Workload-Driven Data Placement for GPU-Accelerated Database Management Systems

Christopher Schmidt[1], Matthias Uflacker[1]

Abstract: An increase in the memory capacity of current Graphics Processing Unit (GPU) generations and advances in multi-GPU systems enables a large unified GPU memory space to be utilized by modern coprocessor-accelerated Database Management System (DBMS). We take this as an opportunity to revisit the idea of using GPU memory as a hot cache for the DBMS. In particular, we focus on the data placement for the hot cache. Based on previous approaches and their shortcomings, we present a new workload-driven data placement for a GPU-accelerated DBMS. Lastly, we outline how we aim to implement and evaluate our proposed approach by comparing it to existing data placement approaches in future work.

Keywords: GPU-Acceleration, Coprocessor-Accelerated Databases, Data Placement

1 Introduction

Advances in hardware lead to an increased specialization of processors. For optimal performance a DBMS is advised to utilize all available hardware resources. GPUs raised an interest as a coprocessor for DBMSs, particularly for main memory column stores, due to their high memory bandwidth and parallel compute capabilities for query processing [Br14; Mo13; YLZ13]. Yet, their memory capacity is limited and data transfer, via PCIe, between on-device and host memory remains a bottleneck [GH11]. Note that with NVLink a faster interconnect is introduced, but it still lacks support by major CPU vendors, which rely on PCIe. Considering these challenges, query processing in coprocessor environments follows one of two major execution models.

The first execution model streams data into the device memory, enabling the execution of one or multiple operators, before results are transferred back into host memory. Despite of recent improvements by Funke et al. [Fu18], where the bandwidth transfer bottleneck is pushed from the interconnect to the GPU memory, initial data transfer accounts for a significant portion of the total execution time.

The second execution model limits query processing to data that is already resident in GPU memory to avoid data movement, in case it is harmful to query execution time [BFT16].

[1] Hasso Plattner Institute, Enterprise Platform and Integration Concepts, August-Bebel-Str. 88, 14482, Potsdam, vorname.nachname@hpi.de

doi:10.18420/btw2019-ws-08

Using the GPU memory as a hot cache requires a periodic data placement, which ensures to provide relevant data for query processing in the hot cache. This approach is limited by the device memory.

Recent GPU generations have seen an increase in memory capacity. Furthermore, combining multiple GPUs in a single system enables an even larger unified GPU memory space. Due to this relaxation of the on-device memory limitations, we see potential to revisit the idea of using GPU memory as a hot cache. Based on limitations in previous data-driven approaches, we propose a novel strategy for the data placement in GPU-accelerated DBMS with the goal to reduce the overall execution time for a given workload.

2 Related Work

GPU-accelerated DBMSs are an active area of research, aiming to fully utilize the hardware capabilities by implementing operators specifically for the GPU [He08] and by tackling challenges when integrating the coprocessor as an additional execution unit into the DBMS, i.e., operator placement [Br14; KHL17] or data placement [BFT16; Mo13].

In CoGaDB [BFT16], data placement is implemented as a background job migrating data into GPU memory. In order to decide for the data to be transferred they implement the strategies least-frequently used (LFU) and least-recently used (LRU). They state that both strategies perform similar in their evaluation. In MapD [Mo13], a strategy based on LFU using the GPU memory as a fast buffer pool is implemented. In contrast to CoGaDB, they partition columns into chunks, allowing a fine-granular data placement. Furthermore, a database administrator (DBA) is able to influence the data placement by pinning chunks into a layer of the memory hierarchy. In this context, we see potential for improvement with both LRU and LFU. While LRU lacks the capability to account for frequently accessed data partitions, LFU tends to keep certain data partitions longer than needed in GPU memory. We believe that even the expertise of a DBA is insufficient to fully eliminate the shortcomings. Additionally both strategies do not account for performance difference of migrated data.

3 Workload-Driven Data Placement

Using the GPU memory as a hot cache for the DBMS, i.e., with a data-driven execution model [BFT16], requires to place columns into GPU memory. Since the placement happens independent of query execution, a separate data placement job is executed. In order to account for changes in a given workload, this job has to occur in periodic intervals or has to be triggered by events, such as a violation of a service level agreement (SLA). The data placement aims to transfer columns following a defined optimization goal, i.e., the reduction of the overall execution time for a given workload. Deciding on the columns to fill the hot cache requires a strategy, i.e., LRU or LFU. In contrast to these, we propose to derive the decision for data placement from a cost metric based on the profit of placing a particular column in GPU memory. In particular, we describe how this profit is updated during query execution by reusing runtime estimates acquired during operator placement.

In general, we define the profit of placing a column in GPU memory over all previously executed queries, by the sum of the profits of each individual query. The profit for a single query is determined by the difference in runtime of executing the query on the GPU to the runtime of executing the query on the CPU. The runtime for the GPU-based execution includes result data transfer only, as the input data resides in GPU memory. Furthermore, in case of a faster runtime on CPU, we assume a profit of zero. This constraint is based on the assumption that columns in GPU memory present a mirrored subset of all columns present in a higher memory hierarchy, as implemented in MapD [Mo13]. Hence, queries can run on the CPU even if columns required to answer the query are present in GPU memory.

Executing each query on both execution devices in order to obtain the accurate runtimes for the calculation of the proposed cost metric is infeasible in a productive DBMS. Therefore, we utilize estimates of the runtime, which are commonly used in query optimization or operator placement. Since query processing in heterogeneous execution environments requires an operator placement, which is based on runtime estimates for given operators [Ka14], these estimates can be reused for the calculation of the cost metric. The operator placement is either integrated into the query optimizer or executed by a dedicated placement optimizer [KHL17]. Therefore, we extend the according component to update the cost metric for each column during the placement decision. Note, letting the placement component update the cost metric allows to take a more fine-granular unit for calculating the profit into account. Thus, the profit per query is replaced by the profit per operator or sub-operator according to the unit for operator placement.

The data placement job evaluates the above defined cost metric for each column and transfers columns into GPU memory in periodic intervals. At first, a list containing a pointer to the column, its accumulated profit and its memory footprint is created. Next, the list is ordered starting with the column providing the highest profit per memory. Afterwards, the candidates for the hot cache, the GPU memory, are drawn from the list starting at the top until the amount of GPU memory reserved is exhausted. In case the current candidate does not fit into the remaining GPU memory the next fitting one is chosen, until no more columns fit within the given memory budget. Thus, we employ a greedy approach. This could be replaced for example by using linear programming to find an optimal solution. Note, parts of the GPU memory are reserved for intermediate and final results of operators. At last the hot cache is updated according to the list of candidates. This involves checking and evicting the columns resident in GPU memory, as well as, transferring new columns into it.

For the proposed workload-driven data placement it remains open to decide for an appropriate interval to update the data placement or for appropriate events that trigger the data placement. Furthermore, it needs to be evaluated whether the cost metric has to incorporate a temporal factor to account for changes in workloads faster.

4 Next Steps

In our work we propose a novel data placement strategy using a cost metric for a GPU-accelerated DBMS, which uses the GPU memory as a hot cache. In a next step, the workload-driven data placement is implemented in an existing open source GPU-accelerated DBMS. Currently, we investigate CoGaDB [Br14] and MapD [Mo13] for this purpose. Once integrated, our strategy is evaluated against existing approaches based on LFU and LRU. For the measurements we will use workloads, based on the star schema benchmark and a modified TPC-H benchmark, which have been used in [BFT16] to allow for comparability. Furthermore, we aim to focus the evaluation on worst case scenarios for LFU, LRU and our workload-driven data placement in order to investigate the limits of each strategy.

References

[BFT16] Breß, S.; Funke, H.; Teubner, J.: Robust Query Processing in Co-Processor-accelerated Databases. In: Proceedings of the 2016 International Conference on Management of Data. SIGMOD '16, ACM, San Francisco, California, USA, pp. 1891–1906, 2016.

[Br14] Breß, S.: The Design and Implementation of CoGaDB: A Column-oriented GPU-accelerated DBMS. Datenbank-Spektrum 14/3, pp. 199–209, Nov. 2014.

[Fu18] Funke, H.; Breß, S.; Noll, S.; Markl, V.; Teubner, J.: Pipelined Query Processing in Coprocessor Environments. In: Proceedings of the 2018 International Conference on Management of Data. SIGMOD '18, ACM, Houston, TX, USA, pp. 1603–1618, 2018.

[GH11] Gregg, C.; Hazelwood, K.: Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In: ISPASS IEEE. Pp. 134–144, 2011.

[He08] He, B.; Yang, K.; Fang, R.; Lu, M.; Govindaraju, N.; Luo, Q.; Sander, P.: Relational Joins on Graphics Processors. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. SIGMOD '08, ACM, Vancouver, Canada, pp. 511–524, 2008.

[Ka14] Karnagel, T.; Habich, D.; Schlegel, B.; Lehner, W.: Heterogeneity-Aware Operator Placement in Column-Store DBMS. Datenbank-Spektrum 14/3, pp. 211–221, Nov. 2014.

[KHL17] Karnagel, T.; Habich, D.; Lehner, W.: Adaptive Work Placement for Query Processing on Heterogeneous Computing Resources. Proc. VLDB Endow. 10/7, pp. 733–744, Mar. 2017.

[Mo13] Mostak, T.: An Overview of MapD (Massively Parallel Database), 2013, URL: `www.smallake.kr/wp-content/uploads/2014/09/mapd_overview.pdf`.

[YLZ13] Yuan, Y.; Lee, R.; Zhang, X.: The Yin and Yang of Processing Data Warehousing Queries on GPU Devices. Proc. VLDB Endow. 6/10, pp. 817–828, Aug. 2013.