ISSN 2395-1621

Hash Joining Tables using GPU CUDA

[#1]Dayananda, [#2]Rajanigandha, [#3]Pallavi, [#4]Harshal,
[#5]Assistant Prof. Mrs. Smita Kakade

[#1234]Student, Computer Engineering ,
[#5]Assistant Professor, Computer Engineering

SAOE, Kondhwa, Pune.

ABSTRACT

The massive parallelism and faster random memory access of Graphics Processing Units (GPUs) promise to further accelerate complex analytics operations such as joins and grouping, but also provide additional challenges to optimizing their performance. There are more implementation alternatives to consider on the GPU, such as exploiting different types of memory on the device and the division of work among processor clusters and threads, and additional performance parameters, such as the size of the kernel grid and the trade-off between the number of threads and the resulting share of resources each thread will get.

In this system, we analyze the performance of implementing database operations on a GPU versus CPU. A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores and threads designed for handling multiple tasks simultaneously. Using table join operations in a database, we compare the performance of CPU (Central processing unit) and GPU (graphics processing unit). Analyzing the result, we found how GPU is more efficient as compared with CPU in terms of performance ratio. Thus the reason for the wide and mainstream acceptance of GPU is that the it is a computational powerhouse, and its capabilities are growing faster than those of the x86 CPU.

ARTICLE INFO

Article History

Received: 1st February 2017

Received in revised form :

2nd February 2017

Accepted: 5th February 2017

Published online :

6th February 2017

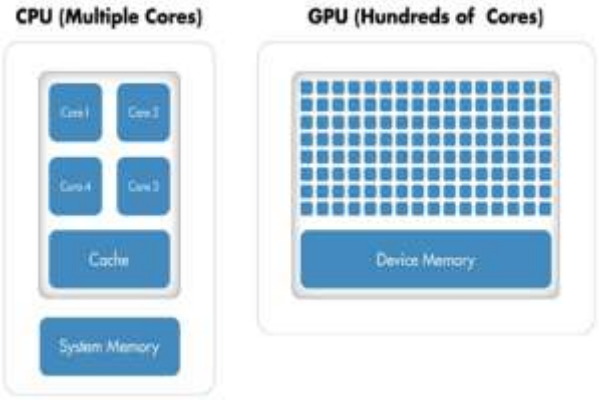
Keywords: Processing Unit, Model Checker, Hash table, CUDA.

I. INTRODUCTION

The research field of formal methods not only focuses on mathematical proofs, but also on automated model checking. Static model checkers[1] check algorithms for possible deadlocks and other unwanted states that are defined by safety properties. To find these problems, a model checker explores the graph of all states the program can reach. A state is represented by a large vector and for each state in this graph, the safety properties are evaluated. Performing the state space exploration requires many computation steps and since the introduction of multi-core processors, large parts of the model checkers are optimized for multi-core systems.
The database that is used to store all those states is often implemented as a hash table. Because it will be accessed every time a new state is visited, the hash table is a critical part of a model checker and has a significant influence on its performance. To increase performance on multi-core systems, proposed a lockless hash table, which is designed with the specific requirements of model checking in mind.[2] The hash table only supports the find or put operation, which finds a state in the database or adds it when it is not found. Because the implementation is not based on locks, it is scalable: even when many threads are storing items in the hash table, the performance of each individual thread does not decrease significantly. The hash table was later improved to a tree database .The tree database combines the performance of a hash table with reduced memory usage.

GPUs have emerged as promising hardware co-processors to speedup various applications, such as scientific computing and database operations.[3] With massive thread parallelism and high memory bandwidth,

GPUs are suitable for applications with massive data parallelism and high computation intensity. One hurdle for the effectiveness of CPU-GPU co-processing is that the GPU is usually connected with the CPU with a PCI-e bus.



On such discrete architectures, the mismatch between the PCI-e bandwidth (e.g., 4–8 GB/sec) and CPU/GPU memory bandwidth (e.g., dozens to hundreds of GB/sec) can offset the overall performance of CPU-GPU co-processing. As a result, it is preferred to have coarse-grained co-processing to reduce the data transfer on the PCI-e bus.[4] Moreover, as the GPU and the CPU have their own memory controllers and caches (such as L2 data cache), data accesses are in different paths.

II. LITERATURE SURVEY

[1] Barnat et al. already showed that GPUs can provide a significant speedup when used for static model checking.

[2] Boˇsnaˇcki et al. extended the model checker PRISM with a CUDA implementation that calculates the transition probabilities Many kinds of data structures can benefit from the parallelism of GPUs, as proved by Misra et al.

The scalability of Laarman's hash table with the massive parallelism of GPUs. The hash table already proved to be scalable up to 48 threads on an x86 CPU

In computing, a **Hash table** (hash map) is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

Model checking or property checking refers to the following problem: Given a model of a system, exhaustively and automatically check whether this model meets a given specification. Typically, one has hardware or software systems in mind, whereas the specification contains safety requirements such as the absence of deadlocks and similar critical states that can cause the system to crash. Model checking is a technique for automatically verifying correctness properties of finite-state systems.

Graphics Processing Unit (GPU), also occasionally called visual processing unit (VPU), is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles

III. METHODOLOGY

We are having huge amount of data of big firm. This data is in the form of dataset and have many tables with various entities. Pass this data to database manager as the CPU and GPU will process data from database manager itself. Tables having keys for identifying uniqueness and relationship among tables. Those keys will be used to create join and form the merged data into single table.

The CPU will work in sequential manner and access database sequentially. Time required by CPU for hash joining is computed and given to performance evaluator. As the data is large, it require more time for computation.

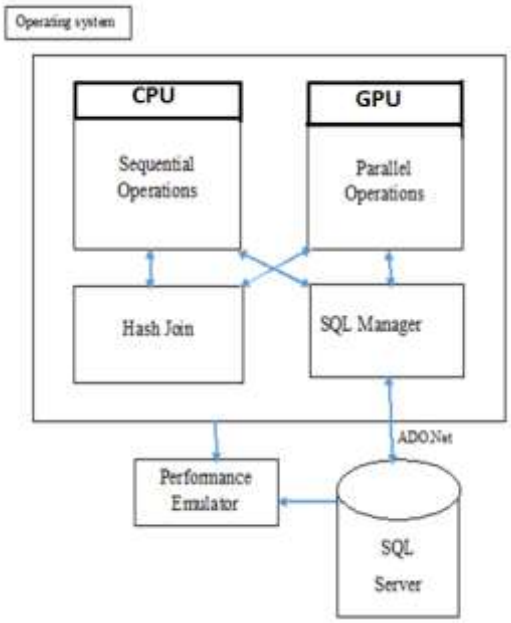
Database provided to GPU perform parallel computation and return the time require for hash joining to performance evaluator. The operation on the GPU Performance evaluator returns the time computed by both CPU and GPU and will show the difference between the computation time .performance of the computation will be shown with the help of progress bar in user interface.

IV. PROPOSED SYSTEM

The modern GPU's parallel architecture gives very high throughput on certain problems, and its near universal use in desktop computers means that it is a cheap and ubiquitous source of processing power. There is a growing interest in applying this power to more general non-graphical problems through frameworks such as NVIDIA's CUDA, an application programming interface developed to give programmers a simple and standard way to execute general purpose logic on NVIDIA GPUs.

In this system performed two operation i.e. sequential operations and Parallel operations.[5] In Sequential

operation, task is performed one after another. In parallel operations, multiple task performed simultaneously. Sequential operation takes more time to execute the task .hence we performed parallel operations for better performance and saving of time. And finally comparison result display on the performance emulator.



V. OBJECTIVE AND SCOPE

A relative speedup of 256 for AMD and 47% for NVIDIA. When the possible optimization of sorting the array of hashes is implemented, this may lead to even better use of caches in the GPU, thus increasing scalability. Further research into the practical applications of our implementation is needed. The hash table can be integrated with a model checker. Then, it is possible to run real-world benchmarks against the hash table. An optimization that may improve the scalability of the hash table on a GPU is the sorting of hashes before inserting them in the database. Future work also needs to focus on the memory limitations of GPUs. A tree database is much more efficient with memory, while it still provides similar scalability as a lockless hash table. Another possible solution is the application of a multi-GPU system. While this requires the hash table to be split across the devices, the total amount of memory bandwidth is increased.

VI. ADVANTAGES

- ✓ To achieve good performance, an efficient division of the work between the CPU and the GPU.
- ✓ Better use of caches in the GPU, thus increasing scalability.

VII. DISADVANTAGES

- ✓ GPU is more expensive.
- ✓ Amount of memory on a GPU is severely limited.

VIII. CONCLUSION

We conclude that the amount of memory on a GPU is severely limited. Even the most expensive GPUs have a memory size of only 12 GB, whereas a high end server may have more than 100 GB of main memory. The potential of a model checker largely depends on the amount of memory it can allocate. When more memory is available, the model checker can explore larger state graphs and check larger models.

REFERENCES

[1] J. Barnat, P. Bauch, L. Brim, and M. Ceˇ ˇ ska. Designing fast LTL model checking algorithms for many-core GPUs. Journal of Parallel and Distributed Computing, 72(9):1083–1097, Sept. 2012. doi: 10.1016/j.jpdc. 2011.10.015.

[2] D. Boˇsnaˇcki, S. Edelkamp, D. Sulewski, and A. Wijs. Parallel probabilistic model checking on general purpose graphics processors. International Journal on Software Tools for Technology Transfer, 13(1):21–35, Oct. 2010. doi: 10.1007/s10009-010-0176-4.

[3] P. Misra and M. Chaudhuri. Performance Evaluation of Concurrent Lock-free Data Structures on GPUs. In 2012 IEEE 18th International Conference on Parallel and Distributed Systems, pages 53–60, Singapore, Dec. 2012. IEEE. doi: 10.1109/ICPADS.2012.18.

[4] F. I. van der Berg and A. W. Laarman. SpinS: Extending LTSmin with Promela through SpinJa. In 11th International Workshop on Parallel and Distributed Methods in verifiCation, PDMC 2012, London, UK, volume 296 of Electronic Notes in Theoretical Computer Science, pages 95–105. Elsevier, September 2012. doi: 10.1016/j.entcs.2013.07.007.

[5] http://fmt.cs.utwente.nl/files/sprojects/152.pdf

[6] http://www.vldb.org/pvldb/vol6/p889-he.pdf

