NESUS cost IC1305
Network for Sustainable Ultrascale Computing

Proceedings of the First PhD Symposium on Sustainable Ultrascale Computing Systems (NESUS PhD 2016) Timisoara, Romania

Jesus Carretero, Javier Garcia Blas
Dana Petcu
(Editors)

February 8-11, 2016

CuDB: a Relational Database Engine Boosted by Graphics Processing Units

Samuel Cremer, Michel Bagein, Saïd Mahmoudi, Pierre Manneback

University of Mons, Belgium

samuel.cremer@heh.be,michel.bagein@umons.ac.be,

said.mahmoudi@umons.ac.be,pierre.manneback@umons.ac.be

Abstract

GPUs benefit from much more computation power with the same order of energy consumption than CPUs. Thanks to their massive data parallel architecture, GPUs can outperform CPUs, especially on Single Program Multiple Data (SPMD) programming paradigm on a large amount of data. Database engines are now everywhere, from different sizes and complexities, for multiple usages, embedded or distributed; in 2012, 500 million of SQLite active instances were estimated over the world. Our goal is to exploit the computation power of GPUs to improve performance of SQLite, which is a key software component of many applications and systems. In this paper, we introduce CuDB, a GPU-boosted in-memory database engine (IMDB) based on SQLite. The SQLite API remains unchanged, allowing developers to easily upgrade database engine from SQlite to CuDB even on already existing applications. Preliminary results show significant speedups of 70x with join queries on datasets of 1 million records. We also demonstrate the "memory bounded" character of GPU-databases and show the energy efficiency of our approach.

Keywords Relational Database, In-Memory, SQLite, GPU

I. Instruction

One of the most common components in many applications is related to database management. Compared to explicit data management (like C/C++ container), the main advantage of a relational database engine is its flexibility in data storage and manipulation. Relational databases are used in enterprise systems (ERP, CRM), in e-business applications (Apache, MySQL, PHP), in many personal applications (FireFox, Skype, GoogleGears, etc.), in embedded systems (iPhone and low cost cellular phones), and also as a native component in OS (e.g. Android and Symbian). With currently more than a billion copies of implementation, SQLite is probably currently the most widely deployed SQL database engine.
In 2004, a first attempt was made to process some database operations with a GPU [1]. At that time, the GPU architectures were not sufficient mature for general-purpose processing. GPGPU frameworks appeared much later. Since the first releases in 2007 of the CUDA framework and in 2009 for the OpenCL framework, it has become common to use GPUs in HPC environments for boosting scientific simulations. Nevertheless, GPUs are not commonly used for boosting database engines. Our goal is to show that a GPU-boosted relational database engine can provide drastic speedups while improving energy efficiency. In this paper we briefly introduce CuDB, a GPU boosted version of SQLite.

II. Related Works

In 2007 appeared GPUQP [2], one of the first experimental relational query processing engine working on a Graphics Processing Unit. With GPUQP, each operator of generated query plans could be processed either on CPU or GPU. The source code did not offi-

cially evolve since 2009 but it contributes to provide a reference database engine for many other contributions. In 2010, two researchers proposed Sphyraena [3], a GPU boosted version of SQLite. Unlike other solutions, Sphyraena does not split the query plans into sequences of parallel primitives which require multiple kernel calls. With Sphyraena, the whole query plan is processed on GPU with a single kernel call. Those previous researches have motivated us to start our own GPU-sided relational database engine. We described some specificities of our GPU-sided database, named CuDB, in a previous paper [4].
Meanwhile other teams started different types of researches, with GPU-database engines as central thematic. Sphyraena was used as base for Virginian [5], with as aim the development of a GPU-adapted table-structure. A group of researchers decide to study the impact of transaction mechanisms within GPU-databases and published the experimental GPUTx engine [6]. The main drawback of GPUTx is that it executes only pre-compiled procedures. Another experimental project is GPUDB [7] which was mainly build to run the Star Schema Benchmark. GPUDB has contributed to prove potential performances of GPU-databases with a reference benchmark.
Another group of researchers wanted to create a database engine which is able to run on different hardware architectures. They used GPUQP as reference engine, and developed the OmniDB [8] engine. The experimental CoGaDB [9] database engine allow the generation of query-plans which are dynamically adapted to the target hardware. Unlike most of previous cited solutions, the online available source code is currently still updated.Note also that two commercial solutions of GPU-sided database engines currently exist [10, 11] and a third database engine just started beta phases [12]. Those commercial solutions are more designed for Geographic Information Systems and the Big Data market. They do not encounter all the issues of a full relational DBMS.

III. THESIS IDEA

Before explaining the internel architecture of CuDB, it is necessary to understand how our reference engine, SQLite, works. SQLite is subdivided into 4 modules: (1) the interface which receive SQL queries, (2) SQL Command Processor which parses the queries and generates query plans, (3) Virtual Database Engine which executes the query plans, and (4) the database. Current version of CuDB engine preserves SQLite API and Command Processor. With CuDB, the Virtual Database Engine and the Database are replaced by our GPU versions. The CPU unit is in charge of parsing queries and translating it into query plans in the first two modules. A query plan is formed by a sequence of opcodes to be processed by a Virtual Machine. Our Virtual Machine is natively designed for GPU parallel architecture as well as our In-Memory Database Engine. This hybrid design was motived by several points: parsing and processing could not expect high speedup although process and storage operation on data can largely benefit of SIMD GPU architectures (several hundreds of synchronized cores). Figure 1 shows the internal architecture of CuDB.

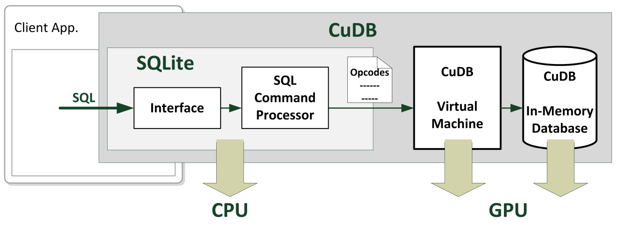

Figure 1: Internal architecture of CuDB.

CuDB engine preserves the original SQLite API, enabling fast, easy and efficient update of existing applications with minor source code updates.To take benefit of the high computation power of GPUs, with GPU-sided virtual machine, each GPU-thread processes the same query plan on its own records, allowing significant speedups with large datasets.
In 2013, a paper specific to the implementation of *SELECT WHERE* and *SELECT JOIN* queries with a GPU-database engine was published [13]. The chosen approach, for the implementation of join operations, was a trivial Cartesian product of tables, which procures a quadratic time complexity. With our engine, we preferred to use a temporary indexation structure for the processing of join-queries, which procure a quasi-linear time complexity. We made performance tests with JOIN queries on two non-indexed tables that are

composed by multiple numerical columns. The selectivity of the queries starts at 10% for small datasets and decreases to 0.1% for the one million row tables. Tables count both the same amount of records. We compared the execution time of CuDB, with a standard SQLite CPU implementation in which tables are stored in RAM memory. The specificities of the hardware we used for this performance evaluation are shown on table 1. Figure 2 shows the average execution time of the multiple join queries.

	CPU	GPU1	GPU2
Reference	Core i7 2600K	GT740	GTX770
Units	4 + HT	384	1536
Frequency	3.8GHz	~1GHz	~1GHz
Bandwidth	21GB/s	80GB/s	224GB/s

Table 1: Hardware specificities

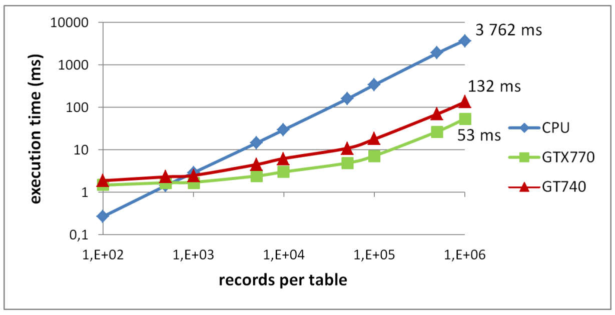

Figure 2: Average execution times with JOIN queries.

Our GPU database becomes as fast as the CPU version when the tables count a minimum of 800 records with GPU1 and 600 records with GPU2. We obtain relevant speedups on large datasets, and even modest GPUs like our GPU1 are able to procure substantial speedups. Our measures also show that performances of our system are clearly memory bounded and depending of query types, the processing time can be more impacted by the memory bandwidth than by the computation power of GPUs.
These results are encouraging but they are produced on non-indexed tables. When the record number of one table increases, performance of a indexed search in $O(log(n))$, running on a single thread CPU, overtakes a trivial parallel brute-force implementation $O(n/p)$, where p is the number of cores. Therefore, we are also currently working on indexation mechanisms for CuDB with better complexity.
During the performance evaluations, we also measured the total power consumption of our platforms. From the measured values we subtracted the idle power consumption to only show the part of energy consumption involved by the computation of the database system. Figure 3 shows the resulting total consumed energy.

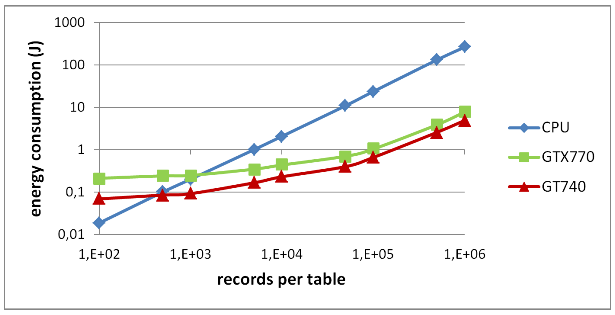

Figure 3: Average energy consumption with JOIN queries.

With our energy consumption tests, we show that the small GPU1 (manufactured in 28 nm) is more efficient than GPU2 (also 28 nm) because of its better "memory bandwidth" over "number of computation units" ratio, what confirms that our GPU database is memory bounded. With CuDB, we are currently working on different types of storage engines with different levels of data compactness and data types. We are also working with SoC architectures to provide a CuDB(m) version which will be dedicated to mobile and embedded applications. Instead of large systems, where the major manufacturers challenge was mainly focused on the processing speed over energy efficiency, small systems dedicated to embedded applications have major energy constraints, particularly due to the portable nature of devices (smartphone, auricular devices). In this field, SoC now offer higher energy efficiency than large systems, mainly due to better integration between components on the same chip (shared memory between CPU and GPU units). So, these small systems using less energy and boosted by environmental constraints, could offer a valuable alternative to existing HPC facilities.

IV. Conclusion and Future Works

In this paper, we have introduced CuDB, a GPU boosted relational database engine. CuDB is based on SQLite and preserves its user interface. We measured relevant speedups while the energy efficiency was increased up to 54 times with large datasets. With join queries, our GPU database always outperforms SQLite when tables counted more than one thousand records. Some significant SQL clauses like ORDER BY are still not being supported by our engine. The SQL support of CuDB needs to be improved, as aiming to run full database benchmarks. We need to deal with the GPU memory limitations and we plan to make a hybrid version of our engine where the CPU cores will process queries on small datasets, while the GPU still manages the greediest processing. We also showed that a GPU-boosted database engine is a memory bounded application. The future GPU architectures with stacked memory will drastically improve the available memory bandwidths. NVidia speaks about 1 TB/s with its next Pascal GPU architecture which will still increase the performances of GPU-database engines.

Acknowledgment

The authors would like to acknowledge the contribution of the Nesus COST Action IC1305.

References

[1] N.K. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manochad, "Fast computation of database operations using graphics processors," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, Paris, France, June 2004, pp. 215-216.

[2] R. Fang, B. He, M. Lu, K. Yang, N.K. Govindaraju, Q. Luo, and P.V. Sander, "GPUQP: query co-processing using graphics processors," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, Beijing, China, June 2007, pp. 1061-1063.

[3] P. Bakkum and K. Skadronr, "Accelerating SQL database operations on a GPU with CUDA," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing*, Pittsburgh, Pennsylvania, March 2010, pp. 94-103.

[4] N. Dechamps, M. Bagein, M. Benjelloun, and S. Mahmoudi, "Boosting Open-Source Database Engines with Graphics Processors," in *Proceedings of the 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Victoria, Canada, November 2012, pp. 262-266.

[5] P. Bakkum and S. Chakradhar, "Efficient Data Management for GPU Databases," 2012. *http : //pbbakkum.com/virginian/paper.pdf* Accessed : 2015-08-11.

[6] B. He, and J. Xu Yu, "High-throughput transaction executions on graphics processors," *VLDB Endowment*, vol. 4, no. 5, pp. 314-325, 2011.

[7] S. Zhang, J. He, B. He, and M. Lu, "OmniDB: towards portable and efficient query processing on parallel CPU/GPU architectures," *VLDB Endowment*, vol. 6, no. 12, pp. 1374-1377, 2013.

[8] Y. Yuan, R. Lee, and X. Zhang, "The Yin and Yang of processing data warehousing queries on GPU devices," *VLDB Endowment*, vol. 6, no. 10, pp. 817-828, 2013.

[9] S. Breß, N. Siegmund, L. Bellatreche, and G. Saake, "An operator-stream-based scheduling engine for effective GPU coprocessing," *Advances in Databases and Information Systems*, vol. 8133, pp. 288-301, 2013.

[10] Parstream, "Parstream - turning data into knowledge," White Paper, November 2010.

[11] GPUdb, *www.gpudb.com*, Accessed : 2015-07-23.

[12] T. Mostak, "An overview of MapD (massively parallel database)," White Paper, Massachusetts Institute of Technology, 2013.

[13] M. Pietron, P. Russek, and K. Wiatr, "Accelerating select where and select join queries on a GPU," *Computer Science (AGH)*, vol. 14, no. 2, pp. 243-252, 2013.