Fast Lists Intersection with Bloom Filter using Graphics Processing Units

Fan Zhang, Di Wu, Naiyong Ao, Gang Wang, Xiaoguang Liu, Jing Liu
Nankai-Baidu Joint Lab, College of Information Technical Science, Nankai University
Email: {zhangfan555, wakensky}@gmail.com, {aonaiyong, wgzwp}@163.com,
liuxg74@yahoo.com.cn, jingliu@mail.nankai.edu.cn

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Program ming-Parallel programming; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Performance, Experimentation

Keywords

Lists intersection, Bloom Filter, GPU

1. INTRODUCTION

Intersection of sorted inverted lists is an important operation in the web search engines. Various algorithms to improve the performance of this operation have been introduced in the literature [1, 3, 5]. Previous research works mainly focused on single-core or multi-core CPU platform and did not consider the query traffic problem arises in the actual systems. Modern graphics processing units (GPUs) give a new way to solve the problem. Wu et al. [6] presented a CPU-GPU cooperative model which can dynamically switch between the asynchronous mode and the synchronous mode. Under light query traffic, asynchronous mode is triggered, each newly arriving query is serviced by an independent thread. Under heavy query traffic, synchronous mode is triggered, all active threads are blocked and a single thread takes control of query processing. Queries are grouped into batches at CPU end, and each batch is processed by GPU threads in parallel. We summarize that putting the operations on GPU has two advantages: The massive on-chip parallelism of GPU may greatly reduce the processing time of lists intersection; A great part of work on CPU is offloaded to GPU. Overall the GPU will significantly increase throughput and reduce average response time in the synchronous mode. In this paper we consider techniques for improving the performance of the GPU batched algorithm proposed in [6] assuming sufficient queries at the CPU end.

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

To intersect two sorted lists M and N with $|M| \leq |N|$, for each document ID (docID) of M we assign it to a unique GPU thread. These threads use binary search to test whether the docID appears in N in parallel, so every thread will make $O(\log n)$ memory accesses. While in this paper, we propose a novel intersection algorithm based on Bloom filter which can greatly reduce the processing time. The main idea is reducing the number of memory accesses from $O(\log n)$ to $O(1)$. The tradeoff is that Bloom filter will introduce some false positives, besides we have to generate an extra hashed index. The size of the extra space can be customized according to the acceptable false positive rate.

2. METHODOLOGY

We generate the Bloom filter [2] for each inverted list offline. The hash functions are the same for all lists, while the length of Bloom filter is variable according to the length of the list. Taking $m = 16n$ as example (n is length of an inverted list, m is the number of bits we store the Bloom filter information for a list, each docID is 32 bits long), the size of Bloom filter data is half the size of the original inverted index, so the space of Bloom filter algorithm is $1.5x$ of the original index. However, compression algorithm can further be applied to the original inverted index to reduce total space. We store the original index and Bloom filter totally on the GPU global memory.

We implement the Bloom filter batched algorithm by replacing the binary search part of the PARA [6] algorithm by membership testers of a given Bloom filter. Each GPU thread is assigned one docID of the shortest list, it makes $O(1)$ lookups in the other lists' Bloom filter to test whether its corresponding docID exists or not. Once a docID exists in all the other lists' Bloom filter, the docID returns as a common docID. Obviously, some false positives will be generated. However, as long as they only account for a very small percentage, it is acceptable in practice. Moreover, the following ranking step will calculate a relevance score for each document and sort the document by score. False positive documents will get relative lower score, as they do not contain every term of the query. So these documents will be put at the end of results returned to the user.

3. EXPERIMENTAL RESULTS

3.1 Experimental Setup

The performance of all algorithms is measured on a system with AMD Phenom II X4 945 CPU. The GPU cards equipped are Tesla C1060 and GTX480.

We use TREC GOV data set and Baidu data set in our experiments. The GOV data set has about 1.25 million pages collected from gov domain web sites, and the Baidu data set is used in real world search engine instead of a synthetic one, which consists 1.5 million web pages.

3.2 Evaluation

We are concerned about performance of binary search and Bloom filter under different computational threshold [6]. We will review the concept of computational threshold first. $\ell(t_i)$ represent the inverted list for term i, and assume:

$$|\ell(t_1)| \leq |\ell(t_2)| \leq \cdots \leq |\ell(t_k)|, \quad (1)$$

Suppose we receive a stream of queries that give rise to the inverted lists $\ell_j(t_i)$ from the i-th term in the j-th query. The assumption (1) implies that $|\ell_j(t_1)| = \min_i |\ell_j(t_i)|$. In PARA, a CPU continuously receives queries until $\sum_j |\ell_j(t_1)| \geq c$, where c is some desired "computational threshold", and sends the queries onto the GPU as a batch. The threshold indicates the minimum computational effort required for processing one batch of queries. Given a certain amount of queries, larger batch size means less GPU kernel invokes, so the overhead of PCI-E transferring could be reduced. Therefore, larger batch size can better use the processing power of GPU. However, the response time per batch will be longer since the computational effort contained in one batch increases.

As Figure 1 shows, compared with the binary search, the Bloom filter improves the throughput significantly. On GOV data set, when the threshold is set to 1920K, Bloom filter algorithm boosts the throughput by 52.7% to 44204 query/s. The price we pay for such achievement is 1.5x space usage and 0.1% false positive results. As there is only one more false result among 1000 correct results, most users will hardly notice such mistake. Besides, the order of the results is determined by the weight in all inverted index lists after intersection, so an incorrect result can not get high rank since it does not actually exists in all lists. Therefore, incorrect results will not appear in the first few pages. The throughput growth on GOV slows down obviously after threshold goes over 480K, which means we have taken nearly full use of GPU's processing power. The speedup of Bloom filter under low threshold is much worse.

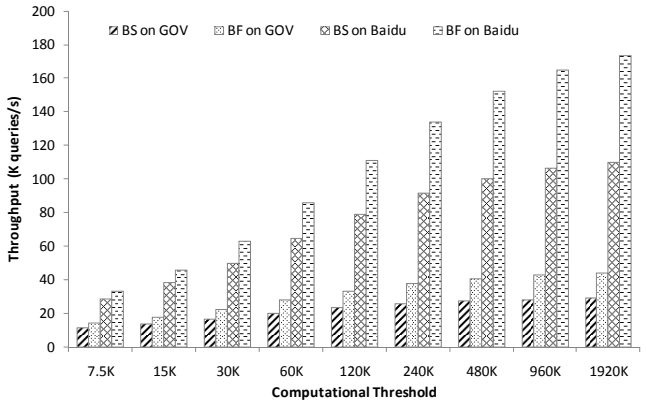

Figure 1: Throughput comparison.

	Throughput on different devices (queries/s)		
	CPU	Tesla C1060	GTX480
GOV	4064	42858	85636
Baidu	21247	164750	303751

Table 1: Bloom Filter performance comparison on different devices. # of hash function: 5, and the threshold is 960K.

GTX480 doubles the number of cores and increases the cores frequency. Besides, GPU memory bandwidth is $1.74x$ compared with the bandwidth of Tesla C1060. In our Bloom filter algorithm, after a GPU thread complete a read operation from global memory, they simply check whether a particular bit is 1. The comparison result determines whether the thread goes on for the next hash function. So the memory bandwidth is the bottleneck of our scheme. Table 1 shows the speedup of Bloom filter algorithm on GTX480 compared C1060 is similar with the bandwidth improvement.

We use the optimized version of skip list [4] as the baseline algorithm on CPU. The algorithm partitions each longer list into several disjoint segments, and jumps with the step of the segment length. Table 1 also demonstrates the throughput of the CPU algorithm. The throughput of Bloom filter algorithm is 10.55x compared with the CPU algorithm on GOV data set, and such speedup increases to 21.07x on GTX480. The intersection throughput of GTX480 has reached 85636 queries/s, which is achieved just by a single node in a server cluster. So the Bloom filter intersection algorithm on GPU improves the system performance significantly.

4. ACKNOWLEDGMENTS

This work was supported in part by the National High Technology Research and Development Program of China (2008AA01Z401) , NSFC of China (60903028,61070014) , RFDP of China (20070055054) , and Science and Technology Development Plan of Tianjin (08JCYBJC13000).

5. REFERENCES

[1] J. Barbay, A. López-Ortiz, T. Lu, and A. Salinger. Faster Set Intersection Algorithms for Text Searching. *ACM Journal of Experimental Algorithmics*, 2006.

[2] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] E. Demaine, A. López-Ortiz, and J. Ian Munro. Experiments on adaptive set intersections for text retrieval systems. *Algorithm Engineering and Experimentation*, pages 91–104.

[4] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Comm. ACM*, 33(6):668–676, 1990.

[5] D. Tsirogiannis, S. Guha, and N. Koudas. Improving the performance of list intersection. *Proceedings of the VLDB Endowment*, 2(1):838–849, 2009.

[6] D. Wu, F. Zhang, N. Ao, G. Wang, X. Liu, and J. Liu. Efficient lists intersection by cpu-gpu cooperative computing. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8, 19-23 2010.