Efficient Privacy Preserving Query Processing using GPGPU

Chongjie Li[†] Toshiyuki Amagasa[†,‡] Hiroyuki Kitagawa[†,]

1. Introduction

Data privacy has been a major concern in nowadays information systems, and hence privacy preserving query processing is becoming a hot research area in the last several years. For this, Agrawal et al.[1] propose a method to execute query operations, such as intersection, join, and aggregation, among different data sources using third parties without disclosing sensitive data. Although it makes it possible to ensure the privacy during query processing, it is suffered from expensive processing cost.

Meanwhile, GPGPU (General-Purpose computation on Graphics Processing Units) has gained its popularity these years because of its great power in mathematical computation, and its rapid growth also leads to more attention.

In this paper, we try to apply the functionality of GPGPU to speed-up this privacy-preserving query processing scheme. Specifically, we exploit CUDA, which is a programming environment for GPGPU, and discuss how the original method can be implemented using the functionality. We also demonstrate the feasibility of the proposed implementation by experiments.

2. Related Work

2.1 Privacy preserving query processing

Agrawal et al.[1] proposed a method for privacy-preserving query processing using third parties. Based on the Shamir's secret sharing method [2], the algorithm generates a degree-k random polynomial q(x) to encrypt data values. The encrypt data values are then forwarded to respective third parties according to the encrypted values, for subsequent processing (Figure 1 a and 1 b). Since data values are encrypted and processed in a distributed environment, it is possible to perform query processing without disclosing the detailed values.

However, computing polynomials and set intersections takes time, which are major cost factor in this algorithm. So, in this work, we attempt to speed up the processing by utilizing GPGPU.

[†] College of Information Science, University of Tsukuba.
[‡] Graduate School of Systems and Information Engineering, University of Tsukuba
[‡] Advanced Research Building B, University of Tsukuba,Tennodai,Tsukuba, 305-8573 Ibaraki, Japan

2.2 Algorithms on GPU

Due to the highly parallel nature of GPU, it shows higher performance in many areas other than graphic processing.

For this reason, many algorithms based on GPU have been proposed. Bitonicsort [3] is an example of such algorithms. The basic idea is to apply fixed number of compare and swap operator to realize sorting. More detailed introduction can be found in Govindaraju et al.[3].

Since GPU program is memory-bonded, many researches have been done to reduce memory access time. Coalesced read and write [4] fully utilizes hardware bandwidth by contiguous memory access.

We attempt to exploit these techniques to accelerate the query processing algorithm.

3. Proposed Method

3.1 Overview

As shown in the following figure, the method is divided into mainly three phases. The phases 1 and 2 are shown in Figure 1(a), and phase 3 is shown in Figure 2(b). Among them, 1 and 3 are done by the data sources and the clients, respectively, while 2 is done in trusted third parties. If we look into each process, phase 1 and 3 are costly to perform. For this reason, we try to utilize GPU for these parts.

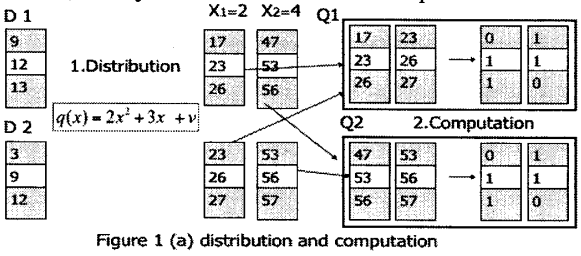

Figure 1 (a) distribution and computation

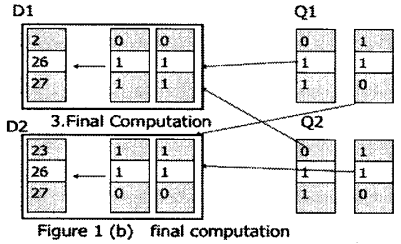

Figure 1 (b) final computation

3.2 Basic method

In phase 0, the data need to be sorted for subsequent process. We simply use bitonicsort in the CUDA-SDK [4] to accomplish this phase. We then encrypt the

secret value by the function f(q(x),v), where q(x) is the randomly chosen polynomial. (For privacy concerns, we can also let coefficient of q(x) be a function of v.) As can be seen, this is one of the computationally intensive parts. We assign each thread of GPU for a different value and compute the polynomials in parallel. Also, we store the temporary result in shared memory to reduce the access latency. In case of large data size which does not fit in the shared memory, we arrange the data into threads with balance.

In phase 2, the only thing for each third party to do is comparing the share-values. Since they are already sorted, this phase can be done in linear time with ease.

Finally, in phase 4, we just compare the result returned from the third parties. We use similar mechanics as phase 0 in which the operations are replaced by logical AND operator ("&").

3.3 Discussion

In the encryption, the encrypted value may become too big to fit in the range of integer numbers, which result in false match. However, most GPU just support single-precision numbers. A possible workaround is to implement integers with arbitrary precision, e.g.,

BigInteger{ char digits[MAXDIGITS]; }

In this implementation, we do not need to stick around numeric numbers, that is, we can use arbitrary strings as each around numeric numbers, that is, we can use arbitrary strings as each digit. For example, we can define the pattern as (a-z|0), thus a 27-based system can be used to generate the corresponding numeric values: "abcde"= $(12345)_{27} = (533795)_{10}$, sure a 32 or 64 based system can be introduced for simplicity or for purpose of adding (A-Z) etc.

4. Experimental Evaluation

Our algorithm is tested on environment as: Core(TM) i7 2.80GHz cpu and 4GB memory; NVIDIA GeForce T220 GPU having 48cores, and memory size of 1275MB.

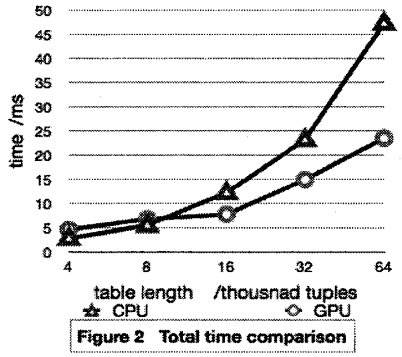

Figure 2 Total time comparison

Figure 2 shows the total time of GPU and CPU realization of 3 phases excluding time for data transfer between clients and third parties and memory allocation. As can be seen from the graph, the advantage of GPU realization revealed when tuple size exceeds 16K.

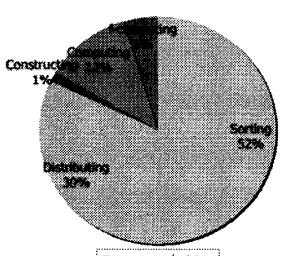

Figure 3 (a) CPU

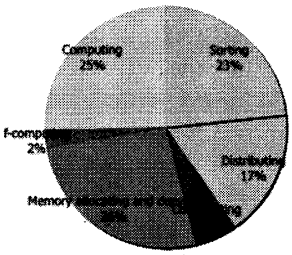

Figure 3 (b) GPU

While graph 3 (a) and (b) indicates the time breakdown of CPU and GPU realization, respectively. We can see that the most compute-intensive part for CPU realization is sorting and distributing, while for GPU realization, the time of memory allocating and coping is also a significant cost.

5. Conclusion

In this study, we propose an efficient privacy preserving query method among multiple data sources. With the help of GPU, we gain a lot of time efficiency comparing with CPU-based realizations. Also our method supports the type of string and allows computing done in high precision. For future work we plan to improve the time of sorting, and dealing with situations while participants are more than two. Also the algorithm for high precision realization is probably to be optimized.

Acknowledgements This research has been supported in part by the Grant-Aid for Scientific Research from JSPS(#21700093).

References

[1] Fatih Emekci, Divyakant Agrawal, Amr El Abbadi, Aziz Gulbeden, Privacy Preserving Query Processing Using Third Parties, Proceddings of the 22rd International Conference on Data Engineering (ICDE'06).

[2] Divyakant Agrawal, Amr El Abbadi, Fatih Emekci,Ahmed Metwally.Database Mangement as a Service: Challenges and Opportunities. Proceedings of the 2009 IEEE International Conference on Data Engineering

[3] N. K. Govindaraju, J. Gray, R. Kumar, and D. Manocha, "Gputerasort: High performance graphics coprocessor sorting for large database management," in Proceedings of ACM SIGMOD International Conference on Management of Data, 2006.

[4] NVIDIA CUDA (Compute Unified Device Architecture),http://developer.nvidia.com/object/cuda.html.