Galactica : A GPU Parallelized Database Accelerator

Keh Kok Yong
Mimos Berhad
Kuala Lumpur, Malaysia
kk.yong@mimos.my

Ettikan K. Karuppiah
Mimos Berhad
Kuala Lumpur, Malaysia
ettikan.karuppiah@mimos.my

Simon Chong-Wee See
NVIDIA Corporation
Singapore
ssee@nvidia.com

ABSTRACT
The amount of business data generated and collected is increasing exponentially every year. A Graphics Processing Unit (GPU) is not used for only optimization of image filtering and video processing, but is also widely adopted for accelerating big data analytics for scientific, engineering, and enterprise applications. However, there are studies pointing out that using GPU as a general-purpose computing device has limitations. In order to exploit current GPU computing capabilities for database operations, we have to take into consideration the characteristics of the GPU and how it can cooperate with the CPU. In this paper, we proposed and implemented a GPU database accelerator, which named Galactica. The experiments result shows proposed GPU database accelerator has outperformed traditional database system. In addition, the Galactica's performance is comparable with a seven nodes distributed Hadoop system. Our results indicate that the GPU is an effective and energy efficient co-processor for executing database operations.

Categories and Subject Descriptors
D.1.3 [**Concurrent Programming**]: Parallel Programming; H.2.4 [**Database Management**]: Parallel Databases; I.3.1 [**Computer Graphics**]: Hardware Architecture-Graphics processors

General Terms
Database, Performance

Keywords
Database, Accelerator, Query, GPU, CUDA, KEPLER, NVIDIA, Parallel Processing, String, Big Data

1. INTRODUCTION
The amount of business data generated and collected is increasing exponentially every year. According to IDC's annual Digital Universe Study, the predicted data growth will be 10-fold from 4.4 zettabytes to 44ZB in 2020 as the Internet of Things (IoT) takes off [1]. Data discovery capabilities are becoming increasingly important for making adequate decisions in increasingly complex business rules and environments. It raises huge challenges to the traditional database system today in dealing with massively stored data and rapidly performing query operations on it. Hence, there is a need to have various types of accelerators to parallelize data and query processing.

A Graphics Processing Unit (GPU) is not used for only optimization of image filtering and video processing, but is also widely adopted for accelerating big data analytics for scientific, engineering, and enterprise applications. Arce accelerates feed-forward neural networks (FFNs) data models, which represents in-matrix operation for financial applications, by using GPUs. It reduces the response time when there is a new set of data arriving [2]. Rungraung parallelizes massive data at very high speed for Automatic Order Matching (AOM) in the Stock Exchange of Thailand (SET) [3]. The ultra-fast analytic application is crucial to drive business success through quick and accurate decision making from mining the massive data.

Over the past decade, GPUs have taken the lead in high performance computing. Its evolution of parallel processing components to fully programmable and powerful co-processors working along with CPU has allowed cheaper, more energy efficient, and faster super computers to be built. Zhe uses a cluster of GPUs with 30 worker nodes to develop a parallel flow simulation using the lattice Boltzmann model (LBM) [4]. The top ranked energy efficient supercomputers in the world, TSUBAME-KFC, Wilkes and HA-PACS, all feature NVIDIA's Kepler GPUs along with high speed network communication links (Infiniband) [1]. These facilities have allowed computational scientists and researchers to address the world's most challenging computational problems by up to several orders of magnitude faster.

In order to exploit current GPU computing capabilities for database operations, we have to take into consideration the characteristics of the GPU and how it can cooperate with the CPU. We propose and implement a GPU accelerated database system called Galactica using CUDA, and benchmark its performance on NVIDIA Tesla Kepler architectures against standard PostgreSQL and various distributed Hadoop systems. The detailed needs for this accelerator and parallel query processing in our work are:

- Partitioning data into fine grained chunks for parallel processing and reducing I/O access
- Applying compression and decompression mechanisms to speedup data I/O operations via PCI-e transfer
- Maximizing the usage of single instruction for multiple data to optimize the degree of parallelism for query operations
- Performance on the GPU implementation should yield a significant acceleration over one based solely on the CPU

2. IMPLEMENTATION
There are several components to formulate Galactica accelerated database system. Figure 1 shows the high level view of the architecture. *Database connector* enables Galactica to connect to

Conference BigDataScience '14, August 04 - 07 2014, Beijing, China

http://dx.doi.org/10.1145/2640087.2644166.

[1] The result was published in November 2013 by the Green500 energy-efficient supercomputers list. The ranking is taken from this website, http://www.green500.org/lists/green201311

any database to perform frontend application interaction, data extraction, query operation and data interchange. *Query processing unit* carries out various processes of query investigation by performing the basic parsing and positioning operations. It targets to break down the input user queries to further clause of objects and collects the decomposed SQL objects to determine either CPU or GPU executable instructions to be used. Then, it produces an execution query plan. There is further adjustment of the plan by analyzing and tracing parallelizable points and rearranges clause of objects execution. Galactica execution engine performs the accelerated query execution on either CPU or GPU. On the other hand, source data from database need to be transformed, and output to a parallel accessible storage system. These components are designed to run on an energy efficient commodity of GPU accelerator. In addition, it powers to strive forward for handling big data challenges.

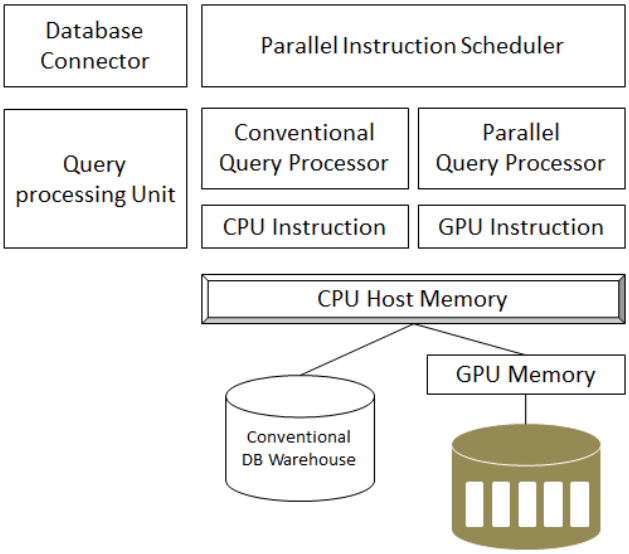


Figure 1: Architecture of Galactica Accelerated Database

Galactica stores data in files format. It is stored in column-based orientation which is designed for optimizing parallel processing by GPU. Input data required to be transferred via database connector to the preprocessor module to formulate the file systems. It partitions a column into multiple chunks of files. Also, there are optimized mechanisms to handle different data type's processes. It also compresses the data into smaller size, which reduces the I/O operation and offloads the task to GPU. We restructured the data processing by co-processing schema structure for given database architecture differently.

We fix the chunk size into total rows for each chunk files. This is to ensure that CPU and GPU have sufficient amount of memory to compute large data set. After our experiment, our test shows twelve millions of rows are performing reasonably well in the use of C2075, K20C and K40. Hype-Q in Kepler architecture supports multiple CPU threads to be launched by chunking jobs on a single GPU simultaneously, thereby intensely rising GPU utilization and reducing CPU idle times. Thus, it further accelerates the preprocessing.

There are dual compression scheme implemented on GPU. First compression scheme applies on Integer data type, it is based on Zukowski work, PFOR-Delta. It store differences instead of actual value. Only the difference of the data is stored between subsequent values. Thus, bit packing mechanism can further optimize it by using just enough number of bits to store each element. Second, Dictionary compression scheme was applied on string data type for much better compression mechanism. It has a dictionary of the unique values in the chunk file for decompression. As such, ultra-fast retrieval on the dictionary is the key of the efficiency. It offloads the lookup process to the GPU. However, this string compression mechanism is only benefited with a lot of duplicated words in the input column of data.

There are a few modules involved to process user query before executed in GPU. The query needs to break into clause of objects, and then it composes the clauses in GPU parameters mapping APIs (Application Programming Interface) table. The API tables are a set of functions to perform GPU operations, such as scan, sum, min, max and others. After forming the table, there is an analysis on reshuffling and rearranging the order of mapping APIs in the table. Finally, it transforms to a set of parallel instructions, which are ready to be executed in GPU. However, Galactica does not support all kind of queries, so it assigns a flag to indicate none GPU executable instruction sets. These are the important steps to strategically execute query in GPU.

GPU scheduler is responsible for keeping the serialized condition for all parallel instruction sets execution paths. It manages the query execution queue either in CPU or GPU. Instruction sets call the GPU executable APIs to perform the parallel operations. The operations that are performed in GPU are such as select, sort, projection, joins and basic aggregation. It utilizes the mixture of in-house built accelerated parallel processing library MiAccLib and open source library, such as Thrust. Those flagged with non GPU executable instruction sets, it then diverted to a standard conventional query engine via the database connector to the CPU execution.

There are few critical areas that govern the performance of the GPU APIs implementation. We use the concurrency through pipelining mechanism to overlap the data transfer via PCI-e bus and the arithmetic computation. These CUDA streams can be executed asynchronously, which queues the works and return to CPU immediately. We avoid the branching divergence of evaluating condition in the kernel. It can drastically slow down the parallel operation in GPU as all threads in a warp not following the active path execution.

3. EXPERIMENTAL RESULTS

There are two exhaustive empirical evaluations that have been carried out in this work. The evaluation is performed on different hardware devices using different kinds of database and distributed systems. It tests the robustness and effectiveness of the system. Section 3.1 evaluates three different query executions in Galactica against traditional database systems by using various sizes of TPC-H dataset. Section 3.2 evaluates various query executions in Galactica against different distributed big data systems. We time the execution period by adding fixed number of data records for each test run. The query execution does not include the data preprocessing. In addition, we compare both returning output from queries. It is to ensure the correctness and accuracy of the results on both systems.

3.1 Experiment 1

This experiment is to scrutinize the performance between a traditional database management system, namely PostgreSQL, and Galactica. The queries consist of a series of aggregation

functions in order to increase the computational workloads. In addition, it consists of a series of table joining and string comparison operations using various sizes of input data. Table 1 shows the actual SQL queries in this experiment.

3.1.1 Hardware and Operation System

We performed this experiment in two different variants of Tesla Kepler series of GPU computational devices:

i. Kepler K20c with 2496 CUDA cores and 6GB GDDR5 RAM, launched in 2013
ii. Kepler K40, with 2880 CUDA cores and 12GB GDDR5 RAM, launched in 2014

These GPU devices are separately set up in two HP Z800 workstations with the same configurations. Each of them contains Dual Intel Xeon X5680 (6 cores) processors clocked at 3.33GHz, 32GB RAM memory, 1TB Hard Disk, and ATI FireMV 2260 display card. For software, it has Microsoft Windows 7 (64bits) and PostgreSQL 9.2 database system in both of the workstations. The workstation hosting K20c runs on CUDA 5.5 (Production Release), while the K40 workstation host runs on CUDA 6 (Release Candidate).

Table 1: Queries for Experiment 1

Query 1	Query 2	Query 3
SELECT l_returnflag, l_linestatus, sum(l_quantity) AS sum_qty, sum(l_extendedprice) AS sum_base_price, avg(l_quantity) AS avg_qty, avg(l_extendedprice) as avg_price, min(l_quantity) AS min_qty, max(l_quantity) AS maxqty, min(l_extendedprice) AS min_price, max(l_extendedprice) AS max_price, count(l_quantity) AS count_order FROM lineitem WHERE l_shipdate <= 19980902 GROUP BY l_returnflag, l_linestatus ORDER BY l_returnflag, l_linestatus;	SELECT customer.c_mktsegment, orders.o_orderdate, orders.o_shippriority, sum(lineitem.l_extendedprice) AS revenue, sum(lineitem.l_quantity) AS quantity, avg(lineitem.l_quantity) AS avg_qty, avg(lineitem.l_extendedprice) AS avg_price, count(orders.o_orderkey) AS total_order FROM customer, orders, lineitem WHERE customer.c_custkey = orders.o_custkey AND lineitem.l_orderkey = orders.o_orderkey AND orders.o_orderdate < 19950315 GROUP BY customer.c_mktsegment, orders.o_orderdate, orders.o_shippriority;	SELECT sum(lineitem.l_extendedprice) AS revenue, sum(lineitem.l_quantity) AS quantity, count(orders.o_orderkey) AS cnt FROM customer, orders, lineitem WHERE customer.c_custkey = orders.o_custkey AND lineitem.l_orderkey = orders.o_orderkey AND customer.c_mktsegment = 'BUILDING';

3.1.2 Datasets

Galactica and PostgreSQL have been tested on TPC-H's data sources. TPC-H implements an ad-hoc decision support benchmark. The ad-hoc nature of the benchmark is intended to mimic a real-life use case for businesses. The database administrators (DBAs) do not know which queries are executed against the database system. It poses a more challenging workload that is more customer-relevant and has garnered the support of the industry for database engines. We use the TPC-H data generator application to create three different sizes of datasets namely 1 GB, 10 GB, and 50 GB.

3.1.3 Empirical Results

This section provides the experiments and results carried out with the datasets mentioned above. The results include the overall timing from issuing queries, executing queries in the GPUs, processing output results, and displaying final results on the screen. Figure 2 shows the results from three sets of experiments. A logarithmic scale is used for the x-axis to represent execution times of the CPU and GPUs results as the ratio between is the results are extremely large. All running time value is expressed in seconds.

The experiments consist of running three different sets of queries against the three datasets with varying sizes on three different computing processors and accelerators. These queries are executed ten times each, and the chart shows the average times. The results for "PostgreSQL" were obtained from CPU execution times alone, while "Galactica K20c" and "Galactica K40" were run on the NVIDIA K20c and K40 GPUs, respectively. Both Galactica benchmarks were obtained using the same set of Galactica source codes compiled using different versions of the NVCC compilers, with CUDA 5.5 used for the K20c and CUDA 6.0 for the K40.

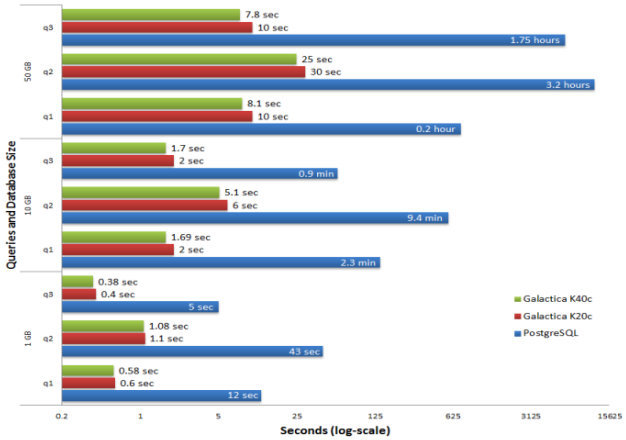


Figure 2: Queries Execution with the TPC-H dataset

3.1.4 Analysis of Results

The proposed GPU database accelerator system achieves a speed up between 39.8x and 807.6x for NVIDIA Kepler GPU. The result shows a multiple orders-of-magnitude performance gains in utilizing GPU. It enables an increase in the number of queries that can be executed and supported in computing a larger dataset. It demonstrates the proposed system is energy efficient.

Despite the size of the input dataset growing from 1GB to 50GB for the queries computation, Galactica exhibited no significant degradation due to utilization of GPUs as accelerators. The queries execution for Galactica completed within a minute, even for the largest dataset. PostgreSQL suffers in using one core per database connection, as those queries take hours when executed on the largest dataset. For example, the second query takes an average of 3.2 hours to complete on PostgreSQL. However, Galactica completes the query in an average time of 30 seconds. This result has indicated that Galactica has good potential for big data analysis.

3.2 Experiment 2

This experiment involves the execution of 6 different queries; each is executed 5 times on a multi-node Hadoop cluster. The queries are executed on a 32GB health care database. Table 2 contains the description of the queries. Each query will have significantly different response times depending on the approach used. In addition, we perform the comparison between a single workstation and a multi-node distributed system.

3.2.1 Hardware

i. Hadoop based Distributed System:
7 Virtual Machines (6 workers and 1 master node) which are distributed across four HP DL380p G8 servers with 12 cores and 32GB RAM

ii. GPU Workstation:
DELL T5500, Intel® Xeon® E5630 @ 2.53GHz with Quad Core Processor, NVIDIA Tesla K20c, 8GB RAM, WD HDD 1 TB, Windows Server 2008 R2 Enterprise 64-bit

3.2.2 Datasets

It uses the test data provided by a local Healthcare sector member. The size of the data source is 32 Gigabytes, which is stored in a PostgreSQL database. There are 25 tables in the entire database and one of the tables contains 120 million of records. The data types in the columns of the tables consist of integers, floats, strings, and dates.

Table 2: Queries for Experiment 2

	Description of Queries
Q1	Selecting sum from one column of 120 million records
Q2	Selecting a name column, counting the name and ordering by top 10 names
Q3	Selecting state code, years from hospital patient records with one disease code selected, group by years and state code, order by years and state code
Q4	Selecting state, years, disease name from hospital patient records where one disease name type is selected and joining disease code with disease name and state code with state names, grouping by years, states and disease names, ordering by years and state code.
Q5	Inserting the results of Selecting state code, years from hospital patient records with ALL diseases type, group by years and state code, order by years and state code
Q6	Selecting state, years, disease name from hospital patient records where three disease name types are selected and joining disease type with disease name and state code with state names, grouping by years, states and disease names, ordering by years, states and disease names

3.2.3 Empirical Results

The average time for each query type was analyzed for PostgresSQL database and Galactica on the DELL T5500 workstation; Hadoop based distributed system on the 7 virtual machines. There are Apache Hadoop, Hadoop with Hive, Impala with Hive, and a GPU server in order to compare the processing time (in seconds). Figure 3 shows the performance comparison of various setups for real time Analytics Processing of Big Data in the Health sector, using Mi-BIS [2] Presentation Dashboard (Integrated with Galactica) to analyze ~120 millions of records in Hadoop and Postgres (RDBMS) servers. Some of the results are reported as NA (Not Available) because the work is still in progress.

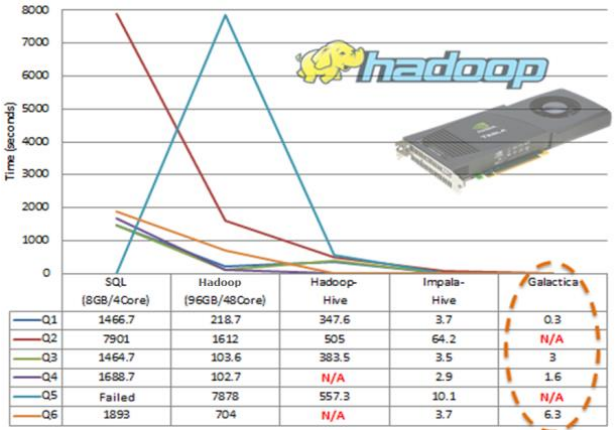
3.2.4 Analysis of Results

Galactica takes the shortest time to process ~120 million records and the cost is also cheaper than implementing 7 nodes of Hadoop or SQL on PostgresSQL (medium and high performance fully tuned PostgresSQL database server). SQL was not able to compute on big data especially for real time analytics. It even failed in the insertion of output data in the same database. Hadoop with Hive is not suitable for real time processing, and would only be useful for batch processing of big data. Impala-Hive is as beneficial as GPU for general queries and could be used to complement the hybrid parallel processing approach especially led by the orchestration engine with predefined rules within the scripts, developed for the middleware layer. Impala Hive is faster compared to Galactica when there are multiple tables to be joined and with huge strings operations to be performed.

[2] Mi-BIS is a MIMOS proprietary Business Intelligence Suite. It creates the reports for the types of query (or SQL select statements) for the different API/connector's.

4. Conclusion

We have presented Galactica as a GPU database accelerator system for assisting the traditional databases in performing a big data analytics. It applies on any data analytic operation by using SQL statement, such as Data Cleansing as an example application. Both of the results from the experiments has certainly shown the GPU based solution is an alternative to existing Hadoop and MapReduce like applications for Big Data processing. Our GPU accelerated database approach has facilitated seamless integration to other front end application via a database connector. In the future, we are going to apply this accelerator to other areas such as Geographic Information System, within the applications using SQL-liked operation. Galactica strives towards the full support of the SQL92 standard and enabling parallel accelerated query processing.



	SQL (8GB/4Core)	Hadoop (96GB/48Core)	Hadoop-Hive	Impala-Hive	Galactica
Q1	1466.7	218.7	347.6	3.7	0.3
Q2	7901	1612	505	64.2	N/A
Q3	1464.7	103.6	383.5	3.5	3
Q4	1688.7	102.7	N/A	2.9	1.6
Q5	Failed	7878	557.3	10.1	N/A
Q6	1893	704	N/A	3.7	6.3

Figure 3: Comparison of Query Execution

5. ACKNOWLEDGMENTS

This research was done under Joint Lab of "NVIDIA - HP - MIMOS GPU R&D and Solution Center". This is the first GPU solution center in South East Asia established in October 2012. Funding for the work came from MOSTI, Malaysia. The authors would like to thank Pradeep Gupta from NVIDIA for the supports.

6. REFERENCES

[1] A. Adshead, "Data set to grow 10-fold by 2020 as internet of things takes off," ComputerWeekly.com, 9 April 2014. [Online]. Available: http://www.computerweekly.com/news/2240217788/Data-set-to-grow-10-fold-by-2020-as-internet-of-things-takes-off. [Accessed 10 May 2014].

[2] P. Arce, C. Maureira, R. Bonvallet and C. Fernandez, "Forecasting high frequency financial time series using parallel FFN with CUDA and ZeroMQ," in Information and Telecommunication Technologies (APSITT), Santiago and Valparaiso, Republic of Chile, 2012.

[3] K. Rungraung and P. Uthayopas, "A High Performance Computing for AOM Stock Trading Order Matching using GPU," in Computer Science and Engineering Conference (ICSEC), Nakorn Pathom, Thailand, 2013.

[4] Z. Fan, F. Qiu, K. Arie and S. Yoakum-Stover, "GPU Cluster for High Performance Computing," in Proceedings of the ACM/IEEE SC2004 Conference, Pittsburgh PA, 2004