key-value sorting, the Borrow-Bit sorting algorithm can help to solve this limitation.

3.2 Borrow-Bit Sorting

3.2.1 Introduction

Java provides a sorting function, Arrays.sort, for sorting a specified range of array into ascending numerical order. Using a specified comparator, Arrays.sort can sort a specified array of objects. This sorting algorithm is a modified mergesort and offers guaranteed n*log(n) performance [45]. To use Arrays.sort for sorting key-value pairs order by value, the array of objects can be set as key-value pairs and the comparator is set to do the comparison order by value. Detailed coding is shown in Table 3.1. However, this sorting method is inefficient.

```
Arrays.sort(inputSequence, new Comparator<ExampleObject>()){
        public int compare(ExampleObject o1, ExampleObject o2)
        {
                return (o1.getValue() - o2.getValue ());
        }
});
```

Table 3.1 – Coding of using Arrays.sort for key-value pairs (order by value)

Adrian Marriott [46] demonstrated that the Arrays.sort was not fast enough and presented a stable sorting algorithm, Pigeon-Hole Sort, which got a better performance. Pigeon-Hole Sort passes through the input sequence for four times which take a time proportional to O(n) and the overall time complexity proportional to O(n+k) where k is the maximum value of input sequence.

In order to improve the performance of Arrays.sort for sorting key-value pairs, a Borrow-Bit Sorting approach is presented. Borrow-Bit Sorting can give a better performance, in term of speed, than Arrays.sort and Pigeon-Hole Sort. The computational complexity of Borrow-Bit Sorting is O(nlogn). In the Borrow-Bit Sorting approach, bitwise and bit shifts operations are mainly used. Bitwise operation operates one or more bit patterns of the numerals' individual bits. The bitwise operations include AND, OR, NOT and XOR, AND and OR operations to be used in the Borrow-Bit Sorting approach.

Bitwise AND operates in two equal length of binary numerals and performs the logical AND operation on each pair of corresponding bits. The result is 1 if both bits are 1 in a pair of bits, otherwise the result is 0.

```
    1101
AND 0101
  = 0101
```

Bitwise OR operates in two equal length of binary numerals and performs the logical OR operation on each pair of corresponding bits. The result is 0 if both bits are 0 in a pair of bits, otherwise the result is 1.

```
   1001
OR 0101
   1101
```

Bit shifts operation operates one or more bit patterns with left-shift or right-shift on the binary representation of integer value.

00101001 **Left-shift 1 bit**

= 01010010

00101001 **Right-shift 1 bit**

= 00010100

3.2.2 CPU Borrow-Bit Sorting

The Java based Borrow-Bit Sorting approach aims to use Arrays.sort(long [] input) to sort the key-value pairs order by value. The main idea is firstly, to combine two unsigned integer types (32-bit) of key and value into an unsigned long type (64-bit) of variable; then to use the Arrays.sort to sort the long type variable; and lastly, to split the long type variable into sorted value and the corresponding key. The pseudo-code of the Borrow-Bit Sorting approach is shown in Table 3.2.

```
Function BorrowBitSort(){
Input:
   Integer Array value, Integer Array key
Result:
   Integer Array with sorted value and Integer Array with corresponding key
Begin:
   Create a Long type array longValue, which array size is same as value
   Loop {
         Assign value to the left most 32-bit of longValue
         Assign the corresponding key to the right most 32-bit of longValue
   }
   Call Arrays.sort to sort longValue
   For each longValue {
         Split right most 32-bit and then assign to key
         Right-shift 32-bit of longValue and then assign to value
   }
End;
}
```

Table 3.2 – Pseudo-code of Borrow-Bit Sorting

There are three main steps in the Borrow-Bit Sorting approach. Firstly, it is to assign **value** to the leftmost 32-bit of the longValue and to assign the corresponding **key** to the rightmost 32-bit of the longValue. The structure of the longValue is shown in

Figure 3.1. Secondly, it is to use Arrays.sort to sort the longValue array into ascending numerical order.

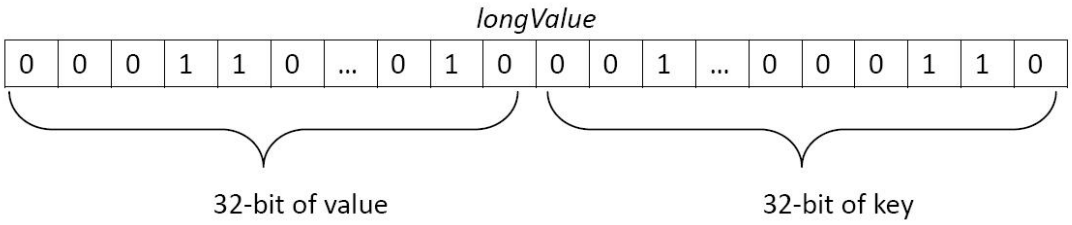


Figure 3.1 – Data structure of longValue

As the **value** is placed on the first 32-bit of longValue, the ascending numerical order of longValue is sorted according to the **value**. Lastly, it is to split the longValue and then assign back with respective **key** and **value**. In splitting longValue to form the **key**, bitwise AND operation is used. Creating a 31-bit integer where each bit is 1 and then using bitwise AND operation with the longValue, the result gets the 32-bit of **key**, the step is shown in Figure 3.2. On the other hand, in splitting longValue to form the **value**, right-shift operation is used. Simply right-shift 32-bit of longValue can form the **value.**

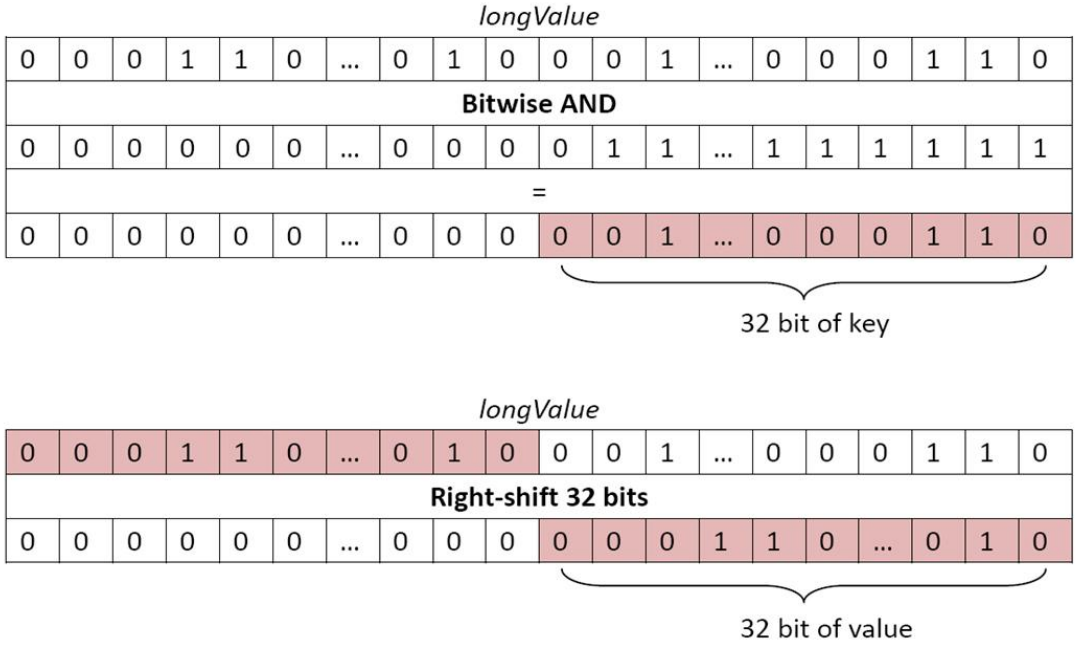


Figure 3.2 – Splitting key and value from longValue

3.2.3 CPU Borrow-Bit Extend Sorting

The Borrow-Bit Sorting approach can sort the key-value pairs order by value. However, both the key and value must be unsigned integer type, which is thus 32-bit. The extended approach can sort key-value pairs where both key and value can be more than 32-bit. Different from the Borrow-Bit Sorting approach, this extended approach uses an index which starts from one and the increment is one by one to form a key. After finishing the sorting process, the original key can be retrieved by the sorted index. The main idea of this approach is the unsigned key does not occupy all 32 bits which also means the total number of key-value pairs is less than 2^{31}, so that value can be borrowed from the unoccupied bits. The prerequisite of this approach is that the key must have enough bits for being lent to value. Thus, the total number of key-value pairs are smaller than 2^{31} and the key has enough bit lent to the value. The advantage of this approach is the capability to sort by value which is more than 32 bits, and the value of key can also be more than 32 bits as it does not pass into the function by using an index to represent the key. The computational complexity of Borrow-Bit Extend Sorting is O(nlogn). The pseudo-code of the Borrow-Bit Extend Sorting approach is stated in Table 3.3. Figure 3.3 demonstrates an example with the value borrows 4 bits from the key.

```
Function BorrowBitExtendSort(){
Input:
   Long Array value, Integer x (indicate how many bits of value have to borrow from key)
Result:
   Integer Array with sorted value and Integer Array with sortOrder
Begin:
   Create a Long type array longValue, which size is same as value
   Create a Integer type array sortOrder, which size is same as value
   Initialize an integer variable index, start from one.
   Loop {
         Assign index to longValue which occupy the right most (32-x) bit
         Assign value to longValue which occupy the left most (32+x) bit
         Increment index by one
    }
   Call Arrays.sort to sort longValue
   For each longValue {
         Split right most (32-x) bits and then assign to sortOrder
         Right shift (32-x) bits of longValue and then assign to value
   }
End;
}
```

Table 3.3 – Pseudo-code of Borrow-Bit Extend Sorting

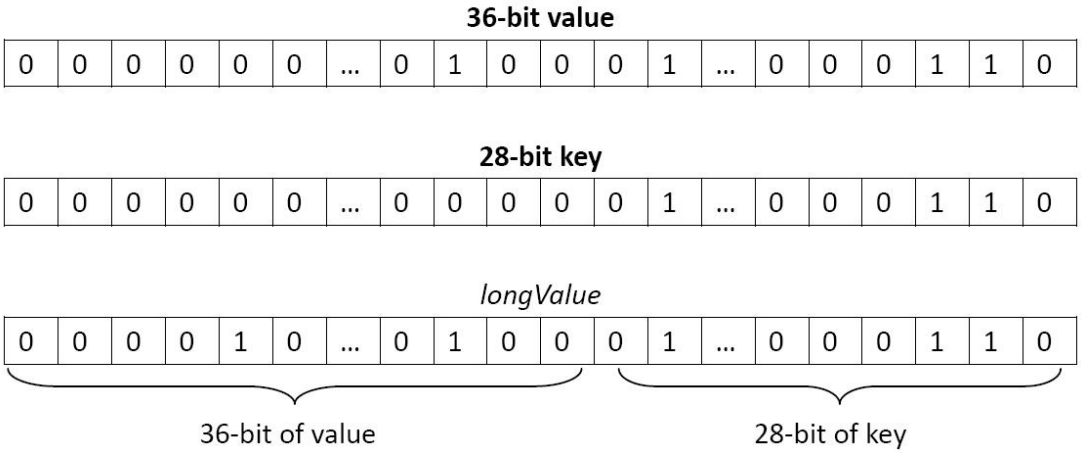


Figure 3.3 – Example of longValue (4 bits of value borrow from key)

3.3 CUDPP Borrow-Bit Radix Sort

In the CUDPP version 1.1.1, the CUDPP radix sort does not provide a function to sort a long type value. It can only sort the unsigned integer type of key-value pairs. The Borrow-Bit sorting algorithm can be applied to the CUDPP radix sort. CUDPP Borrow-Bit Radix Sort can sort key-value pairs where both key and value can be more than 32 bits.

Figure 3.4 shows the example of CUDPP Borrow-Bit Radix sort that the value has to borrow 4 bits from key. The core part of CUDPP Borrow-Bit Radix Sort is to firstly generate key and value; and then to sort the key-value pairs order by the key first, followed by the value; finally, to recover the key and value. The pseudo-code of CUDPP Borrow-Bit Radix Sort is shown in Table 3.4. One important point is that only one time of transferring the input data from host CPU to GPU memory is required in the first cudppSort() function. In the second cudppSort(), it just uses the GPU memory data so that it does not need to transfer the result of first cudppSort() from GPU memory to CPU and then from CPU back to GPU memory.

```
Function CUDPPBorrowBitRadixSort(){
Assumption:
   The key have enough bits lend to value.
Input:
   Long Array value, Integer x (indicate how many bits of value have to borrow from key)
Result:
   Integer Array with sorted value and Integer Array with index
Begin:
   Create an Integer type array index which size is same as value
   Initialize an integer array index, starting from 1, each value is incremented by 1.
   intValue = keyValueGenerator(index, value);
   cudppSort(index, intValue);
   cudppSort(intValue, index);
   value = recover(index, intValue);
End;

function keyValueGenerator(index, value){
   Create an Integer type array intValue which size is same as value.
   For each value{
      Split right most x bits and then replace the bits in index (start from left 2nd)
      Right shift x bits of value assign to intValue
   }
   return intValue;
}

function recover(index, intValue){
   Create an long type array longValue which size is same as intValuen
   For each intValue{
      Assign intValue into longValue and then left shift x bits
      Get x bits of index (start from left 2nd)` and then assign to the right most x bits of longValue
   }
   return value;
}
}
```

Table 3.4 – Pseudo-code of CUDPP Borrow-Bit Radix Sort

Original key-value pair	
Key	Value
00000000000000000000000000000001	01110101001111101010011111010100**1010**
00000000000000000000000000000010	00110111001101101010011111010100**1110**
00000000000000000000000000000011	01111111001111101010011111010100**0000**
00000000000000000000000000000100	01000111001100101010011111010100**0100**
00000000000000000000000000000101	01001110001111101010011111010100**0001**
00000000000000000000000000000110	01100011111111101010011111010100**0110**
00000000000000000000000000000111	01110001001111101010011111010101**1001**
00000000000000000000000000001000	00010111001111101010011111010100**0111**
00000000000000000000000000001001	01001111000011101010011101010100**1100**
00000000000000000000000000001010	01100001001111101010011111010101**1001**

keyValueGenerator	
Key	Value
0**1010**00000000000000000000000000001	01110101001111101010011111010100
0**1110**00000000000000000000000000010	00110111001101101010011111010100
0**0000**00000000000000000000000000011	01111111001111101010011111010100
0**0100**00000000000000000000000000100	01000111001100101010011111010100
0**0001**00000000000000000000000000101	01001110001111101010011111010100
0**0110**00000000000000000000000000110	01100011111111101010011111010100
0**1001**00000000000000000000000000111	01110001001111101010011111010101
0**0111**00000000000000000000000001000	00010111001111101010011111010100
0**1100**00000000000000000000000001001	01001111000011101010011101010100
0**1001**00000000000000000000000001010	01100001001111101010011111010101

CUDPP sort by key	
Key	Value
00000000000000000000000000000011	01111111001111101010011111010100
00001000000000000000000000000101	01001110001111101010011111010100
00100000000000000000000000000100	01000111001100101010011111010100
00110000000000000000000000000110	01100011111111101010011111010100
00111000000000000000000000001000	00010111001111101010011111010100
01001000000000000000000000000111	01110001001111101010011111010101
01001000000000000000000000001010	01100001001111101010011111010101
01010000000000000000000000000001	01110101001111101010011111010100
01100000000000000000000000001001	01001111000011101010011101010100

01110000000000000000000000000010	00110111001101101010011111010100

CUDPP sort by Value	
Key	Value
00111000000000000000000000001000	00010111001111101010011111010100
01110000000000000000000000000010	00110111001101101010011111010100
00100000000000000000000000000100	01000111001100101010011111010100
00001000000000000000000000000101	01001110001111101010011111010100
01100000000000000000000000001001	01001111000011101010011101010100
01001000000000000000000000001010	01100001001111101010011111010101
00110000000000000000000000000110	01100011111111101010011111010100
01001000000000000000000000000111	01110001001111101010011111010101
01010000000000000000000000000001	01110101001111101010011111010100
00000000000000000000000000000011	01111111001111101010011111010100

Final Step - recover	
Key	Value
00000000000000000000000000001000	00010111001111101010011111010100011 1
00000000000000000000000000000010	00110111001101101010011111010100111 0
00000000000000000000000000000100	01000111001100101010011111010100010 0
00000000000000000000000000000101	01001110001111101010011111010100000 1
00000000000000000000000000001001	01001111000011101010011101010100110 0
00000000000000000000000000001010	01100001001111101010011111010101100 1
00000000000000000000000000000110	01100011111111101010011111010100011 0
00000000000000000000000000000111	01110001001111101010011111010101100 1
00000000000000000000000000000001	01110101001111101010011111010100101 0
00000000000000000000000000000011	01111111001111101010011111010100000 0

Figure 3.4 – Example of CUDPP Borrow-Bit Radix Sort

3.4 Experimental Results

In this section, various experimental results obtained by using different sorting approaches for key-value pairs are reported. The experimental platform is an AMD Athlon™ 64 Processor 3000+ 2.29GHz machine with 3GB of memory and the graphic card is GeForce GT 240, which has 12 multiprocessors and 96 CUDA cores with 1 GB memory. The programming implementation uses the version of CUDA 3.2, CUDPP 1.1.1 and JCDUA 0.3.2a.

In order to compare the CPU and GPU sorting in real and practical case, the experimental results show the execution time of key-value pairs sorting with CPU and GPU. The GPU execution time includes the time of transferring input data from host CPU to GPU's on-board memory.

Figure 3.5 shows the execution time of sorting key-value pairs, both keys and values are unsigned integer type (32-bit), by Arrays.sort, PigeonHole Sort, CPU Borrow-Bit sorting and CUDPP radix sort. The input values were randomly generated and the size ranged from 1K to 50M. By comparing the sorting result run by CPU (Arrays.sort, PigeonHole Sort and CPU Borrow-Bit sorting), the processing time of CPU Borrow-Bit sorting is the fastest and the memory usage is the smallest. Out of memory will occur if the total number of elements is larger than 2 million in Arrays.sort and PigeonHole Sort. By comparing the sorting results using CPU and GPU, the execution time of CPU borrow-bit sorting is the fastest if the total number of elements is less than 10K. It is acceptable that the overhead of GPU is large when transferring the input data from host CPU to GPU's on-board memory. The performance of CUDPP radix sort is much

better than other CPU's sorting approaches if the total number of element is large enough. GPU is 12x faster than CPU when the total number of element is 50 million.

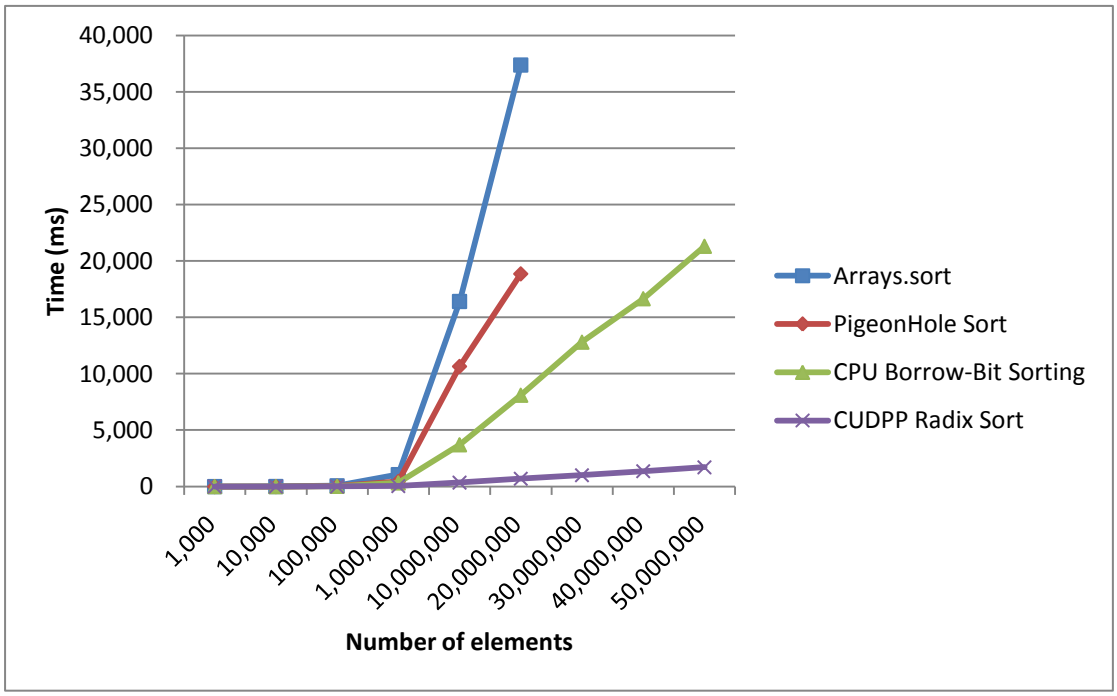


Figure 3.5 – Execution time in Arrays.sort, PigeonHole Sort, CPU Borrow-Bit Sorting and CUDPP Radix Sort

Figure 3.6 shows the execution time of sorting key-value pairs by CPU Borrow-Bit Extend Sorting and CUDPP Borrow-Bit radix sort. In this experiment, each value borrows 4 bits from the corresponding key. As a result, CUDPP Borrow-Bit radix sort is 4x faster than CPU Borrow-Bit Extend Sorting.

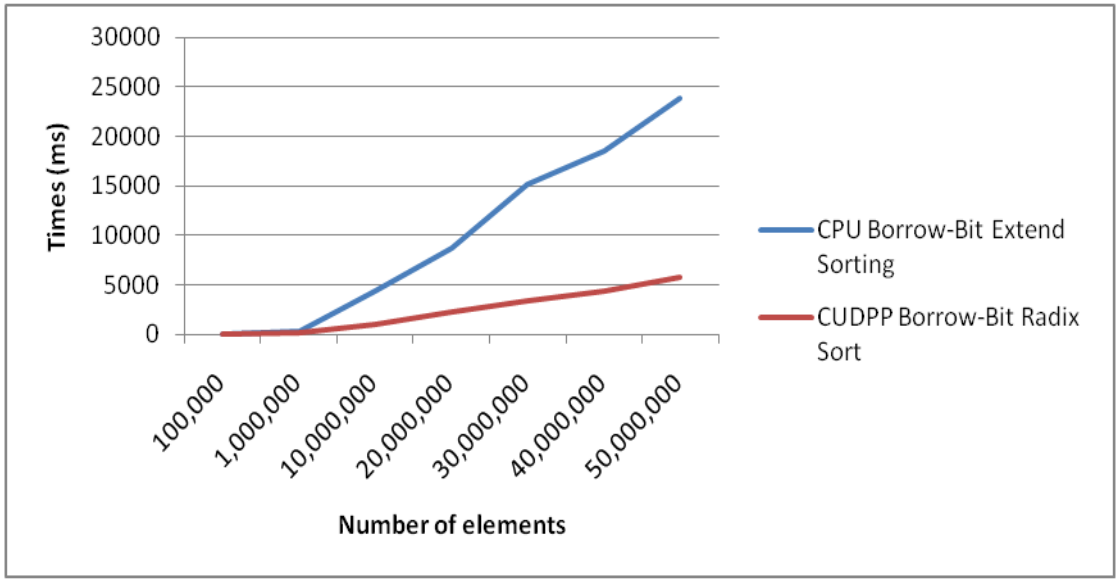


Figure 3.6 – Execution time in CPU Borrow-Bit Extend Sorting and CUDPP Borrow-Bit Radix Sort

Figure 3.7 and Figure 3.8 report the execution time of sorting key-value pairs where both keys and values are unsigned integer type (32-bit) by CPU Borrow-Bit Sorting and CUPDD radix sort with different number of input elements as well as different random ranges in generating input elements. In different random ranges of input elements, we can measure the performance in different sparsity of data. CPU Borrow-Bit Sorting approach requires more execution time for sorting larger random ranges of value. On the other hand, the execution time is steady in CUDPP radix sort. Therefore, CUDPP radix sort has a more stable performance.

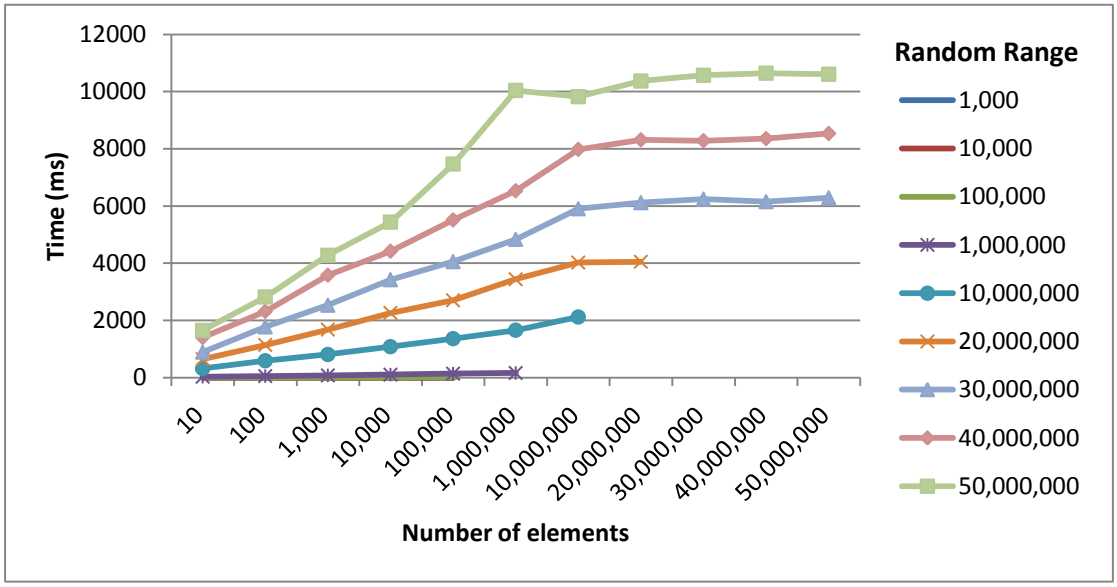


Figure 3.7 – Execution time in CPU Borrow-Bit Sorting with different number of elements and different random ranges

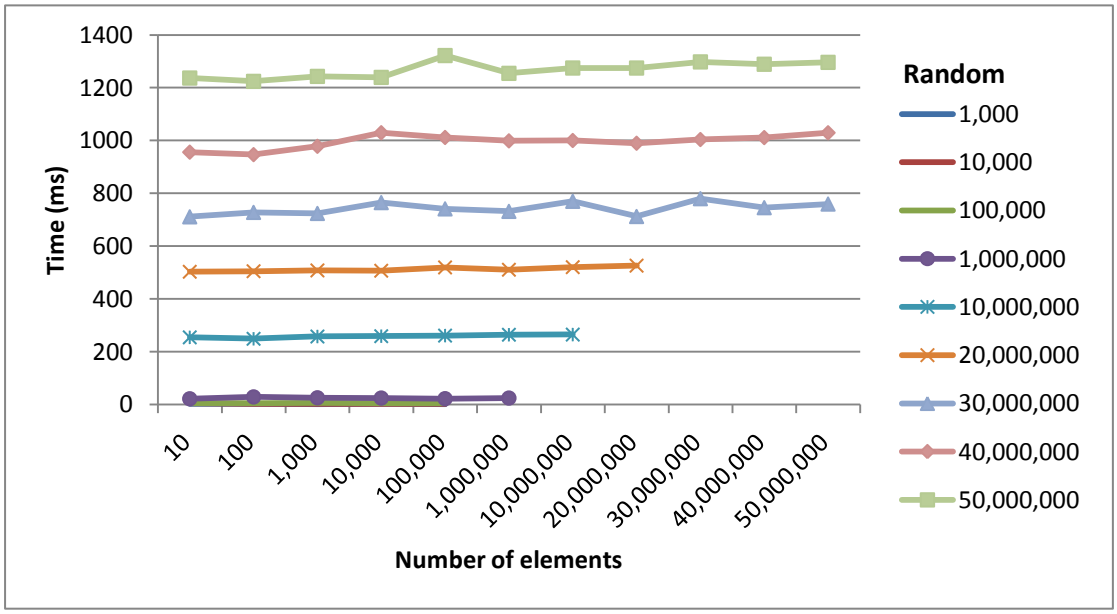


Figure 3.8 – Execution time in CUDPP Radix Sort with different number of elements and different random ranges

3.5 Conclusion

In this chapter, the Borrow-Bit sorting approach is presented. Applying Borrow-Bit sorting approach into Arrays.sort for sorting key-value pairs is much better than the original Arrays.sort which uses Comparator. Although the Borrow-Bit sorting approach may not cover all the sorting problems, it can speed up the processing time of users who simply use the Java's Arrays.sort in their program implementations.

Also, using the Borrow-Bit sorting approach, we can solve the limitation of the CUDPP 1.1.1 version by using CUPDD Borrow-Bit radix sort to sort the key-value pairs where both key and value can be more than 32-bit. For the prerequisites, the total number of key-value pairs should be smaller than 2^{31} and the key should have enough bits lent to the value. Although, the latest version of CUDPP provides a complete sorting implementation which supports 64-bit key-value pairs, the JCUDA does not support it, therefore some JAVA programmers cannot benefit from the latest version of CUDPP while the CUDPP Borrow-Bit radix sort can still contribute to some practical cases.

Chapter 4

Hot Term Extraction Analysis

4.1 Introduction

Nowadays, social media has become part of our life. People can easily share their knowledge and opinions on the Web through different channels such as forums, blogs, online news and newsgroups, which are called Consumer Generated Media (CGM). CGM is the major source of digital Word of Mouth (WOM). There is no standard format for the content of CGM. Users freely type in their opinions on the Web.

WOM refers to oral communication and the transferring of information from one person to another [47]. From the marketing view point, Word of Mouth Marketing (WOMM) refers to people who use WOM to promote their business. In digital marketing, WOM is like a personal recommendation that customers share their satisfaction or dissatisfaction over the product or service in CGM, which will affect other users' decision when they search the related information in CGM. In a recent WOM study conducted by Google [48], 94% of WOM conversations are predominant and the search activity impacts more than 15% WOM conversations.

People usually search information by web search engines. When they input a query, usually a list of ranked documents according to the keyword of the query will appear. It is not easy for the users to figure out the most relevant information among the long list of search results. Besides, some hidden keywords not included in the query

may also be useful to the users. Hot term extraction can list out the most relevant terms and thus can provide WOM insight to users.

In this chapter, a hot term is defined as a word or a bi-gram Chinese character appearing in at least a certain percentage of documents. Based on the frequency of the term occurring in the documents which are searched by query, we can say that the more frequent the term appears, the more related it is to the query. Some hidden information may also be useful for users. In view of the huge amount of search results in a real time application, a fast hot term extraction approach is required. By taking the advantage of the parallelism characteristic of GPU, a GPU based hot term extraction framework will be presented in the following sections.

Since WOMM is closely related to social media, some social media tools have been developed in the IT industry. The tools can help clients to gain marketing benefits and insights from digital WOM. An overview of the social medial tools in the industry will be presented in the next section.

4.2 Social medial tools in industry

There exist some companies which develop social medial analysis tools to provide a WOM monitoring platform. Some of the functions provided by the tools include:

- CGM search engine - users can type in keywords to search the related CGM posts.
- Sentiment analysis - giving positive, negative or neutral score of each post.
- Hot topic discovery - showing the hottest topic in the search results.
- Charting - showing the distribution of posts in CGM and the daily post trend within the search period.
- Dashboard – showing insights and analytics summary.
- Hot Term Analysis – showing the most frequent term.

K-Matrix Digital Intelligence Ltd. [5]: It is a digital intelligence solution company. It provides comprehensive solutions to assist companies in adapting to the fast growing Internet and gaining the most digital communication and marketing benefits and insights from the digital community [5].

k-matrix CI is one of the self developed applications by K-Matrix Digital Intelligence Ltd. It is a web based application providing real time intelligence and monitoring platform of WOM in CGM. k-matrix CI collects around 3 million posts daily from different CGM web sites, including blogs, micro-blogs, forums, online news and newsgroup. Up to July 2011, the database of k-matrix CI has collected 2.1 billion posts. Having a huge amount of data, k-matrix CI can help clients to monitor the WOM of their brand or enterprise. It can also give insights to the consumers to take a faster step in managing their digital marketing.

CIC [49]: It is a social business intelligence company. CIC enables businesses to fully leverage the power of social media and Internet word of mouth (IWOM) intelligence across the organization [49]. It provides solutions to assist companies in meeting their needs for social media marketing and social business. IWOMmaster is a web based application developed by CIC. It focuses on the IWOM in the Mainland and provides a monitoring platform to clients. According to the CIC official website, it collects over 100 million posts every month from different CGM websites in the Mainland and currently it has already collected over 1 billion consumer comments.

Visible Technologies [50]: It is a social media solution company. It provides both software and services to assist companies in getting benefits from social communities. Visible Intelligence^{TM} is a social media platform which helps clients to navigate the social web, engage with dynamic communities and discover insights [50]. It provides different types of data source such as social networking sites (e.g. Facebook), micro blogs (e.g. Twitter), video sites (e.g. YouTube), forums, blogs, news and etc.

Since WOM is important for business marketing, many companies have been developing social media monitoring tools [51, 52, 53, 54]. However, only a few of them include the hot term analysis feature.

4.3 Review of Hot Term Extraction

There are some similar related works about hot term extraction algorithms. Also, there are various extraction methods and purposes in different algorithms.

Frequent word sequence extraction

The frequent word sequence extraction discovers some interesting sequences by finding maximal frequent sequences in the document collection. According to Ahonen-Myka et al. [4, 55, 56, 57, 58], there are some definitions as regards maximal frequent sequence.

Definition 1: *A sequence with length k, $p = w_1 \ldots w_k$, is a subsequence of a sequence q if all the words w_i, $1 \leq i \leq k$, occur in q and they occur in the same order as in p. If a sequence p is a subsequence of a sequence q, we can also say that p occurs in q.*

Definition 2: *A sequence p is frequent in a set of sentences S if p is a subsequence of at least σ_{min} sentences of S, where σ_{min} is a given minimum frequency threshold.*

Definition 3: *A sequence p is a maximal frequent sequence in a set of sentences S if there does not exist any sequence p' in S such that p is a subsequence of p' and p' is frequent in S.*

Given a document collection and frequency threshold σ, if a sequence appears for more than σ times, the sequence is considered as frequent. Furthermore, a sequence is maximal if there is no other frequent sequences contain this sequence. The

following sentences are used in the example stated in [57]. If the minimum frequency threshold is set as 2, the maximal frequent sequence can be found as “congress retaliation against foreign unfair trade practices”.

1. The **Congress** subcommittee backed away from mandating specific **retaliation against foreign** countries for **unfair** foreign **trade practices**.
2. He urged **Congress** to reject provisions that would mandate U.S. **retaliation against foreign unfair trade practices**.
3. Washington charged France West Germany the U.K. Spain and the EC Commission with **unfair practices** on behalf of Airbus.

This extraction method is pretty flexible. Users can set different value of frequency threshold to find the frequent sequences. And also, the maximal frequent sequences can provide the most informative summarization to users. However, the main objective of this extraction method does not focus on the speed issue. The processing time of this extraction method is not fast enough in real time applications.

Text Clustering

Text clustering is one of the text mining techniques which can help users to organize document collections effectively. By applying text clustering to the search engine, the document results can be grouped into a number of clusters. Documents having high similarity with each other will be put in the same cluster. Users can easily browse the clustered documents which are related to the users’ query [3].

Beil et al. [59] introduced a text clustering approach which used frequent item (term)

sets. They presented two algorithms for frequent term-based text clustering, with one aimed for flat clustering while the other aimed for hierarchical clustering. As the general problems of text clustering are the large document size and high dimensionality of data, using the frequent term sets can reduce the dimensionality of the document vector space. They have used the technique of association rule mining to find the frequent term sets. Terms that appear in a certain number of common documents are formed into clusters. Therefore, documents which are grouped together into a cluster are about the same topic. Furthermore, P. Ponmuthuramalingam et al. [60] presented an effective term-based text clustering. They presented four algorithms which had higher F-measure value and better clustering quality than [59].

Yanjun et al. [61] proposed a text clustering algorithm named Clustering based on Frequent Word Sequences (CFWS). They used the suffix tree method to extract frequent word sequences and then used the frequent word sequences to perform clustering. They proved that it was more effective to use frequent word sequences in clustering than those algorithms which ignored the word sequences in documents.

Word Count

Recently, Google has introduced a software framework named MapReduce [62] which supports distributed computing on large scale data processing on clusters of computers. MapReduce provides highly efficient execution time on distributed computing and the scale of data is at least 1TB. The MapReduce programming model consists of two primitives which are *map* operation and *reduce* operation. The *map* operation takes input key-value pairs and produces a set of intermediate key-value

pairs. It groups all intermediate values with the same intermediate key and passes them to the reduce operation. The *reduce* operation collects the results in groups and merges together with the values of each group to form a possibly smaller set of values. Figure 4.1 shows the MapReduce programming model.

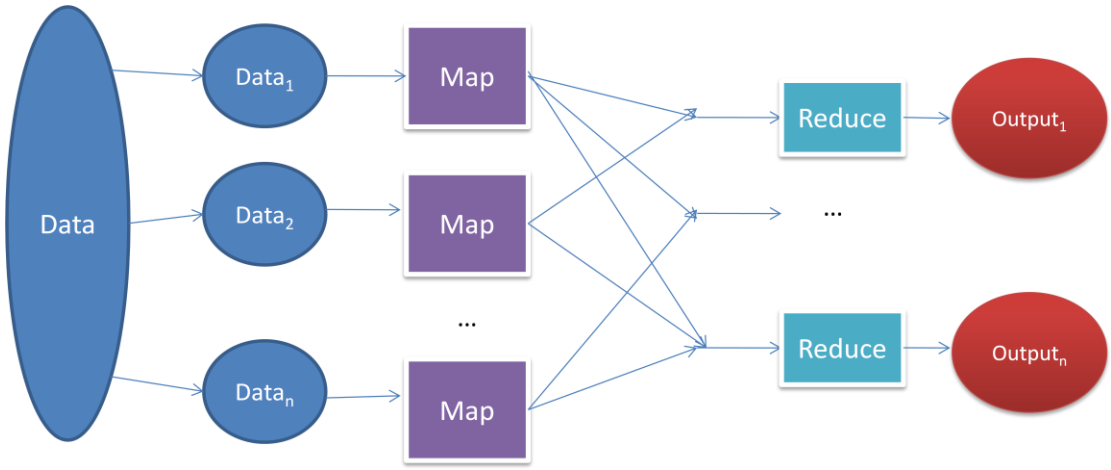


Figure 4.1 – MapReduce programming model

Word counting is a typical example of MapReduce. Given a set of words, the input key-value pairs represent the word and counter. The word is a key and the value is 1. The map operation groups all the input pairs which have the same key and the reduce operation sums up all the values of each group. Therefore, the frequency of each word is counted. The full example can be found in the tutorial provided by Hadoop [63].

Word Cloud

Word cloud or tag cloud is a visual representation of summarizing the content of websites or text documents. Typically, the font size of the words in word cloud represents the word frequency while the font color indicates other information. For

example, PubCloud [9] summarizes the search results from PubMed and the font size of the words indicates the word frequency and the font color indicates the recency of publications. Recently, Cui et al. [64] have presented a dynamic word cloud visualization method which aims to preserve the semantic coherence and spatial stability of the cloud. Since word cloud emphasizes the visualization of representation, it provides an effective representation for summarizing text content.

4.4 Hot Term Extraction - Preprocessing

The preprocessing phase aims to transform documents into a representation that can be used by our hot term extraction algorithm. As all the documents are assumed from CGM, there is no standard format for the contents which mainly include Chinese Characters, words, numbers, and punctuation. A term is defined as an English word or bi-gram Chinese Characters. The preprocessing phase is important as some frequent terms are useless and invalid and the hot term extraction algorithm can be made faster by filtering those terms. Therefore, some rules are set for defining a term. To ensure the validity of the Chinese term, a term must have two valid Unicode Chinese Characters which starts with the character code 4E00 [65]. In English term, the maximum length of pure alphabet terms is 30 characters [66]. For terms with numeric characters, the maximum length is 8 characters. Furthermore, if the documents are in some html formats or Bulletin Board Code [67] which is widely used in forums, all the tags have to be removed. Lastly, the minimum length of terms is 2 characters.

Before applying the hot term extraction algorithm, we need to give a unique term id to each term. All terms are saved in the database with a unique term id. At the same time, the frequency of each term is counted, so that the global term frequency is recognized. After the global frequency is recognized, most frequent terms and least frequent terms can be treated as stop words. Filtering of these terms is needed. By removing the stop words, the result will be more accurate and the processing time will be shortened.

4.5 CPU Based Hot Term Extraction

4.5.1 Algorithm

```
Function Main(){
Input:
   String of Keyword
Output:
   A List of Hot Term <Term Id, Frequency> with corresponding Doc Id List <Long>
Begin:
   List of Stop Word = Call Get_StopWord();
   Filtered Term Id and Filtered Doc Id = Call Search(Keyword)
   Sorted Index = Call Sorting(Filtered Term Id)
   Boundary = Call FindBoundary(Sorted Term Id)
   List of Term Frequency = Call TermFrequency(Boundary)
   Sorted List of Term Frequency = Call Sorting(List of Term Frequency)
   Output = Call Get_HotTerm();
End;
}
```

Table 4.1 – Pseudo-code of CPU based hot term extraction algorithm

Table 4.1 shows the pseudo-code of the CPU based hot term extraction algorithm. The CPU based hot term extraction algorithm aims to find the most frequent terms and the corresponding documents. When a user inputs a query, the search engine will return the related documents and each document will have different number of terms. Collecting all the terms and then calculating the term frequency, the number of most frequent terms can be found. In this algorithm, the calculation of term frequency is performed by sorting and finding boundary processes. The use of this approach for finding term frequency is mainly due to the huge number of terms. It can reduce memory usage and may not need to create an additional counter to

calculate all term frequencies. Moreover, since there is no counting method provided by GPU, using this approach can avoid finding term frequency in GPU based hot term extraction. With the use of the same approach in both CPU and GPU based hot term extraction algorithm, the experiment result can then be compared.

The CPU based hot term extraction algorithm consists of seven major steps. Firstly, getting stop words from database, it is used for filtering the terms after the search documents are produced. According to the query, the search engine will return related documents and any useless terms will be filtered at the same time. Thus, there remain the filtered term ids and the corresponding document ids. It then requires the counting of the term frequency and sorting of the filtered term ids. After the term ids are sorted, we have to find the boundary index of each term id. The boundary index refers to the last position of each sorted and unique term id. As the boundary index of each term is known, it is easy to find the term frequency of each term and they can be assigned to a list which contains an index and frequency of each term. After sorting the list in descending order by frequency, the most frequent terms are sorted on the top of the list. Lastly, the terms can be retrieved by using the index of the sorted list. The flow chart and an example of the CPU based hot term extraction algorithm are shown in Figure 4.2 and Figure 4.3 respectively.

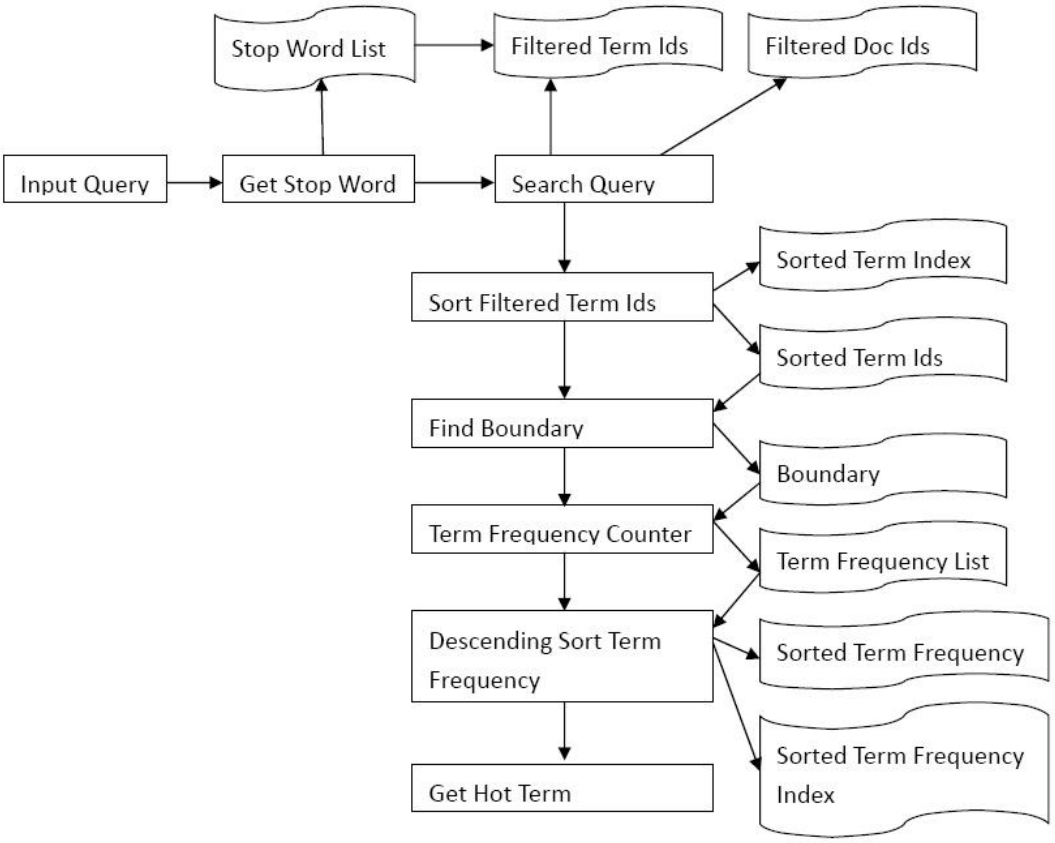


Figure 4.2 – Flow of CPU based hot term extraction

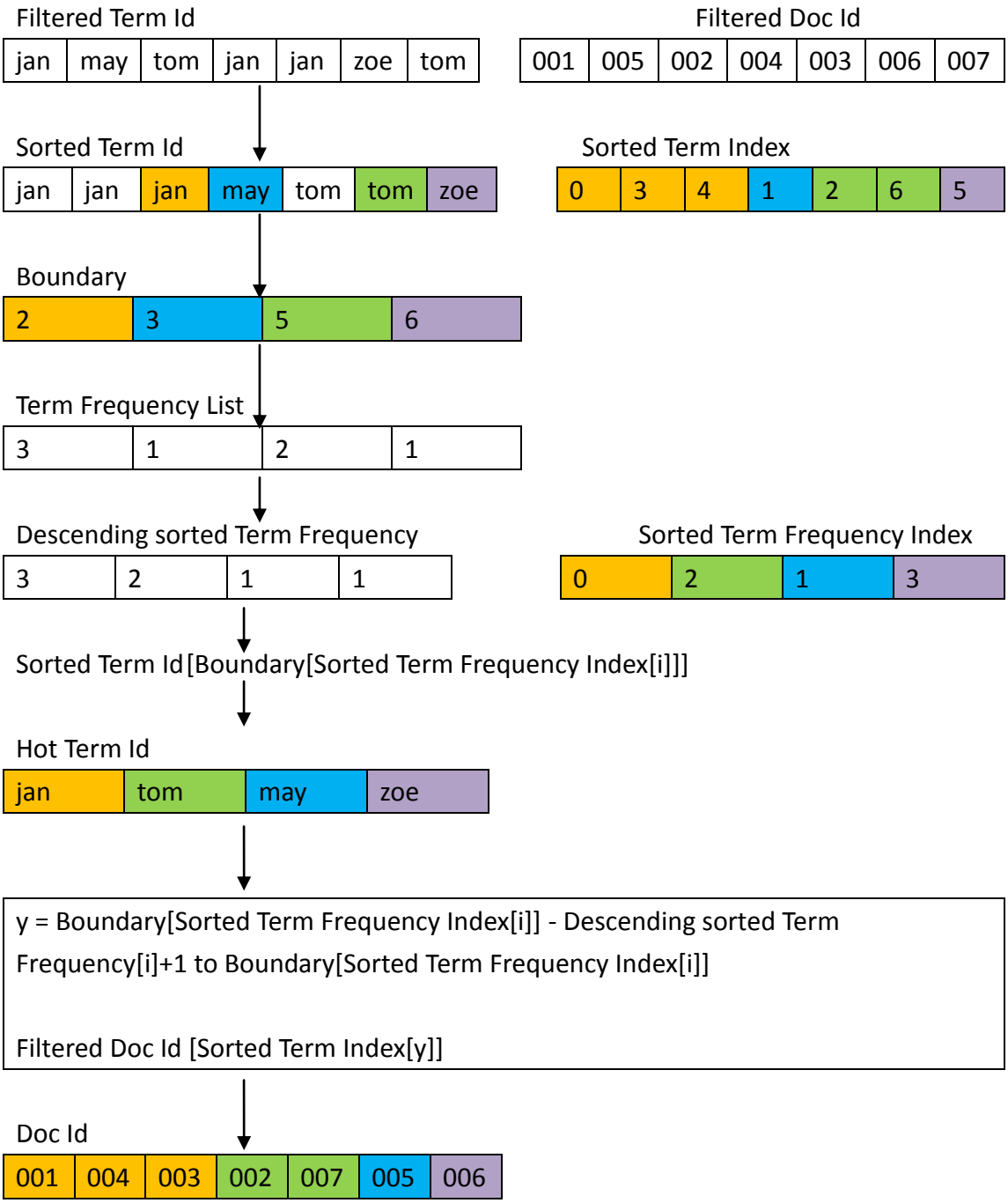


Figure 4.3 – Example of CPU based hot term extraction

4.5.2 Merge Sort

The CPU based hot term extraction algorithm has several steps. In the whole process, the stage of sorting filtered term ids occupies most of the processing time. In order to shorten the processing time, we may use the merge sort and multi-core processor characteristics to implement the sorting.

A multi-core processor has two or more independent processors that read and execute program instructions. Just like parallel programming, each processor executes instructions concurrently, which makes it more efficient than a single processor.

Merge sort was invented by John von Neumann [68]. It is based on a divide and conquer algorithm which divides the unsorted list into two sub-lists each at about half of the total size. The merge sort is then recursively applied to sort each sub-list. Lastly, the two sub-lists are merged into one sorted list. An example is shown in Figure 4.4.

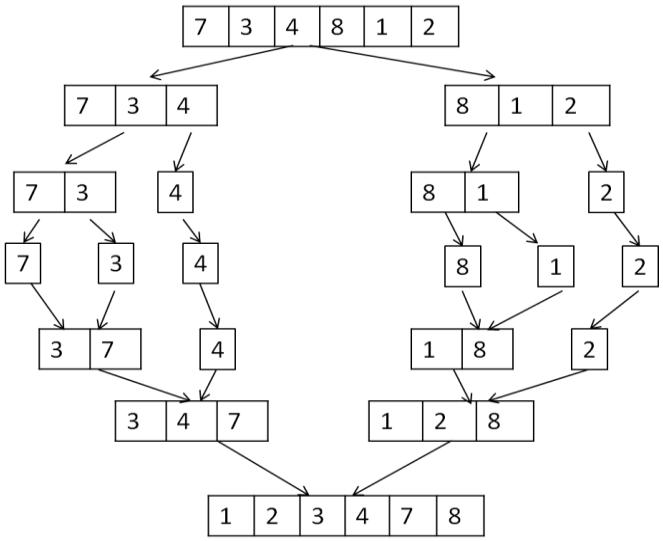


Figure 4.4 – Example of Merge Sort

In our hot term extraction algorithm, the filtered term ids are returned after searching the keyword. Depending on the number of cores of the processor, we can divide the term id list into N sub-lists with each at approximately equal size. Each sub-list independently runs Arrays.sort on a core of the processor. After all the sub-lists are sorted, they are merged into a sorted list. The implementation of merge part is the same as that of merge sort. The elements of 2 sub-lists are compared and a sorted list is then generated. The pseudo-code is shown in Table 4.2.

```
public static int[] merge(int[] number1, int[] number2) {
        long[] number3 = new long[number1.length + number2.length];

        int i = 0, j = 0, k = 0;
        while(i < number1.length && j < number2.length) {
            if(number1[i] <= number2[j])
                number3[k++] = number1[i++];
            else
                number3[k++] = number2[j++];
        }

        while(i < number1.length)
            number3[k++] = number1[i++];
        while(j < number2.length)
            number3[k++] = number2[j++];

        return number3;
}
```

Table 4.2 – Pseudo-code of merge sort

4.6 GPU Based Hot Term Extraction

4.6.1 Algorithm

```
Function Main(){
Input:
   String of Keyword
Output:
   A List of Hot Term <Term Id, Frequency> with corresponding Doc Id List <Long>
Begin:
   Term Id and Doc Id = Call Search(Keyword)
   List of Stop Word = Call Get_StopWord();
   Filtered term Id = Call GPU_FilterTermId(term id, Stop Word Flag)
   Sorted Index = Call GPU_Sort(Filtered term id)
   Boundary = Call GPU_FindBoundary(Sorted Term Id)
   List of Term Frequency = Call GPU_TermFrequency(Boundary)
   Sorted List of Term Frequency = Call GPU_Sort(List of Term Frequency)
   Output = Call Get_HotTerm();
End;
}
```

Table 4.3 – Pseudo-code of GPU based hot term extraction algorithm

Table 4.3 depicts the pseudo-code of our GPU based hot term extraction algorithm. Although its flow is the same as that of the CPU based counterpart, the implementation is designed for running on GPU. The computational complexity of GPU based hot term extraction is O(nlogn). The algorithm consists of four main GPU functions: binary search, parallel scan operation, stream compaction and sorting. The binary search and stream compaction can be used to filter the useless term id in GPU_FilterTermId function; by using parallel scan operation and stream compaction, the boundary and term frequency in GPU_FindBoundary and GPU_TermFrequency can be find out; GPU sorting is respectively performed for sorting the term id and

term frequency.

In the stage of filtering the stop words (GPU_FilterTermId), binary search and stream compaction are used. Given a raw term id and stop word id, we can use binary search to find out which term id corresponds to a stop word and mark it as 1. After generating a 1/0 flag, we can filter out all stop words marked as 1 by using stream compaction. As the CUDPP library provides stream compaction API which is called cudppCompact, it is easy to filter out the stop words. However, the CUDPP library does not provide any search API, we have to develop on our own. Luckily, binary search can be easily mapped onto the GPU.

As binary search works only on a sorted list of elements, the stop word id is sorted in CPU before passing onto the GPU. With a sorted list of id, all term ids are prepared to fit into the uniform-grid data structure so that it can perform several sequential binary searches at the same time. Each grid cell initiates a search for itself in the sorted list and the search assigns 1/0 in the output for every grid cell. The output is a list of 1/0 flag where the index of the list is the location of term id and 0 represents that the term is a stop word. The coding of GPU based binary search is shown in Table 4.4. Also, an example of the whole process is shown in Figure 4.5.

```
__global__ void GPU_BinarySearch(int *output, int *termId, int*
stopWord, int last, int gridx )
{
    int i = threadIdx.x+blockDim.x*blockIdx.x;
    int in = i+blockIdx.y*blockDim.x*gridx;
    int first = 0;
    output[in] =1;
    while(first<=last){
        int mid= (first+last)/2;
```

```
        if(termId[in] > stopWord[mid]){
            first = mid+1;
        }else if (termId[in] < stopWord[mid]){
            last = mid-1;
        }else{
            output[in] = 0;
            break;
}}}
```

Table 4.4 – Coding of GPU_BinarySearch

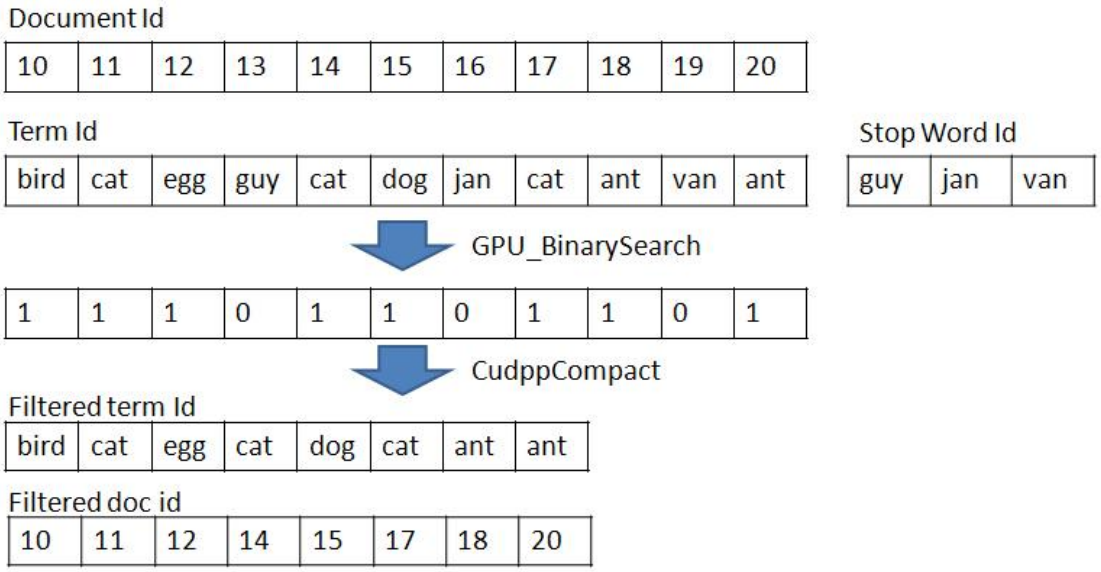


Figure 4.5 – Example of filtering term id and doc id

Now we have a list of filtered term id, the next stage is to sort the filtered term id list. In this stage, if all the term ids are 32-bit positive integers, the algorithm can directly use key-value pair sort which is provided by CUDPP library. However, if the term ids are more than 32 bits positive integer, the algorithm has to use the CUDPP Borrow-Bit radix sort which has been introduced in Chapter 3. For simplicity, we now assume that all the term ids are 32-bit positive integers. By performing a key-value pair sorting, the key is an index of filtered term id which starts from 1 to the total number of filtered term id and the value is the filtered term id. An example of this stage is shown in Figure 4.6.

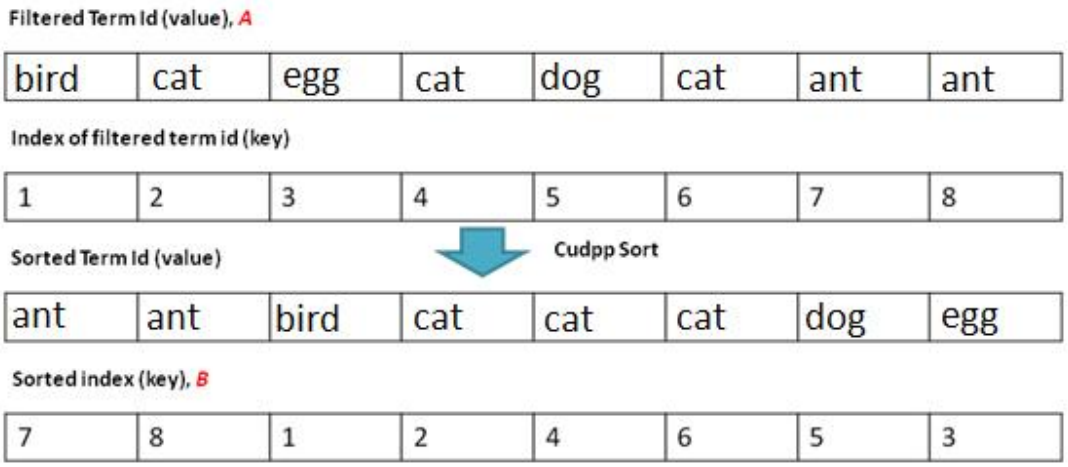


Figure 4.6 – Example of key-value pair sorting on filtered term id

After sorting the filtered term id, we can bring the sorted term id to the next stage in order to find the boundary index which aims to calculate the term frequency. The sorted index is used for retrieving the term id at the last stage.

There are two objectives in the boundary finding stage. The first one is to find the first position of each unique term id. It aims to prepare the list of 1/0 flag for counting frequency in the next stage. The second one is to find the last position of each unique term id. It aims to prepare the list of 1/0 flag for getting the term frequency in the next stage.

Since the program works on GPU, it needs a parallel data structure program instead of a sequential program. Same as the GPU binary search function, the sorted term ids are prepared to fit into a uniform-grid data structure so that it can perform several sequential programs in parallel. The function for finding the first position of each unique term id is called segmentFlag whose program code is shown in Table 4.5. If the input[i] is not equal to input[i-1], it means that the element of index i is the first term which has a different term id, so the output is marked as 1. However, if the

input[i] equals to input[i-1], it means that the index i is not the first element, so the output is marked as 0.

```
__global__ void segmentFlag(int *d_out, int *d_in, int clength, int
gridx)
{
    int i = threadIdx.x+blockDim.x*blockIdx.x;
    int in = i+blockIdx.y*blockDim.x*gridx;

    if(in==0){
        d_out[0]=1;
    }else if(d_in[in] != d_in[in-1]){
        d_out[in]=1;
    }else if(d_in[in] == d_in[in-1]){
        d_out[in]=0;
    }
}
```

Table 4.5 – Coding of segmentFlag

The function for finding the last position of each unique term id is called compactFlag, whose program code is shown in Table 4.6. If the input[i] equals to input[i+1], it means that the element of index i is not the last term which has the same term id of index i+1 , so the output is marked as 0. However, if the input[i] does not equal to input[i+1], it means that the index i is the last element, so the output is marked as 1.

After the GPU_FindBoundary stage, we will have segmentFlag and compactFlag. An example is shown in Figure 4.7.

```
__global__ void compactFlag(int *d_out, int *d_in, int clength, int
gridx)
{
    int i = threadIdx.x+blockDim.x*blockIdx.x;
    int in = i+blockIdx.y*blockDim.x*gridx;
    if(in==clength-1){
     d_out[clength-1] = 1;
     return;
    }
    if(d_in[in] == d_in[in+1]){
         d_out[in]=0;
    }else{
         d_out[in]=1;
    }
}
```

Table 4.6 – Coding of compactFlag

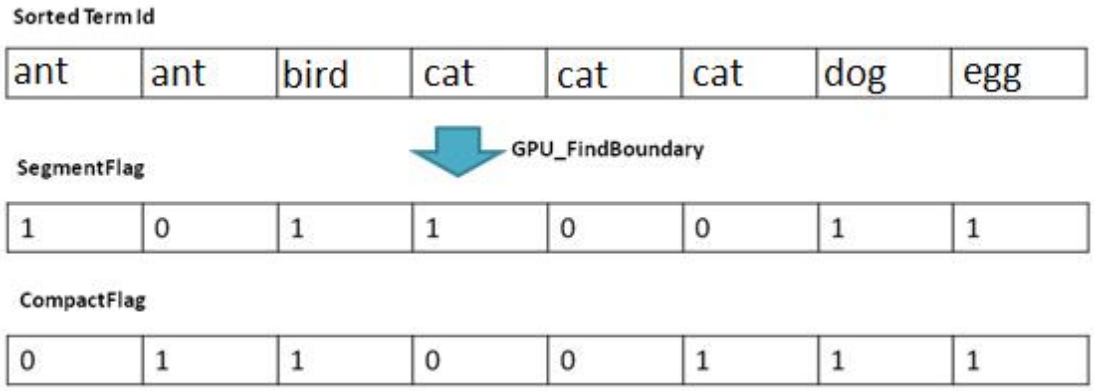


Figure 4.7 – Example of segmentFlag and compactFlag

After the segmentFlag and compactFlag are prepared, we can count the term frequency. There are three steps in the GPU_TermFrequency stage. Firstly, an input list whose size is the same as the sorted term id and the value of each element is 1 is prepared. Then, a segmented scan with segmentFlag is performed. The largest value or index of each segment is the frequency of a term. Secondly, a stream compaction with compactFlag to generate the term frequency list is performed. Finally, a scan

with the term frequency list is performed. It aims to generate an index of each unique term id in the sorted term id list. An example is shown in Figure 4.8.

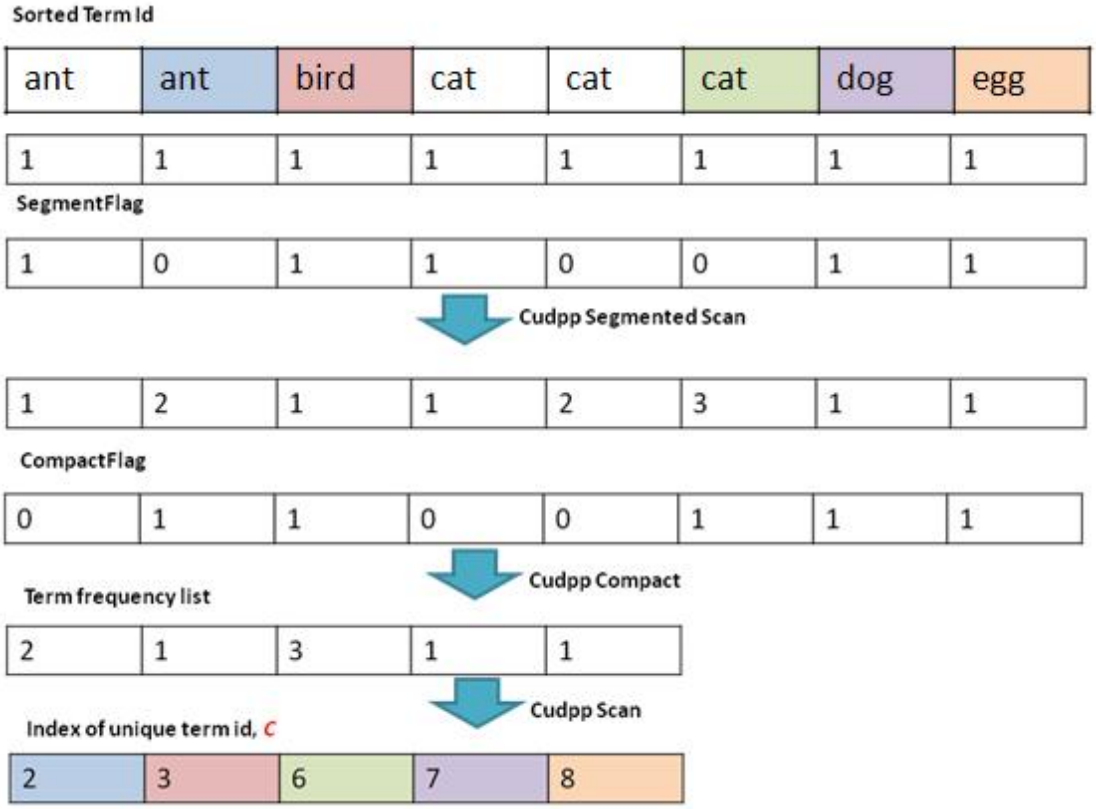


Figure 4.8 – Example of counting term frequency

After the GPU_FindBoundary stage, we have a term frequency list. In order to know which term is hot, sorting of the term frequency list is necessary. Again, cudppSort is used to perform key-value sorting of the term frequent list. An example is shown in Figure 4.9.

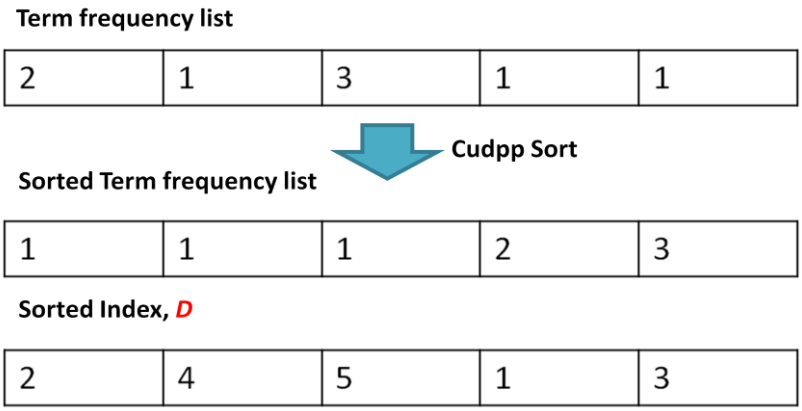


Figure 4.9 – Example of sorting term frequency list

The final stage is to get the hot terms and related documents. Using the list of A, B, C and D, the highlighted labels in the above figures are used to retrieve the hot term and the related documents. Figure 4.10 below shows the process of getting hot terms. To illustrate, the hottest term – "cat" which is shown in grey in the figure can be obtained by firstly using the last value ***D*** to find the index of ***C.*** Then the value of ***C*** is used to find the index of ***B*** and finally, the value of ***B*** is used to find the hot term in ***A***.

Figure 4.10 – Example of retrieving hot terms

This stage is performed on CPU, even it is a GPU based hot term extraction. To get the hot terms and related documents, a lot of intermediate results such as the

aforementioned list of A, B, C and D are needed. Due to the memory limitation of GPU devices, more memory will be used if the intermediate results are kept in the last stage. If more memory has been used on the intermediate results, it will decrease the number of terms handled by the hot term extraction algorithm. Therefore, we should free up the memory right after transferring any intermediate results from device memory to host memory.

4.6.2 Experimental Result

The experimental results of CPU and GPU based hot term extraction on different size of search results are reported in this section. The experimental platform is an Intel® Xeon® Processor X3450 with 2.66GHz machine with 16GB of memory and the graphic card is GeForce GTX 470 that has 14 multiprocessors and 448 CUDA cores with 1280 MB memory. Due to the limitation of memory in GeForce GTX470, it only supports up to around 32 million terms in the GPU based hot term extraction.

The experimental data was collected by the K-Matrix Digital Intelligence Ltd. The data collection period was between 25th March 2010 and 13th April 2010. There are around 19 billion terms with 10 million unique terms distributed in 50 million documents. The documents are collected from different forums in various regions (Hong Kong, Macau and China). Details of the forums are listed in Appendix B.

Cost Analysis

The prices of GPU and CPU that used in the experiment are similar. In CPU, the price of Intel® Xeon® Processor X3450 is around HKD $2,500. In GPU, the price of GeForce GTX 470 is around HKD $2,800. Since the prices are similar, using the GPU to perform hot term extraction is worth if the performance of GPU is better than CPU.

Speed Experiment

Figure 4.11 reports the processing time of the hot term extraction in CPU and GPU with different number of term ids. In CPU, we use the merge sort characteristic to

divide the term id into approximately half size and then perform the sort into each core processor. Lastly, the sort results are merged. There are 4 cores with multi-threading in Intel® Xeon® Processor X3450. We can fully utilize 8 cores in the program. Obviously, the performance of single core is the worst, and the performance of three to eight cores are very close. It proves that the merge sort applied in multi-core processors has a better performance than running a sorting in a single processor. Moreover, the performance trends are converged by adding CPUs. The memory architecture of CPU is different from GPU. GPU has more arithmetic logic units devoted to data processing while CPU has more control units dedicated to data caching and flow control. The architecture of CPU was designed to optimize the performance of sequential code. It makes use of control logic to allow instructions from a single thread of execution to execute in parallel or even out of their sequential order while maintaining the appearance of sequential execution [69]. To maintain the high speed of execution, the merge sort process requires large amount of cache memories for reducing instruction and data access latencies, so that the performance is limited. Nevertheless, the merge sort brings around 5x performance speed up when the number of term is 32 million. The best performance is still in GPU. Comparing with the CPU based hot term extraction, the GPU based hot term extraction has around 1.7x performance speed up when the number of term is 32 million. Although the processing time difference is just around a second, it is very important in the real time systems.

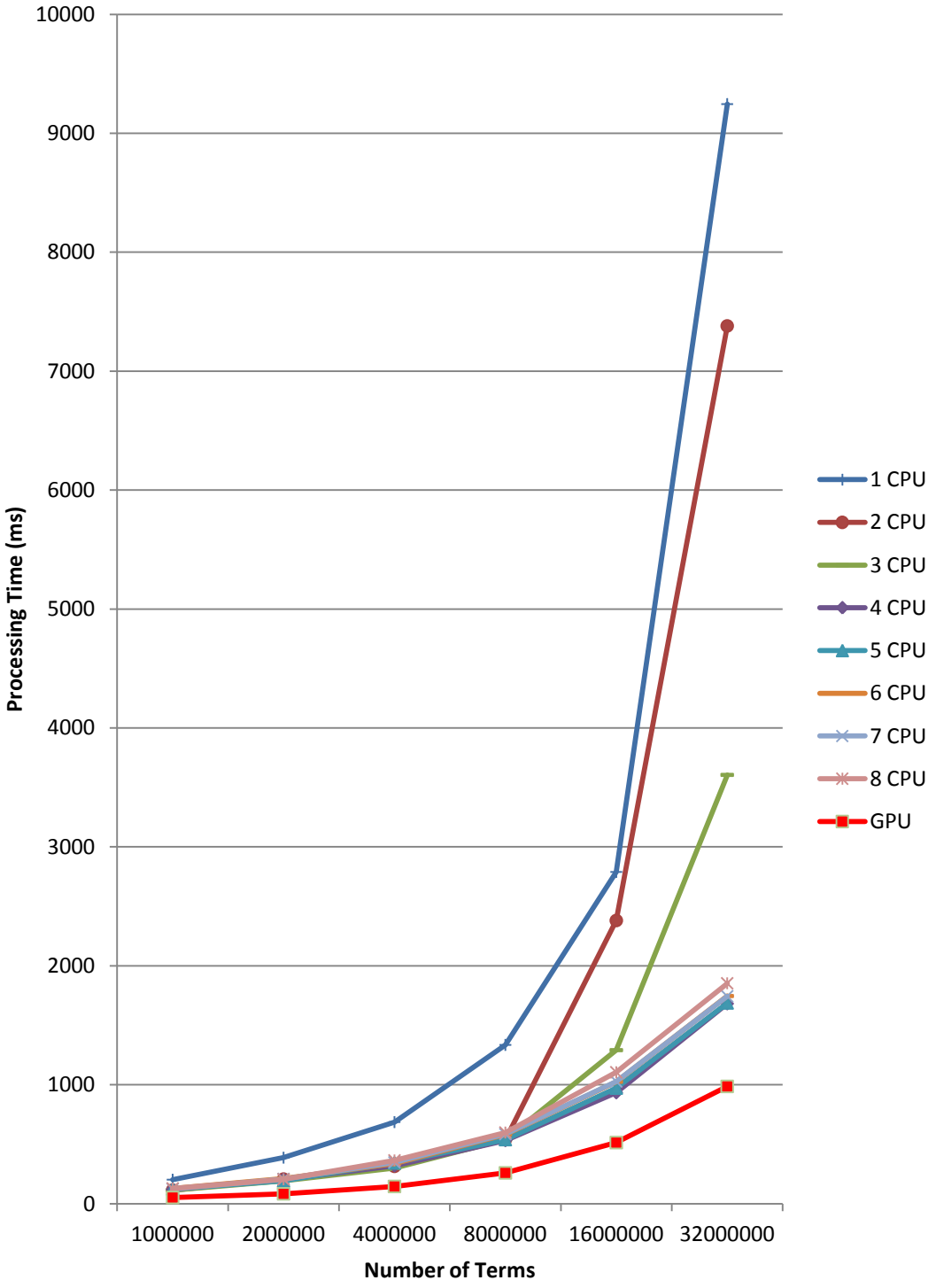


Figure 4.11 – Performance of Hot Term Extraction with different number of terms

Processing Time Occupation

Figure 4.12 and Figure 4.13 show the occupation of processing time in hot term extraction with CPU and GPU. They demonstrate two important facts: (1) sorting process occupies more than 80% processing time in CPU based hot term extraction; (2) the process time of transferring data from host to device and from device to host occupies about 12% in GPU based hot term extraction.

With a view to fact (1), hot term extraction spends most of the time in sorting process. In Chapter 3, we proved that the GPU sorting is much faster than CPU sorting. Therefore, the GPU based hot term extraction can get a better performance with using GPU sorting. According to fact (2), the overhead in transferring data from host memory to device memory or vice versa is the tradeoff of using GPU device to perform programming. Therefore, a well designed algorithm is important in a way that avoids the unnecessary overhead and waste of resources.

Another important point is that the process of getting hot term list is running on CPU, even in the GPU based hot term extraction. In order to get the hot term list in the last step, lots of intermediate results are required. Since the memory is limited in the GPU device, for providing more memory to support more number of terms in the algorithm, the intermediate results are immediately transferred from device memory to host memory.

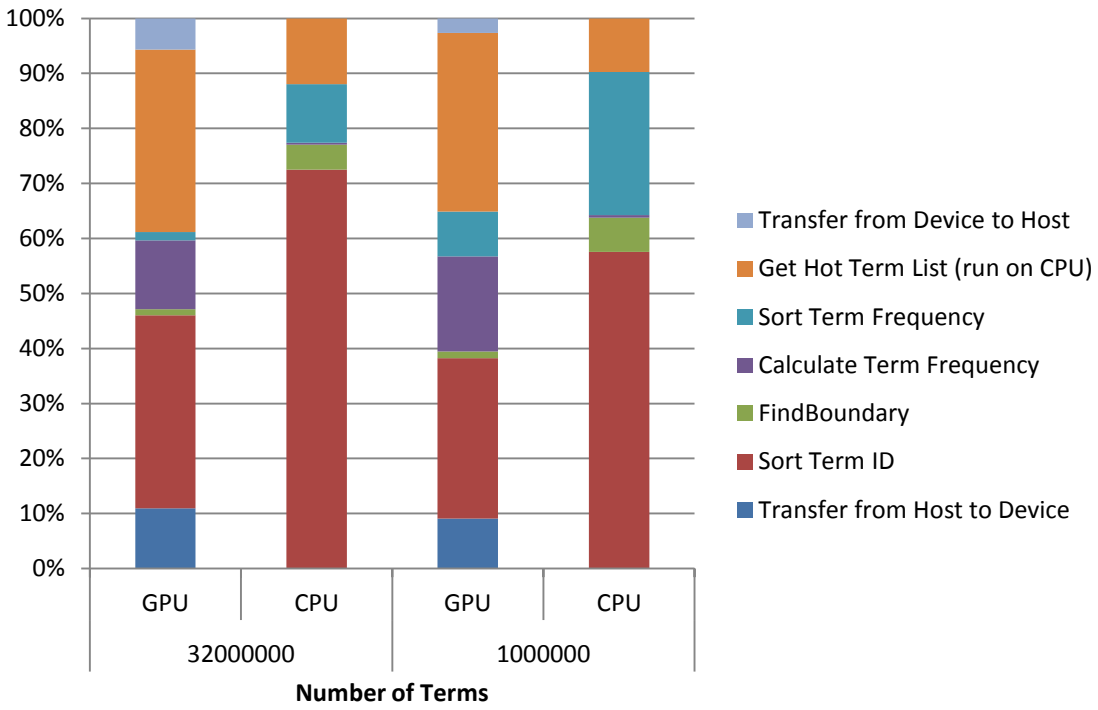


Figure 4.12 – Processing time breakdown with different number of terms

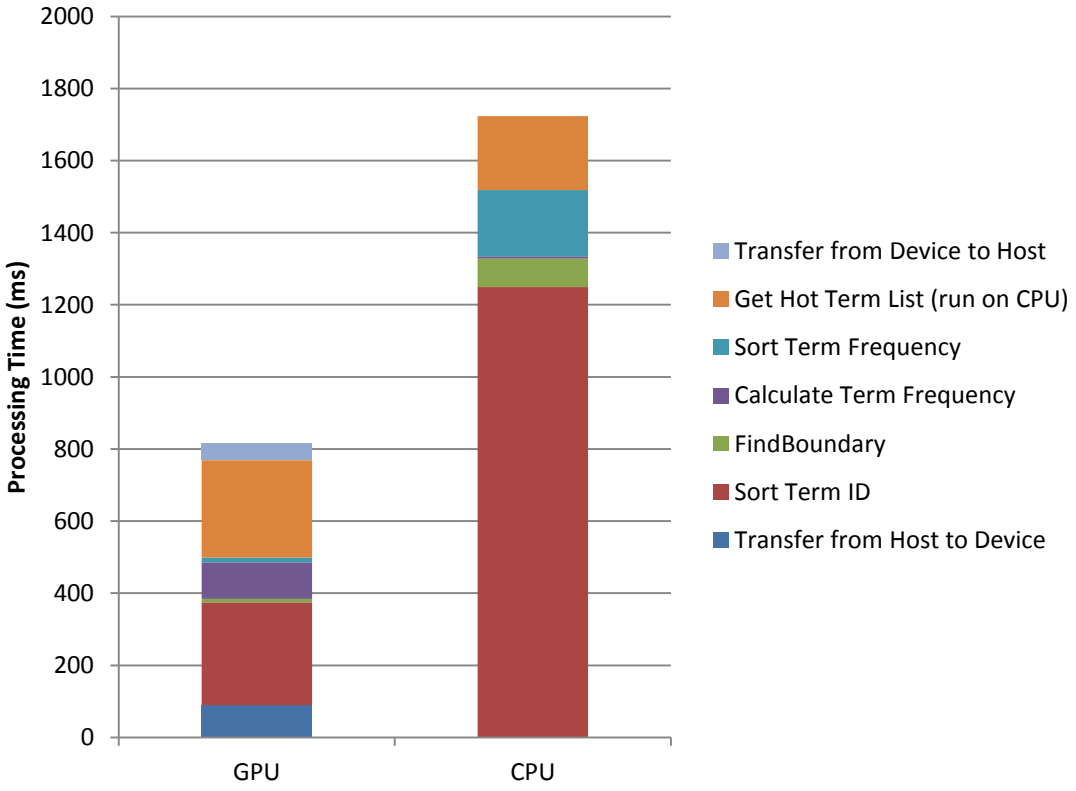


Figure 4.13 – Processing time breakdown in 32 million of terms

Hot Term Extraction

Figure 4.14 shows three examples of searching different keywords with return of top 30 hot terms. To quote the query of “smartone” as an example, there are 1,565 documents returned with 63K terms in those documents. Among these 63K terms, 28.49% of them are “smartone”. The second hot term is “plan” which has 8.43%. And the third hot term is “iphone” of 8.30%. The term “iphone” is highly related to “smartone” because Smartone was selling iphone during the searching period and many people were discussing the iphone plan and Smartone.

Regarding the query of “pccw” example, there are 4165 documents returned with 185K terms in those documents. Among these 185K terms, 33% of them are “pccw” and the second hot term is “上網” of 5.06%. Other hot terms are also relevant to the keyword searched. It proves that people were concerning those hot terms during the searching period.

As illustrated in the query of “曼聯”, the second hot term is “拜仁”. Since there was a football match “曼聯 vs 拜仁” on 31st March,2010, and some people discussed in the forums and frequently mentioned these two hot terms.

The hot term extraction is beneficial to different users. For the example of “smartone” query, if the user is a Smartone staff, he may want to know the comment of users. So some terms such as “問題”, “唔好” are useful to him. If the user is in a role of marketing, he may want to know something about competitors. Terms such as “pccw”, “csl”, “轉台” are useful to him. If the user is a customer, he may want to know the features about Smartone. Terms such as “plan”, “上網”, “3g”, ”電話”, “wifi”

are useful and “iphone” may also attract to him.

Keyword: smartone
Documents: 1565
Terms: 63947

smartone	28.49%
plan	8.43%
iphone	8.30%
上網	5.50%
3g	3.50%
mobile	3.44%
電話	3.31%
問題	3.04%
discuss	2.87%
好多	2.80%
vodafone	2.41%
3gs	2.37%
pccw	2.36%
手機	2.34%
無限	2.30%
phone	2.28%
用緊	2.25%
唔到	2.02%
sms	1.93%
寬頻	1.91%
個月	1.80%
一個	1.77%
出機	1.67%
想轉	1.47%
唔好	1.44%
wifi	1.36%
csl	1.31%
轉台	1.25%
香港	1.25%
流動	1.23%

Keyword: pccw
Documents: 4165
Terms: 185575

pccw	33.00%
上網	5.06%
plan	4.53%
電話	4.33%
mobile	4.04%
discuss	3.62%
問題	3.32%
wifi	3.18%
寬頻	2.99%
個月	2.90%
3g	2.67%
香港	2.65%
30m	2.51%
msn	2.44%
一個	2.14%
hkbn	2.14%
投訴	2.05%
好多	2.00%
用緊	1.98%
速度	1.94%
forum	1.89%
公司	1.77%
服務	1.77%
iphone	1.73%
hotmail	1.68%
手機	1.66%
router	1.58%
唔好	1.56%
續約	1.42%
hkepc	1.32%

Keyword:曼聯
Documents: 80228
Terms: 3124117

曼聯	23.32%
拜仁	6.38%
vs	5.14%
路士	4.41%
車路	4.41%
討論	3.94%
慕尼	3.86%
尼黑	3.86%
仁慕	3.82%
discuss	3.26%
45	3.26%
英超	3.23%
聯賽	3.05%
冠軍	2.34%
即時	2.28%
時討	2.26%
歐聯	2.24%
利物	1.90%
物浦	1.89%
車仔	1.88%
uwants	1.86%
歐洲	1.72%
61	1.66%
直播	1.64%
有線	1.64%
朗尼	1.61%
賽冠	1.58%
大戰	1.56%
19	1.49%
洲聯	1.46%

Figure 4.14 – Example of top 30 hot terms with different keyword searching

Chapter 5

Conclusion

5.1 Contribution

In this thesis, we have proposed a flexible key-value pair sorting algorithm and a GPU based hot term extraction algorithm. The proposed algorithms have been tested on different web forums as listed in Appendix B and demonstrated by various experimental results, which indicate that they are efficient.

To the best of our knowledge, this is the first attempt to perform hot term extraction in search results with GPU and a preliminary version of this work will be reported in [70]. The contributions of this thesis include:

1) A flexible key-value pair sorting algorithm called Borrow-Bit sorting: The Borrow-Bit sorting can flexibly set the number bits in key and value, as long as the sum of key and value is within 64-bit. In the experiment results, we show that the Borrow-Bit approach provides flexible and efficient sorting. In addition to addressing the limitation on CPU, it also works on GPU. It provides a faster key-value pairs sorting when using JAVA. Moreover, it provides a method to use CUDPP Radix Sort for sorting key-value pairs where both key and value are more than 32 bits.

2) A fast hot term extraction method using GPU: In the convenient digital world, users always use the search engine to find what they want. Among the huge amount of search results, the hot term extraction results can provide hidden information to users. We can enjoy the advantage of parallel characteristics by implementing the

hot extraction with GPU, it is much more efficient to get the most frequent terms in the search results. Comparing the processing time with the CPU based hot term extraction, the GPU based method can archive around 1.7x speed up.

5.2 Future Work

Due to the limited time of conducting this research, there are some potential enhancement areas in hot term extraction.

1) N-Gram preprocessing

In the proposed hot term extraction algorithm, bi-gram is used to define a term. Since each document consists of many words, using n-gram to get all the combination of various terms is inefficient. Therefore, we use the bi-gram as the basic unit of term. To make the result more accurate, we would like to apply n-gram in the top N hot term lists. Firstly, the location of each word in document is stored. After generating the top N hot term lists, we can combine the terms which are located in consecutive sequences.

On the other hand, in order to reduce the memory usage in the hot term extraction, we have to minimize the redundant terms in the pre-processing step. Named entity recognition can be chosen to help identify terms like people, locations and organizations. For example, if the dictionary consists of the term “電訊盈科”, the named entity recognition can help reduce three bi-gram terms including “電訊”, “訊盈” and “盈科”. The named entity recognition helps to reduce the number of terms and improve the accuracy in the hot term extraction.

2) Hot Term Clustering

In hot term extraction, we can treat the resulted listing as a cluster of searching query. We can form a link to another cluster which is searched by another query if two clusters have the same hot terms. Figure 5.1 shows the example of “smartone” cluster and “pccw” cluster. In the cluster, the hot terms are shown in different colors and different font sizes in Figure 5.1. The font size is proportional to the term frequency, so that the hotter terms may have a larger font size. There are some hot terms which are the same in these two clusters, so we can form a linkage between these two clusters. Each link between the two clusters can perform another search with AND operation. Users can click the link to perform another search to find the results they are interested in and the corresponding hot term lists. For example, if a user clicks the link of “iphone”, another search query is formed to have “smartone AND pccw AND iphone”. The search will automatically find the documents which consist of “smartone”, “pccw” and “iphone”, and then list out the corresponding hot term lists.

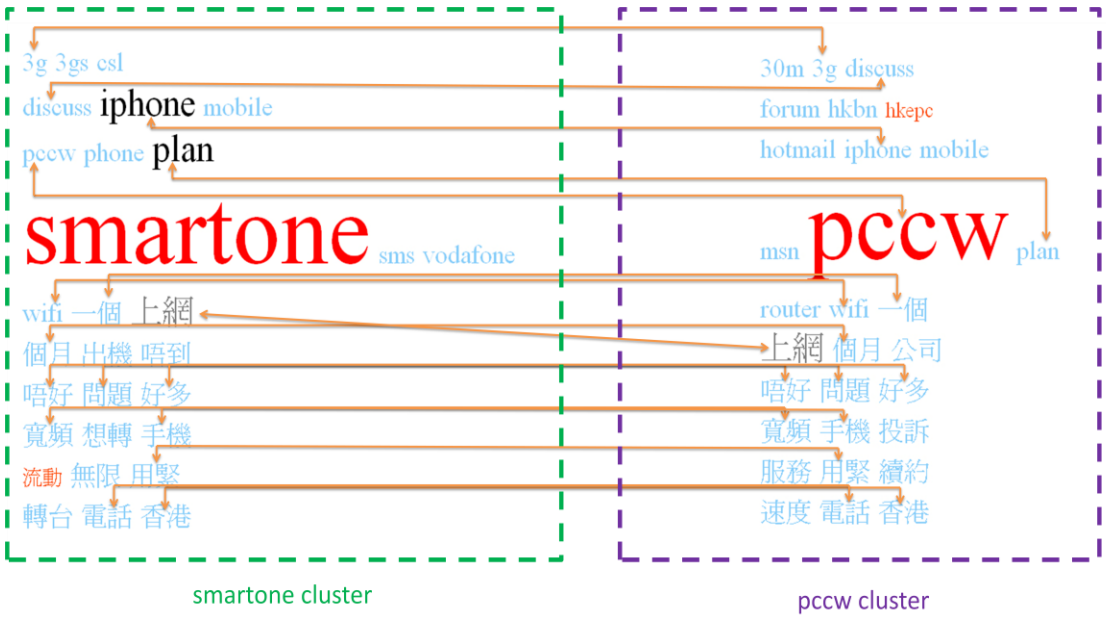


Figure 5.1 – Example of hot term clustering

In practice, the hot term clustering is very useful. We can find the WOM of different features or products in different brands easily. Figure 5.2 shows an example in the cosmetic industry. There are four clusters searched by different keywords, namely “biotherm”, “shiseido”, “shu uemura” and “fancl”. The hot term “美白” appears in these four clusters. Users can easily find the WOM of “美白” between these four brands and the hot term clustering can let users know that this hot term is important in the cosmetic industry. In addition, the appearance of the hot term - “睫毛” in the hot term lists of the “biotherm” cluster and “shiseido” cluster reveals that this term has more WOM in these two brands than “shu uemura” and “fancl”. In other words, this term attracts interests from more people and may also be meaningful to the user.

Since the hot term clustering widely reuses the hot term extraction algorithm, a fast algorithm is important. The GPU can help to achieve faster performance than CPU.

3) Multiple GPUs

In this thesis, the proposed GPU based hot term extraction only uses a single GPU. To enhance the processing time of hot term extraction, we can implement the algorithm with multiple GPUs. In CPU based hot term extraction, we use multiple cores to perform merge sort for sorting term id. It is proved that the processing time of using merge sort in multiple cores is much faster than sorting in only a single core. By applying this idea to GPU, we can carry out the merge sort in multiple GPUs and achieve a better performance.

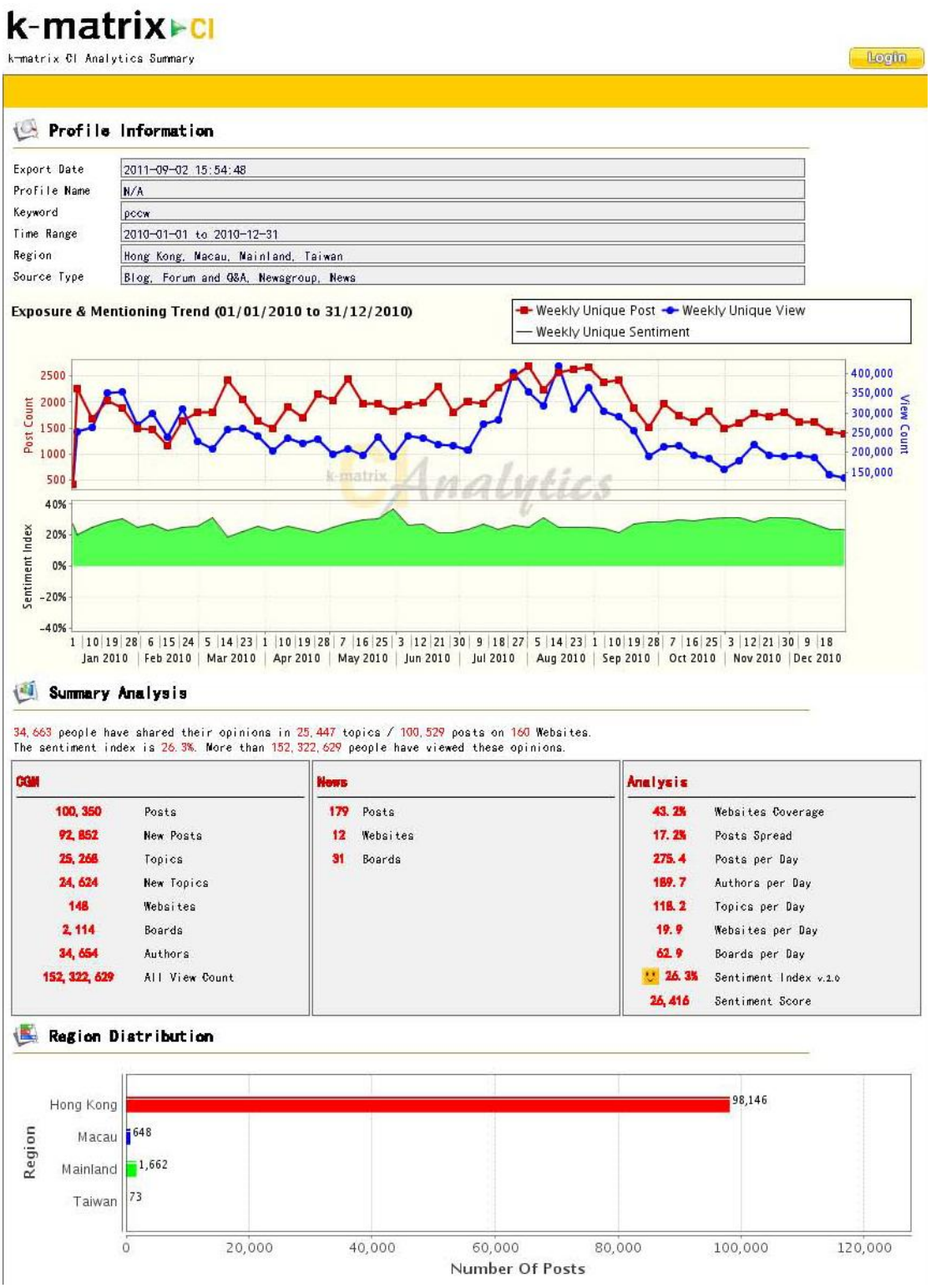
On the other hand, the overhead in transferring data from host memory to device

memory or vice versa in GPU is a bottleneck. Using multiple GPUs can reduce of this overhead. From a single GPU, it is sequentially transfer data from host memory to device memory or vice versa. However, using multiple GPUs can asynchronous transfer the data which means that multiple transfer the data at the same time. As a result, we can coordinate works across multiple GPUs to archive faster processing time.

Figure 5.2 – Cosmetic Industry Example in hot term clustering

Appendices

A. K-Matrix CI Report – PCCW



k-matrix CI

k-matrix CI Analytics Summary

Source Type Distribution

	Forum and Q&A	News	Blog	Newsgroup
Distribution (%)	97.4	0.2	1	1.4

Hot Topic List (Top 10)

Title	Websites		Sentiment Index (%)	Contribution (%)	Correlation Index (%)
PCCW 等待出iphone4報到區^^	Uwants 討論區		19.6 %	1.5 %	94.5 %
新出令con. GEO$100/2對. create. dueba. seeshell. peach. neo. candy. barbie. mimo. alisa	3boys2girls		93.4 %	1.2 %	53.3 %
---> 終極平plan (PCCW固網電話, 家居寬頻, Wifi無線寬頻) <---	香港討論區		39.9 %	0.9 %	91.2 %
小J數码, 15号更新: 黑莓9000, 香港行货各版本: 港3, 数码通, PCCW, CSL, 成色95新	家电论坛		1.4 %	0.9 %	100.0 %
[香港] ★ ★ ★ 亂二 ★ ★ ★(終於都出左商城@全部 l 積分0.0)【開左N群~招朋友仔一齊玩, 唔駛有咩大台】#214 解決PCCW 上唔到官網方法	2000FUN論壇		29.9 %	0.8 %	94.3 %
團訂 :) 人仔SMS線卡 言使下下用48蚊買張卡甘貴都可以同人仔既FD SMS!	3boys2girls		0.8 %	0.6 %	19.0 %
全三男既CON款係晒入面丫(!)_唔買對唔住自己喇^100%REAL+NEW..日日NK交收(!!)^_^	3boys2girls		0.0 %	0.5 %	92.5 %
用pccw既plan...有機未呀	香港討論區		17.3 %	0.5 %	96.4 %
【暑期工攰】pccw熱線攻略/分享	香港高登討論區		37.3 %	0.5 %	100.0 %
[台灣] ★ ★ ★ ★ ★ 視葉亂終於改版喇! ! 內測已開放★ ★ ★ ★ ★	2000FUN論壇		30.0 %	0.4 %	87.1 %

Top Site List (Top 10)

Websites		Sentiment Index (%)	Contribution (%)	Post Count	Topic Count	Unique Author
香港討論區		26.8	24.6	24737	7018	9868
Uwants 討論區		28.5	17.9	18002	5406	6808
香港高登討論區		15.2	14.1	14208	2412	5044
電腦領域 HKEPC Hardware		23.9	13.3	13406	2330	3024
3boys2girls		39.4	8.3	8364	1699	1833
鈴聲之王討論區		22.2	3.4	3441	608	901
AV100Fun討論區		32.6	2.3	2353	325	239
2000FUN論壇		21.9	2.3	2278	303	601
親子王國		33.6	2.2	2175	536	1083
yahoo知識		37.9	1.5	1463	981	998

Top Author List (Top 10)

Authors		Sentiment Index (%)	Contribution (%)	Post Count	Topic Count	Most Active Board
Qo丫霖oQ		21.4	1.0	1039	25	3boys2girls - 化妝用品/顏色con
Jacky_Hsu		1.4	0.8	850	4	家电论坛 - 数码产品交易区
i_am_up_post		90.0	0.6	640	6	3boys2girls - 化妝用品/顏色con
`小泳`		95.9	0.6	638	10	3boys2girls - 化妝用品/顏色con
Bab!諾		0.0	0.6	638	1	3boys2girls - 預購團訂
孤.星.淚		1.1	0.4	445	392	Uwants 討論區 - 寬頻上網
鴨仔		13.1	0.4	443	235	電腦領域 HKEPC Hardware - 網絡寬頻
Maykhera		4.2	0.4	400	8	澳門流動社區 - 【票券買賣】
ha_97		8.6	0.3	336	117	香港討論區 - 手機上網討論
某人兄		11.1	0.3	332	136	電腦領域 HKEPC Hardware - 網絡寬頻

B. Data Collection – CGM

Online news (Hong Kong)	Blog (Hong Kong):
太陽報	news.newsgroup.com.hk
大公報	news.nntp.hk
成報	my.newsgroup.la
星島日報	news.3home.net
文滙報	news.balabu.net
都市日報	news.hkux.net
蘋果日報	news.wonderspace.net
sina.com.hk 體育	news.wonderfuland.net
雅虎香港新聞	news.hkpcug.org
蘋果動新聞	news.idsam.com
經濟通新聞	news.ourrice.com
The Standard - Breaking News	news.sporthk.net
The Standard - Sections	
明報即時新聞	
HKEJ instant news	
頭條網 - 即時新聞	
Reuters	

Forum (China)	
绿茶股票论坛	OFFICE 精英俱乐部
天易投资论坛	中国时尚论坛
银江论坛	走进中关村
通吃岛证券论坛	易购网上购物论坛
海涛股票论坛	都市论坛
理想论坛	凤网社区
鼎砥投资论坛	上海热线互动社区
福建飞狐论坛	中国孕育网
現金流投资论坛	中国早教网
包卜篮球论坛	中国灯具论坛
詹姆斯中文论坛	北方论坛
克里斯·保罗中文网	照明技术论坛
街盟论坛	摇篮网
潮流长安论坛	建筑论坛
科比中文网	中华室内设计网
环球鞋网论坛	阿里巴巴论坛
虎扑论坛版	汉网论坛
街頭籃球	大众论坛
一起 NBA	泡泡网论坛
爱死科比	西岸论坛
我乐 NBA	海运俱乐部
我爱科比	赛迪网论坛
鞋帮	车天下社区
CN-KIX	聚友社区
触动球鞋论坛	宁波百姓论坛
1626 活动区	中国网和平论坛
TBBA 篮球论坛	中国女性时尚论坛
新新球鞋论坛	外企白领论坛
天涯論壇	中华网论坛
百度知道	合众外贸论坛
都市客论坛	福步外贸论坛
IXPUB 技术社区	中华网_科技论坛
ZDNetChina 中文社区	太平洋亲子网
ChinaByte 论坛	FGLADY 风尚伊人网论坛
51CTO 技术论坛	特价王网上购物论坛
IT 世界网论坛	丫丫论坛丫丫社区
计世网论坛	大洋论坛

ChinaUnix.net	中国宠物论坛-PET86
中国 IT 实验室-IT 社区	好孩子育儿网
YOKA 时尚论坛	育儿网论坛
时尚论坛	我爱北京交友聚会网(爱北论坛)
搜狐社区	拍拍网社区
ITPUB 论坛	西陆论坛
IT168 数码影人	年轻 e 族
IT168 办公维修论坛	篱笆论坛
IT168 家电论坛	麻辣社区
IT168DIY 烧友会	西祠胡同社区
网易论坛	东南网-榕树大院
丁丁网生活论坛	POCO 论坛
TOM 社区	中关村在线论坛_游戏社区
新浪论坛	中国 LED 网论坛
广州妈妈	粉丝网主论坛
西安妈妈	中关村软件论坛
大众点评网	成龙汽车网
广州论坛	汽车画报
淘宝帮派	51.Com
太平洋电脑网	和讯论坛
新浪亲子女人论坛	CHE168
淘宝打听	愛卡汽車網
太平洋女性网论坛	中国站长论坛
VOGUE 时尚网	易车网社区
校内网论坛	电脑之家(宽带山社区)
ChinaRen 社区	CSDN 社区
21CN 社区	汽车之家
北青网论坛	ELLE
腾讯论坛	妆点网
京华论坛(千龙论坛)	中国证券网
小熊在线	海报社区
网上车市	OnlyLady 论坛
汽车之友	花花女人社区
搜城论坛	爱丽女性社区
瑞丽论坛	时空网
慧聪网	19 楼
无垠社区	证券大智慧
电脑商网经销商论坛	新华网

Forum (Hong Kong)	Forum (Macau)
鈴聲之王討論區	澳門流動社區
香港討論區	澳門互動社區
Uwants 討論區	澳門凸區論壇
大眾論壇	澳門討論區
香港互聯網站	
失敗論壇	
web4share	
香港製造論壇	
GoalGoalGoal 球迷大聯盟	
數碼天地論壇	
電腦領域 HKEPC Hardware	
PDA User Message Board	
親子王國	
香港高登討論區	
我的討論區	
跑步舍討論區	
Hk-kicks	
香港運動網	
she.com	
2000FUN 論壇	
esdlife	
3boys2girls	
MY903	
Geoexpat	
信報論壇	
Asiaxpat	
Shemom	
明報討論區	
HD-DC	
攝出中國	
奧林派	
HKCUG	
dcfever	
yahoo 知識	
BeautyExchange	
OpenRice	
香港人网	

Blog (China)	Blog (Hong Kong)
网易博客	Blogger
新浪博客	Xanga
腾讯博客	Sina Blog HK
百度空间	Yahoo Blog HK
天涯博客	
聚友博客	
瑞丽博客	
和讯博客	
poco.cn	
博尚网	
同学网微博客	
中国教育人博客	

Bibliography

1. R. Feldman, M. Fresko, Y. Kinar, Y. Lindell, O. Liphstat, M. Rajman, Y. Schler, and O. Zamir, “Text mining at the term level,” In Principles of Data Mining and Knowledge Discovery, Lecture Notes in Computer Science, vol. 1510, pp. 65–73, 1998.
2. B. Larsen and C. Aone, “Fast and Effective Text Mining Using Linear-time Document Clustering,“ Proc. KDD 99, pp. 16-22, 1999.
3. Carrot Search. http://search.carrotsearch.com/carrot2-webapp/search
4. H. Ahonen, “Knowledge discovery in documents by extracting frequent word sequences,” Library Trends, vol.48, issue 1, pp. 160–181, 1999.
5. K-Matrix Digital Intelligence Ltd. http://www.kmatrixonline.com
6. K-Matrix CI. http://ci.kmatrixonline.com
7. J. McEntyre and D. Lipman, “PubMed: bridging the information gap,” Cmaj, vol. 164, no. 9, pp. 1317-1319, 2001.
8. B. Y.-L. Kuo, T. Hentrich, B. M. Good, and M. D. Wilkinson, “Tag clouds for summarizing web search results,” In WWW '07: Proc. of the Int. World Wide Web Conf., pp. 1203–1204, 2007.
9. PubCloud. http://dev.biordf.net/PubCloud/gc.jsp?query=genes&type=abstract&startYear=2009&endYear=2010&option=most_recent&recent=200&percent=10
10. NVIDIA CUDA. http://developer.nvidia.com/category/zone/cuda-zone
11. Kirk, D. B. & Hwu, W.-m. W., “Programming Massively Parallel Processors: A Hands-on Approach,” Morgan Kaufmann, Boston, Massachusetts, USA, 2010.
12. NVIDIA, "NVIDIA CUDA C Programming Guide v.4.0," 2011.
13. J. Nickolls, I. Buck, M. Garland and K. Skadron, “Scalable parallel programming with CUDA,” ACM Queue, vol. 6, no.2, pp. 40-53, 2008.
14. J. Krüger and R. Westermann, “Linear algebra operators for GPU implementation of numerical algorithms,” ACM Transactions on Graphics, vol. 22, no. 3, pp. 908 - 916, 2003.
15. N. Galoppo, N. K. Govindaraju, M. Henson, and D. Manocha, “LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware,“ in Proc. ACM/IEEE Conf. Supercomput., pp. 3, 2005,.
16. G. Quintana-Ortí, F. D. Igual, E. S. Quintana-Ortí, and R. van de Geijn, “Solving dense linear algebra problems on platforms with multiple hardware accelerators,” in Proc. of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2009.

17. P. Harish and P. J. Narayanan, “Accelerating Large Graph Algorithms on the GPU Using CUDA,” In HiPC, pp. 197 - 208, 2007.
18. A. Buluc, J. R. Gilbert and C. Budak, ”Solving Path Problems on the GPU,” Parallel Computing, vol. 36, 2010.
19. K.A. Hawick, A. Leist and D.P. Playne, “Parallel graph component labelling with GPUs and CUDA,” Parallel Computing, vol.36, issue 12, pp. 655 – 678, 2010.
20. O. Kalentev, A. Rai, S. Kemnitz and R. Schneider, “Connected component labeling on a 2D grid using CUDA,” Journal of Parallel and Distributed Computing, vol. 71, issue 4, pp. 615 – 620, 2011.
21. V. Garcia, E. Debreuve, and M. Barlaud, “Fast k Nearest Neighbor Search Using GPU,” Proc. CVPR Workshop Computer Vision on GPU, 2008.
22. B. Catanzaro, N. Sundaram, and K. Keutzer, “Fast support vector machine training and classification on graphics processors,” Proceedings of the 25th International Conference on Machine Learning, pp. 104 - 111, 2008.
23. I. Chiosa and A. Kolb, “GPU-based Multilevel Clustering,” IEEE Transactions on Visualization and Computer Graphics (TVCG), vol. 17, no. 2, pp. 132 – 145, 2011.
24. M. Schatz, C. Trapnell, A. Delcher and A. Varshney, “High-throughput sequence alignment using Graphics Processing Units,” BMC Bioinformatics, vol. 8, no. 1, p.474, 2010.
25. C. Trapnell and M. Schatz, “Optimizing data intensive GPGPU computations for DNA sequence alignment,” Parallel Computing, 2009.
26. J. Krüger, P. Kipfer, P. Kondratieva, and R. Westermann, “A particle system for interactive visualization of 3D flows,” IEEE Transactions on Visualization and Computer Graphics (TVCG), vol. 11, pp. 744 - 756, 2005.
27. J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, “Sparse matrix solvers on the GPU: Conjugate gradients and multigrid” ACM Transactions of Graphics., vol. 22, no. 3, pp. 917 - 924, Jul. 2003.
28. A. E. Lefohn, “A streaming narrow-band algorithm: Interactive computation and visualization of level-set surfaces,” Master’s thesis, University of Utah, Dec. 2003.
29. J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten, “Accelerating molecular modeling applications with graphics processors, Journal of Computational Chemistry,” J. Comp. Chem., vol. 28, pp. 2618 - 2640, 2007.
30. J. A. van Meel, A. Arnold, D. Frenkel, S. F. P. Zwart, and R. G. Belleman, “Harvesting graphics power for MD simulations, Molecular Simulation,” vol. 34, pp. 259 - 266, 2008.
31. T. D. R. Hartley, U. Catalyurek, A. Ruiz, F. Igual, R. Mayo, and M. Ujaldon,

"Biomedical image analysis on a cooperative cluster of gpus and multicores," in ICS '08: Proceedings of the 22nd Annual International Conference on Supercomputing, New York, NY, USA, ACM, pp. 15 - 25, 2008.

32. S. A. Manavski, "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography," In ICSPC 2007: Proc. of IEEE Int'l Conf. on Signal Processing and Communication, pages 65 - 68, 2007.
33. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. Purcell, "A survey of general-purpose computation on graphics hardware," Comput. Graph. Forum, vol. 26, no. 1, pp. 80 - 113, 2007.
34. S. Sengupta, M. Harris, Y. Zhang and J.D. Owens, "Scan primitives for GPU computing," in: Proc. of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, 2007.
35. CUDPP. http://code.google.com/p/cudpp/
36. D. Horn, "Stream reduction operations for GPGPU applications," In GPU Gems 2, Pharr M., (Ed.). Addison Wesley, chapter 36, pp. 573 - 589, Mar. 2005.
37. J. Hensley, T. Scheuermann, G. Coombe, M. Singh and A. Lastra, "Fast summed-area table generation and its applications," Computer Graphics Forum, vol. 24, issue 3, pp. 547 – 555, September 2005.
38. M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with CUDA," In H. Nguyen, editor, GPU Gems 3, Addison Wesley, chapter 39, pp. 851 – 876, Aug. 2007.
39. A. Greß, M. Guthe and R. Klein, "GPU-based collision detection for deformable parameterized surfaces," Computer Graphics Forum, vol. 25, issue 3, pp. 497 – 506, September 2006.
40. G. E. Blelloch, "Prefix sums and their applications," Technical Report, 1990.
41. N. Satish, M. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore GPUs," In IPDPS, pp. 1–10, 2009.
42. JCUDA. http://www.jcuda.org/
43. Thrust. http://code.google.com/p/thrust/
44. D. Merrill and A. Grimshaw, "High Performance and Scalable Radix Sorting: A case study of implementing dynamic parallelism for GPU computing," Parallel Processing Letters, vol. 21, no. 2, pp. 245-272, 2011.
45. Java Arrays.sort. http://docs.oracle.com/javase/6/docs/api/java/util/Arrays.html#sort(java.lang.Object[])
46. A. Marriott, "Maximizing Performance with Bespoke Programming," Logos Software.
47. Word of Mouth. http://en.wikipedia.org/wiki/Word_of_mouth

48. Google/KellerFay, Word of Mouth and the Internet Study, June 2011. http://www.gstatic.com/ads/research/en/2011_Word_of_Mouth_Study.pdf
49. CIC. http://www.cicdata.com/
50. Visible Technologies. http://www.visibletechnologies.com/
51. Radian6. http://www.radian6.com/
52. Alterian. http://www.alterian.com/socialmedia/products/
53. Collect Intellect. http://www.collectiveintellect.com/
54. Lithium Social Media Monitoring. http://www.scoutlabs.com
55. H. Ahonen-Myka, "Finding all maximal frequent sequences in text," in: Proceedings of ICML-99 Workshop on Marchine Learning in Text Data Analysis, pp. 11–17, 1999.
56. H. Ahonen-Myka, "Discovery of frequent word sequences in text," in: Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining, 2002, pp. 16–19.
57. H. Ahonen-Myka, "Mining all Maximal Frequent Word Sequences in a Set of Sentences," Proceedings of the 14th ACM International Conference on Information and Knowledge Management, pp. 255-256, 2005.
58. H. Ahonen-Myka and A.Doucet, "Data mining meets collocations discovery," In Inquiries into Words, Constraints and Contexts, Festschrift for Kimmo Koskenniemi, pp. 194–203. CSLI Publications, University of Stanford, 2005.
59. F. Beil, M. Ester and X. Xu, "Frequent term-based text clustering," in: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 436–442, 2002.
60. P. Ponmuthuramalingam and T. Devi, "Effective Term Based Text Clustering Algorithms," (IJCSE) International Journal on Computer Science and Engineering vol. 02, no. 05, pp. 1665-1673, 2010.
61. Y. Li, S.M. Chung and J.D. Holt, "Text document clustering based on frequent word meaning sequences," Data and Knowledge Engineering, vol. 64, issue 1, pp. 381– 404, 2008.
62. J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Operating Systems Design and Implementation, pp. 137–149, 2004.
63. Word Count Example. http://hadoop.apache.org/common/docs/current/mapred_tutorial.html
64 W. Cui, Y. Wu, S. Liu, F. Wei, M. Zhou, and H. Qu. "Context-preserving, dynamic word cloud visualization," Computer Graphics and Applications, IEEE, vol. 30, issue 6, pp. 42–53, 2010.
65. Unicode Chinese Characters. http://www.khngai.com/chinese/charmap/tbluni.php?page=0

66. Longest word in English. http://en.wikipedia.org/wiki/Longest_word_in_English
67. Bulletin Board Code. http://en.wikipedia.org/wiki/Bulletin_Board_Code
68. Merge Sort. http://en.wikipedia.org/wiki/Merge_sort
69. Kirk, D. B. & Hwu, W.-m. W., “Programming Massively Parallel Processors: A Hands-on Approach,” Morgan Kaufmann, Boston, Massachusetts, USA, 2010.
70. M.F. Cheng, F.L Chung, S.N. Chuang, “GPU Accelerated Hot Term Extraction from User Generated Content”, 2012 26th International Conference on Advanced Information Networking and Applications Workshops, WAINA, pp.851-856, 2012.