LETTER  Special Section on Data Engineering and Information Management

An Experimental Study across GPU DBMSes toward Cost-Effective Analytical Processing*

Young-Kyoon SUH*(a), Member, Seounghyeon KIM(b), Joo-Young LEE(c), Hawon CHU(d), Junyoung AN(e), and Kyong-Ha LEE††(f), Nonmembers

SUMMARY In this letter we analyze the economic worth of GPU on analytical processing of GPU-accelerated database management systems (DBMSes). To this end, we conducted rigorous experiments with TPC-H across three popular GPU DBMSes. Consequently, we show that co-processing with CPU and GPU in the GPU DBMSes was cost-effective despite exposed concerns.

key words: GPU, database management systems (DBMS), analytical processing, cost-effectiveness, economic perspectives

1. Introduction

Remarkable technological advances have led to explosive growth and demand of Graphical Processing Unit (GPU) in many diverse application areas. Riding this wave, the database community has also exerted to realize GPU acceleration on large-scale data processing. An earlier study [1] tried to accelerate SQLite directly on GPU. Another study accelerated the performance of an indexed nested-loop join operator in VoltDB by applying GPU [2]. These endeavors paid off by yielding GPU database management systems (DBMSes) [3]–[5] to accelerate query processing on massive data.

However, there has been less effort to empirically demonstrate how well the GPU DBMSes perform. More specifically, how cost-effective utilizing GPU in analytical processing on large-scale data is in real GPU DBMSes has not been well studied. Due to the lack of such work, users face challenges in deciding to purchase extra, expensive GPUs with their limited budget, to boost the performance of their legacy services.

In addition, the users may raise several questions as follows:
1) Is utilizing GPU economically good for large-scale analysis? 2) Can a GPU DBMS generally outperform a CPU DBMS? 3) Should a newer GPU architecture or multiple GPUs be adopted for the GPU DBMS to further enhance its performance? 4) Do the GPU DBMS scale well with growing data? 5) What kind of hardware and software setups do make utilizing the GPU DBMS most economical?

To answer these questions, this letter conducts a cost-effectiveness study on real-world GPU-accelerated DBMSes in large-scale data analysis. More specifically, we empirically evaluate the performance of the GPU DBMSes with TPC-H benchmark [6] and then draw several performance implications from the results. We also propose a sophisticated empirical methodology for this study. Our results indicate that utilizing GPU is fairly cost-effective in analytical processing on the DBMSes despite revealed scalability concerns. Based on these analysis results, we recommend the best hardware setup for a GPU DBMS based on database size, as shown in Table 1 in Sect. 4.

Certainly, there have been several empirical studies [7]–[9] that reported the performance of GPU-based data processing engines. However, their works focused on only a single GPU DBMS or their own prototypes rather than real-world GPU DBMSes. Our recent study raised some performance issues observed from several real-world GPU DBMSes [10]. Nevertheless, none of these works addressed the “economic worth” of utilizing GPU in analytical processing under a low budget circumstance. To the best of our knowledge, this is the first work to empirically assess the economic validity of GPU utilization across multiple state-of-the-art GPU DBMSes.

Our contributions are summarized as follows.

- We conduct an empirical study across multiple real-world GPU DBMSes with TPC-H workloads.
- We propose a novel methodology for empirically studying GPU DBMSes in a holistic way.
- We assess the economic worth of GPU utilized for analytical processing in the DBMSes. Based on our findings, we confirm the cost-effectiveness of co-processing with CPU and GPU in the DBMSes.
- We recommend the best configuration settings for a given GPU DBMS according to database size.

The following section briefly introduces three GPU-accelerated DBMSes that we choose for our study. In Sect.3 we propose a novel empirical evaluation method-
ology for the DBMSes. Section 4 then presents evaluation results obtained along with the proposed methodology and draws several economic implications. Finally, Sect. 5 concludes this letter by summarizing our contributions and mentioning future work.

2. GPU-Accelerated DBMSes

For our study we considered quite a few GPU DBMSes, among which we selected the following three systems: BlazingSQL [3], OmniSci DB [4], and PG-Strom [5]. These systems passed our criteria as of whether or not (1) the executable code of a DBMS is available, (2) the code accompanies no fee, (3) the code is open-source, and (4) the DBMS can support relational queries.

First, BlazingSQL is an open-source, distributed GPU-accelerated SQL engine [3]. Data in BlazingSQL are organized in the columnar memory format provided by Apache Arrow [11]. Query compilation is performed by using RAPIDS [12] and vectorized query instructions are executed on the GPU using SIMD.

OmniSci DB (hereafter, OmniSci) [4] is another open-source SQL engine supporting columnar data format. OmniSci is designed to exploit massive parallelism on heterogeneous computing environments consisting of both CPUs and GPUs. To boost its performance, OmniSci keeps hot data in GPU memory to reduce transfer I/O overheads over the PCIe bus. OmniSci also supports hybrid query execution through parallelized code, allowing a processor to evaluate on multiple data items at the same time. OmniSci can parallelize computation across multi-GPUs and multi-CPUs as well as run on CPU-only systems. Inter-GPU communications can also be configured via NVLink.

Lastly, PG-Strom [5] is an extension devised for PostgreSQL, aiming at accelerating processing SQL workloads for big data by exploiting GPGPU technology. Currently, PG-Strom supports scan, join and group-by operations on GPUs. A major advantage of PG-Strom is that since it is in the form of a module, PG-Strom can be easily configured in an environment where PostgreSQL is deployed. PG-Strom has some advanced features, such as SSD-to-GPU Direct SQL, multiple GPUs, and columnar store, to reduce the overhead of disk I/Os during query processing. The support of multiple GPUs is only available through a paid license.

3. Proposed Methodology

Figure 1 illustrates our proposed methodology that exploits the TPC-H benchmark, which can also be replaced with a custom workload or other types of TPC benchmarks. First, we configure a target GPU DBMS server with system parameters such as GPU model, GPU type (single or dual), and RAM size. Then, we switch off noncritical daemon processes that are identified in advance due to the possibility of having a strong impact on experiments. In turn, we populate a database(s) for query workloads. In this example, the TPC-H database is populated along with a scale factor (SF) specified in the experiment. We then run a query on that DBMS. Running that query is repeated n times, and for each run, we set both nvprof [13] and timer on and off before and after the query execution, respectively. Once n runs are completed, the measured profiling and timing data are recorded into a repository. Here, we use MongoDB for schema flexibility, but the proposed methodology does not prefer a specific system for collection. Finally, the stored data are agglomerated and analyzed from an economic point of view.

4. Experimental Evaluation

In this section, we describe our environment settings, propose two evaluation metrics for our analyses, and present the evaluation results across the GPU DBMSes.

4.1 Environment Settings

Our experiments were conducted on a cluster of three homogeneous servers, each of which was equipped with one six-core CPU Intel i7-8700K @ 3.7 GHz, 48 GB RAM (three Samsung 16GB DDR4 PC4-19200s), and 1 TB NVMe SSD (Samsung m.2 type). For GPU, we used either NVIDIA GeForce GTX 1080 Ti 11GB or NVIDIA GeForce RTX 2080 Ti 11GB. For some experiments we installed two
1080 Ti GPU cards into the servers. All the experiments were run on CentOS 7.7 with a kernel of 3.10.0 and a CUDA driver of 10.2.

4.2 Evaluation Metrics

To assess the economic worth of exploiting GPUs in query processing, we devised two metrics inspired by a previous study [14]. The first metric concerns data processing performance. More specifically, we examine how fast a GPU DBMS is capable of processing a query on a populated database. The TPC-H database can be scaled up in accordance with a given scale factor (SF). SF = 1 indicates a capacity of approximately 1,372 Megabytes (MB). Given this fact, we define the first metric, termed MPS. This metric represents the amount of data in MB that the DBMS can process per second. Equation (1) defines MPS as follows:

\[ MPS = \frac{SF \times 1,372MB}{QT} \]

where QT is query processing time in seconds.

The second and key metric explains how much monetary cost should be paid to obtain the data processing performance as defined in Equation (2):

\[ MSK = \frac{MPS}{C} \]

where C represents $1,000 as unit cost. MSK is expressed as MPS divided by the total hardware cost for building a physical server; the higher the MSK, the higher the performance at the same price. To calculate C, we referred to price information on a well-known US website [15]; namely, about $363 for the CPU, $1,165 for the GTX 1080 Ti, $1,454 for the RTX 2080 Ti, and $938.15 for all the other hardware combined (For a discontinued component, we use the price of its replacement model with similar specifications).

4.3 Evaluation Results

The first experiment (Fig. 2) explores two aspects: (I) how cost-effective co-processing with both CPU and GPU is in large-scale data analysis, and (II) what kind of GPU setup leads a DBMS to be most cost-effective. In the experiment, we tried to configure a GPU DBMS server with four different setups: (i) one 1080 Ti, (ii) one 2080 Ti, (iii) two 1080 Tis, and (iv) two 2080 Tis. As a result, all the three GPU DBMSes allowed for a single GPU mode ((i) and (ii)), but OmniSci only allowed for the dual modes ((iii) and (iv)). For comparison purpose, we also included in the experiment PostgreSQL using CPU only. On the four DBMSes with these various setups, we ran all TPC-H queries, except a few queries that failed. Figure 2 displays the results from six representative queries, and the rightmost group denoted as AVG represents MSK averaged over all the queries executed on the respective setup.

Concerning (I), OmniSci with a single 1080 Ti was most economical based on MSK. More specifically, OmniSci outperformed PostgreSQL using only CPU by 9x, PG-Strom by about 44x, and BlazingSQL by 1.5x. However, PG-Strom was almost 5x less economical than CPU-based PostgreSQL. From these results PG-Strom was the “least cost-effective” GPU DBMS at SF = 8.

Regarding (II), BlazingSQL economically preferred 2080 Ti to 1080 Ti while PG-Strom and OmniSci did not reveal such preference. In addition, the dual GPU setups certainly offered some performance gain, which was not so cost-effective as that of a single GPU setup. We see that a single 1080 Ti was the best economic configuration across the GPU DBMSes.

The last experiment relates to how MSK changes over growing databases. Figure 3 exhibits the result. OmniSci with dual 1080 Tis showed the optimal economic performance up to SF = 32. But when BlazingSQL and OmniSci were configured with a single 1080 Ti, they revealed scalability concerns beyond SF = 8 and SF = 16, respectively. On the other hand, PostgreSQL and its GPU extension module, PG-Strom, scaled fairly well. From the perspectives of scalability, considering the rising trend starting from SF = 64, PG-Strom is expected to be the most economically powerful on large-scale data beyond SF = 128.
4.4 Summary

Several economic implications can be drawn. First, a DBMS utilizing GPU and CPU together was more economical than a DBMS relying on CPU only, considering the performance per dollar. In Fig. 2, BlazeSQL and OmniSci applying the co-processing overwhelmed PostgreSQL by almost up to 10x. Considering that PG-Strom was five times less economical than PostgreSQL, however, GPU may not be necessarily economical for certain DBMSes.

Second, the degree of cost-effectiveness of utilizing GPU varied across the GPU DBMSes. For 1080 Ti, BlazingSQL was about 30x more economical than PG-Strom, and OmniSci, about 1.5x more economical than BlazingSQL, in Fig. 2. Also, Figure 2 implies two things. One is that the newer and more expensive GPU model (2080 Ti) did not always yield a better economic advantage than the older and cheaper GPU (1080 Ti) in a DBMS like OmniSci. Another is that dual GPUs were not always economical compared to a single GPU when applied to a DBMS like OmniSci.

Third, the studied DBMSes reveal different strengths in various aspects. Figure 4 shows integrated views of our analyses based on scalability, MSK, and MPS across the DBMSes. PostgreSQL and PG-Strom have great strengths in scalability. OmniSci reveals the best cost-effectiveness (OmniSciM indicates OmniSci with multiple GPUs). BlazingSQL, however, exposes some inferiority to the other DBMSes, regarding scalability, performance, and investment. One thing to emphasize, though, is that when used with 2080 Ti, BlazingSQL becomes more economical. Namely, its cost-effectiveness is similar to that of OmniSci with 1080 Ti, or approximately 1.5% lower.

Table 1 The best configuration per DBMS we studied based on a database scale

| Database Scale | DBMS          | GPU Type        |
|----------------|---------------|-----------------|
| SF ≤ 16        | OmniSci       | 1080 Ti         |
| SF = 32        | OmniSci       | 1080 Ti + 1080 Ti|
| SF = 64        | PostgreSQL    | -               |
| SF = 128       | PostgreSQL    | -               |
| SF > 128       | PG-Strom      | 1080 Ti         |

This implies that BlazingSQL can produce better cost-effectiveness when employing a higher-end GPU architecture for its analytical processing.

Lastly, based on our results we can derive an economically desirable GPU setup for a DBMS, as shown in Table 1. When SF is between 32 and 128, using PostgreSQL without exploiting GPU can be the best option. When SF ≤ 32, it is most economical to use OmniSci with 1080 Ti installed. When managing and analyzing data beyond SF = 128, PG-Strom with one 1080 Ti can be the most economical option.

In summary, our analysis results imply that a GPU DBMS can cost-effectively replace a CPU-based DBMS for better analytical processing on large-scale data.

5. Conclusion and Future Work

In this letter, we proposed an empirical methodology for GPU DBMSes and analyzed the economic worth of their analytical processing in various aspects. We will further assess their cost-effectiveness with more configurations.

References

[1] P. Bakkum and K. Skadron, “Accelerating SQL database operations on a GPU with CUDA,” GPGPU’10, pp.94–103, 2010.
[2] A. Nguyen, M. Edahiro, and S. Kato, “GPU-Accelerated VoltDB: A Case for Indexed Nested Loop Join,” HPCS’18, pp.204–212, 2018.
[3] BlazingSQL Inc., https://blazingsql.com/
[4] OmniSci, https://omnisci.com/platform/omniscidb
[5] PG-Strom, https://heterodb.github.io/pg-strom/
[6] Transaction Processing Performance Council, “TPC-H,” http://www.tpc.org/tpch/default.asp, 2020.
[7] E. Furst, M. Oskin, and B. Howe, “Profiling a GPU Database Implementation: A Holistic View of GPU Resource Utilization on TPC-H Queries,” DaMoN’17, pp.3:1–3:6, 2017.
[8] P. Chrysogelos, et al., “Hardware-conscious query processing in GPU-accelerated analytical engines,” CIDR’19, 2019.
[9] A. Raza, et al., “GPU-accelerated data management under the test
of time,” CIDR, 2020.

[10] H. Chu, S. Kim, J.-Y. Lee, and Y.-K. Suh, “Empirical Evaluation across Multiple GPU-Accelerated DBMSes,” DaMoN’20, pp.16:1–16:3, 2020.

[11] Apache, https://arrow.apache.org/

[12] NVIDIA, https://developer.nvidia.com/rapids

[13] NVIDIA, https://docs.nvidia.com/cuda/profiler-users-guide/index.html, 2020.

[14] E. Danovaro, A. Clematis, A. Galizia, G. Ripepi, A. Quarati, and D. D’Agostino, “Heterogeneous Architectures for Computational Intensive Applications: A Cost-Effectiveness Analysis,” JCAM, vol.270, pp.63–77, 2014.

[15] The Newegg website, https://www.newegg.com/. 2020.