GPU-accelerated join-order optimization

Andreas Meister
Supervisor: Gunter Saake
University of Magdeburg, Germany
andreas.meister@ovgu.de

ABSTRACT

Join-order optimization is an important task during query processing in DBMSs. The execution time of different join orders can vary by several orders of magnitude. Hence, efficient join orders are essential to ensure the efficiency of query processing. Established techniques for join-order optimization pose a challenge for current hardware architectures, because they are mainly sequential algorithms. Current architectures become increasingly heterogeneous by using specialized co-processors such as GPUs. GPUs offer a highly parallel architecture with a higher computational power compared to CPUs. Because join-order optimization benefits from parallel execution, we expect further improvements by using GPUs. Therefore, in this thesis, we adapt join-order optimization approaches to GPUs.

1. INTRODUCTION

When a query contains several joins (e.g., Query 5 of the TPC-H benchmark), the execution time can differ by several orders of magnitude depending on the join order [20], see Figure 1. Therefore, join-order optimization is an essential step within query processing. So far, proposed join-order optimization approaches were almost exclusively sequential. This poses a challenge, because sequential algorithms cannot fully utilize the potential of current hardware architectures. Current architectures make increasingly use of parallelism to satisfy the ever increasing application requirements. Hence, current architectures integrate highly parallel co-processors such as *Graphical Processing Units (GPUs)*. Based on the higher parallelism compared to *Central Processing Units (CPUs)*, GPUs offer an higher computational power [9]. Because join-order optimization can be improved by parallelization [11], we expect further improvements for join-order optimization using GPUs by increasing the evaluated search space or reducing execution times. Furthermore, GPU-accelereated join-order optimization approaches enables us to schedule query processing and optimization tasks to different processors and, hence, the parallel execution of query processing and optimization.

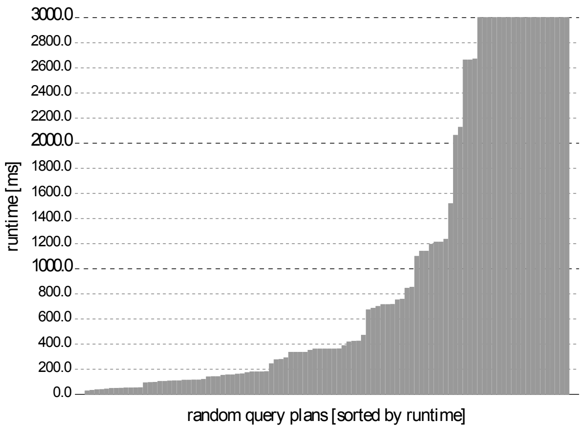

Figure 1: Execution times of different join orders for query 5 of TPC-H taken from [20]

Therefore, in this thesis, we adapt join-order optimization approaches to GPUs to further improve the quality of the results of different approaches for join-order optimization.

The remainder of this paper is structured as follows. In Section 2, we discuss the properties and benefits of GPUs. In Section 3, we present the problem of join-order optimization and existing approaches to solve this problem. In Section 4, we present our research plan and summarize in Section 5.

2. GPU-ACCELERATION

Traditionally, *Database Management Systems (DBMSs)* only use CPUs to optimize and execute queries. However, current system architectures change from single-core CPU systems to multi-core CPU systems supported by multiple dedicated or coupled co-processors [13]. Different co-processors such as GPUs exist providing different features based on their specialized architectures. Because GPUs offer a parallel architecture with one of the highest performance per dollar ratio and C-like programming interfaces [21], GPUs are the most widespread co-processors used to accelerate a variety of approaches.

2.1 Challenges in GPU-acceleration

Although GPUs are used in a variety of diverse applications, GPU-acceleration is not applicable to every approach. Approaches must fully utilize the specialized architecture of

Proceedings of the VLDB 2015 PhD Workshop

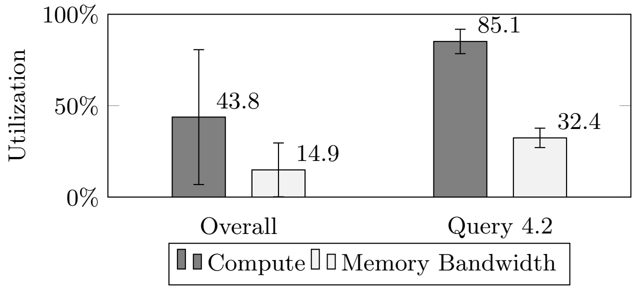

Figure 2: Average utilization of a Tesla K20M running a star schema benchmark (SF=10) on CoGaDB. Overall, less than half of the compute resources are used.

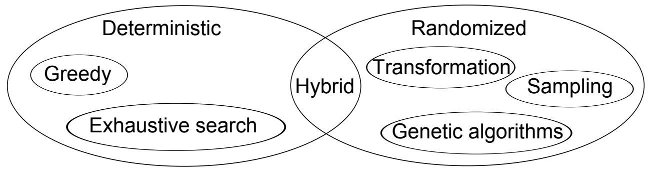

Figure 3: Categories of join-order optimization approaches

GPUs to benefit from GPUs. This poses several challenges for adapting existing approaches.

For example, the parallel execution of tasks is only possible if branching is avoided within the calculations.

Besides the flow of operations, for calculations on GPUs also the data has to be available on the GPU cores. Therefore, data needs to be transferred from the main memory to the GPU device memory. Especially for processing larger sizes of data, this transfer bottleneck must be considered, because GPUs offer only a limited capacity of memory.

Furthermore, we need to use the different memory types of GPUs. Before the data is processed, the data needs to be transferred efficiently from the large but slow device memory to the small and fast on-chip or cacheable memory via coalesced memory access to achieve peak performance.

Whether we can avoid these challenges within an implementation depends on the properties of the approaches.

2.2 Query Processing vs. Optimization

In DBMSs, approaches can be roughly categorized in two different groups: Query processing and optimization. In the past, database researchers mainly used GPU-acceleration for query processing [12].

Query processing is mainly data-bound. Therefore, the transfer bottleneck and small storage space of GPUs limit the applicability of query processing on GPUs. For example, in CoGaDB [6], query processing can only utilize less than 45% of a Tesla K20M in average while processing the star schema benchmark, see Figure 2. In contrast to query processing, optimization in DBMSs is compute and not data-bound, because optimization in DBMSs uses small-sized statistics to solve complex optimization problems. Hence, query processing and optimization can be executed in parallel on GPUs to exploit the unused computational power of GPUs.

Optimization of DBMSs consists of several independent sub tasks. Plenty of these sub tasks, such as selectivity estimation or join-order optimization, benefit from a parallel execution and, hence, would also benefit from GPUs. Unfortunately, we are only aware of GPU-accelerated approaches for selectivity estimation [2, 3, 4, 14]. Although for join-order optimization the benefits of parallelism were already proven [11], the execution on GPUs is still not evaluated.

3. JOIN-ORDER OPTIMIZATION ON GPUS

Given an arbitrary number of relations, which should be joined, the task of join-order optimization is to determine an optimal or efficient order in which the joins of the relations should be performed. Because the execution time of different join-orders can vary by several orders of magnitude [20], a high quality of results of join-order optimization is essential for efficient query processing. Hence, join-order optimization is one of the most critical optimization problems in DBMSs. Join-order optimization is an NP-hard problem [23]. Therefore, the computation of efficient join orders is a challenging task, especially if the problem size increases.

In order to reduce the complexity of join-order optimization, different heuristics to prune the search space were proposed, such as restricting the tree form [1] or postponing Cartesian products [28]. Unfortunately, pruning the search space may lead to non-efficient solutions [25]. Hence, another approach is needed to efficiently determine efficient join-orders: parallelization.

Current architectures offer high computational power by using parallel (co-)processors. Unfortunately, so far, mainly sequential algorithms were proposed, which cannot fully utilize this parallelism. The benefits from parallelization for join-order optimization were already proven for multi-core CPUs [11]. Because current specialized co-processors such as GPUs offer even higher parallelism, we expect further improvements for join-order optimization using GPUs.

The improvement of join-order optimization will depend on the category of the join-order optimization approach. In Figure 3, we show the three categories of join-order optimization approaches: deterministic, randomized, and hybrid approaches.

3.1 Deterministic Approaches

Given the same input, deterministic approaches provide the same result for multiple repetitions. We can further divide deterministic approaches into two categories: greedy and exhaustive search approaches.

Greedy approaches iteratively construct the result by joining at each step one relation to the partial result. The next joinable relation will be selected by using heurisitics, such as join selectivity [27], to minimize the overall costs. Although using heuristics reduce the complexity of join-order optimization, greedy approaches can stuck in local optima. Hence, greedy approaches cannot guarantee an optimal solution [19]. Because greedy approaches only apply simple rules to determine the same result for one specific input, even for multiple repetitions, we do not expect that greedy approaches will exploit the potential of GPUs except for batch-processing multiple queries.

In contrast to greedy approaches, exhaustive search approaches guarantee an optimal solution, but can only be applied on a small problem size.The optimal solution can either be constructed in top-down or bottom-up manner. Bottom-up approaches, such as *Dynamic Programming (DP)* [24], iteratively construct the final result by using partial results. In contrast to bottom-up approaches, top-down approaches can avoid unnecessary operations by using multiple enhancements, such as prioritization and branch-and-bound pruning, which are not applicable for bottom-up approaches [8].

For parallelizing top-down approaches, a limiting factor might be the memoization. If the data structure for memoization is accessed in parallel, these accesses have to be synchronized, if the same item is accessed.

3.2 Randomized Approaches

Greedy approaches cannot guarantee an efficient result and deterministic approaches are limited to a small problem size. Hence, several randomized approaches were proposed to ensure an efficient but not optimal solution for bigger problem sizes. We can categorize randomized approaches into three different groups: transformation-based approaches, sampling, and *Genetic Algorithms (GAs)*.

Transformation-based approaches, such as simulated annealing [15], randomly select transformation rules to create new join trees. Based on the approach, only improvements or also deterioration of the corresponding costs are considered.

In addition to transformation-based approaches, random sampling was proposed [10]. Based on an enumeration of all possible join trees, join trees are uniformly selected, constructed and evaluated.

In contrast to sampling, GAs iteratively improve the quality of results by using the concept of evolution [5]. Starting with a randomized initial population, new individuals are iteratively created by combining features of existing individuals (crossover) or by randomly changing single features of existing individuals (mutation). In order to converge to an efficient solution, at each iteration existing and created individuals are rejected by a selection process.

3.3 Hybrid approaches

Since deterministic and randomized approaches for join-order optimization have both different advantages and disadvantages, hybrid approaches were proposed, such as Iterative Dynamic Programming [17]. Hybrid approaches can adapt their execution based on the complexity of the optimization problem. For simple problems, proposed algorithms work like an exhaustive search, whereas for complex problems, heuristics are used to simplify the optimization process and apply the exhaustive search only for subspaces of the whole search space.

3.4 Benefits of GPUs

In this section, we presented different categories for available join-order optimization approaches. The benefit of GPU-acceleration will depend on the category of the join-order optimization approach. On the one hand, GPU-accelerated deterministic approaches will be applicable to more complex problems by reducing the execution time. The parallel execution of the DP on multi-core CPUs increased the applicability from 12 to 25 joins. By using GPUs, we hope to increase applicability to 50 joins. On the other hand, GPU-accelerated randomized approaches will provide more efficient plans by evaluating an greater search space in the same execution time. For example, the result quality of GAs can be increased by up to 20% [22]. Because exhaustive search approaches benefit from parallelism, hybrid approaches can shift the threshold for using heuristics and, therefore, can provide optimal solutions for more complex problems using GPUs.

4. TOWARDS GPU-ACCELERATED JOIN-ORDER OPTIMIZATION

By exploiting the highly parallel architecture of GPUs, we expect to improve the join-order optimization, see Section 3.

Therefore, in this thesis, we will evaluate the effects of GPUs on different join-order optimization approaches. In order to evaluate the effects of GPUs on join-order optimization, we will implement a standalone query optimization framework similar to Opt++ [16]. In contrast to Opt++, our main goal of the optimization framework is not to provide an efficient extensibility, but a comprehensive comparison of different parallelization strategies for join-order optimization approaches. Therefore, we will implement different versions of multiple join-order optimization approaches within the framework by using different parallelization strategies such as single-threaded, multi-threaded execution on CPUs, and the execution on GPUs. Applying different parallelization strategies for multiple optimization approaches within one framework enables us to easily measure the effects on run time and quality of results of the different parallelization strategies for the join-order optimization approaches. Because single and multi-threaded CPU versions are already available for DP, we will start evaluating DP and continuously extend the framework by integrating further join-order optimization such as GA.

Although join-order optimization is depended on the quality of cost estimations [18], the cost estimation is not the focus of this thesis. In order to provide accurate cost estimations, we can use available GPU-accelerated selectivity estimations [2, 3, 4, 14]. This enables us to evaluate the effects of the quality of cost estimations on GPU-accelerated join-order optimization. In order to exclude the effects of selectivity estimation, we could determine the selectivity of operators beforehand and perform the join-order optimization with the exact operator selectivities.

Because join-order optimization is not an end in itself but performed to accelerate the query processing, we will integrate our planned optimization framework into existing DBMSs. Thus, we will also be able to evaluate the effects of query processing using GPU-accelerated join-order optimization. We will integrate the implemented framework into CoGaDB [6] and Postgres [26] to obtain generally valid results. CoGaDB is a GPU-accelerated, main-memory column-store [6], whereas Postgres is a disk-based row-store.

The integration of our planned join-order optimization framework into CoGaDB is especially interesting, because CoGaDB already uses GPUs for query processing. Therefore, we need to find an efficient way to schedule the optimization and processing tasks in a way so that the optimization does not affect the query processing on GPUs. CoGaDB already offers HyPE, a framework to schedule tasks to different (co-)processors [7]. Furthermore, HyPE manages memory and required data transfer dependent on scheduling decision automatically. By adapting HyPE to schedule optimization as well as query processing tasks, we will be able to efficiently perform query processing and optimization tasks on different (co-)processors in parallel.

In order to evaluate the benefits of GPUs for join-order-optimization, similar to Han et al. [11], we will use an artificial data set using star-shaped queries, typical for OLAP queries. This enable us to compare our results with already published evaluation results.

5. SUMMARY

In this paper, we presented our planed research on GPU-accelerated join-order optimization. We presented a brief overview of existing join-order optimization approaches. Furthermore, we discussed which optimization approaches should benefit from GPUs and what advantages we expect from a GPU-accelerated join-order optimization. In addition, we elaborated a research plan for evaluating the effects of GPU-acceleration on join-order optimization within our planned GPU-accelerated query optimization framework.

6. REFERENCES

[1] R. Ahmed, R. Sen, M. Poess, and S. Chakkappen. Of Snowstorms and Bushy Trees. volume 7, pages 1452–1461. VLDB Endowment, 2014.

[2] D. R. Augustyn and L. Warchal. GPU-Accelerated Query Selectivity Estimation Based on Data Clustering and Monte Carlo Integration Method Developed in CUDA Environment. GID, pages 215–224. Springer, 2013.

[3] D. R. Augustyn and L. Warchal. GPU-Accelerated Method of Query Selectivity Estimation for Non Equi-Join Conditions Based on Discrete Fourier Transform. GID, pages 215–227. Springer, 2015.

[4] D. R. Augustyn and S. Zederowski. Applying CUDA Technology in DCT-Based Method of Query Selectivity Estimation. GID, pages 3–12. Springer, 2012.

[5] K. Bennett, M. C. Ferris, and Y. E. Ioannidis. A Genetic Algorithm for Database Query Optimization. ICGA, pages 400–407. Morgan Kaufmann Publishers, 1991.

[6] S. Breß. The Design and Implementation of CoGaDB: A Column-oriented GPU-accelerated DBMS. *Datenbank-Spektrum*, 14(3):199–209, 2014.

[7] S. Breß, N. Siegmund, M. Heimel, M. Saecker, T. Lauer, L. Bellatreche, and G. Saake. Load-Aware Inter-Co-Processor Parallelism in Database Query Processing. *Data & Knowledge Engineering*, 2014.

[8] D. DeHaan and F. W. Tompa. Optimal Top-down Join Enumeration. SIGMOD, pages 785–796. ACM, 2007.

[9] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover. GPU Cluster for High Performance Computing. SC, pages 47–58. IEEE, 2004.

[10] C. A. Galindo-Legaria, A. Pellenkoft, and M. L. Kersten. Fast, Randomized Join-Order Selection - Why Use Transformations? VLDB, pages 85–95. Morgan Kaufmann Publishers, 1994.

[11] W.-S. Han, W. Kwak, J. Lee, G. M. Lohman, and V. Markl. Parallelizing Query Optimization. *PVLDB*, 1(1):188–200, 2008.

[12] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, and P. V. Sander. Relational Query Coprocessing on Graphics Processors. *TODS*, 34:21:1–21:39, 2009.

[13] J. He, S. Zhang, and B. He. In-cache Query Co-processing on Coupled CPU-GPU Architectures. *PVLDB*, 8(4):329–340, 2014.

[14] M. Heimel, M. Kiefer, and V. Markl. Self-Tuning, GPU-Accelerated Kernel Density Models for Multidimensional Selectivity Estimation. SIGMOD, pages 1477–1492. ACM, 2015.

[15] Y. E. Ioannidis and E. Wong. Query Optimization by Simulated Annealing. SIGMOD, pages 9–22. ACM, 1987.

[16] N. Kabra and D. J. DeWitt. OPT++ : An Object-oriented Implementation for Extensible Database Query Optimization. *VLDB Journal*, 8(1):55–78, Apr. 1999.

[17] D. Kossmann and K. Stocker. Iterative Dynamic Programming: A New Class of Query Optimization Algorithms. *TODS*, 25(1):43–82, 2000.

[18] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdzic. Robust Query Processing Through Progressive Optimization. SIGMOD, pages 659–670. ACM, 2004.

[19] V. Muntes-Mulero, C. Zuzarte, and V. Markl. An inside analysis of a genetic-programming based optimizer. IDEAS, pages 249–255, 2006.

[20] T. Neumann. Engineering High-Performance Database Engines. *PVLDB*, 7(13):1734–1741, 2014.

[21] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[22] P. Pospichal, J. Jaros, and J. Schwarz. Parallel Genetic Algorithm on the CUDA Architecture. EvoApplicatons, pages 442–451. Springer, 2010.

[23] W. Scheufele and G. Moerkotte. On the Complexity of Generating Optimal Plans with Cross Products (Extended Abstract). PODS, pages 238–248. ACM, 1997.

[24] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access Path Selection in a Relational Database Management System. SIGMOD, pages 23–34. ACM, 1979.

[25] M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and Randomized Optimization for the Join Ordering Problem. *VLDB Journal*, 6(3):191–208, Aug. 1997.

[26] M. Stonebraker and G. Kemnitz. The POSTGRES Next Generation Database Management System. *CACM*, 34(10), Oct. 1991.

[27] A. Swami. Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques. SIGMOD, pages 367–376. ACM, 1989.

[28] A. Swami and A. Gupta. Optimization of Large Join Queries. SIGMOD, pages 8–17. ACM, 1988.