Why it is time for a HyPE: A Hybrid Query Processing Engine for Efficient GPU Coprocessing in DBMS

Sebastian Breß
supervised by Gunter Saake
University of Magdeburg
sebastian.bress@ovgu.de

ABSTRACT

GPU acceleration is a promising approach to speed up query processing of database systems by using low cost graphic processors as coprocessors. Two major trends have emerged in this area: (1) The development of frameworks for scheduling tasks in heterogeneous CPU/GPU platforms, which is mainly in the context of coprocessing for applications and does not consider specifics of database-query processing and optimization. (2) The acceleration of database operations using efficient GPU algorithms, which typically cannot be applied easily on other database systems, because of their analytical–algorithm-specific cost models. One major challenge is *how* to combine traditional database query processing with GPU coprocessing techniques and efficient database operation scheduling in a GPU-aware query optimizer. In this thesis, we develop a hybrid query processing engine, which extends the traditional physical optimization process to generate hybrid query plans and to perform a cost-based optimization in a way that the advantages of CPUs and GPUs are combined. Furthermore, we aim at a portable solution between different GPU-accelerated database management systems to maximize applicability. Preliminary results indicate great potential.

1. INTRODUCTION

Since the invention of relational *database management systems* (DBMSs) 40 years ago, performance demands of applications have been ever increasing with no border in sight. A new emerging trend applies *General Purpose Graphical Processing* (GPGPU) on database systems to create GPU-accelerated database management systems. In fact, a plenitude of research has investigated the acceleration of database systems using GPUs: relational operators [3, 10, 18], index-scan acceleration [4, 28], kNN-search [12, 39], online aggregation [29, 30, 38], sorting [14], spatial range queries [35], XML path filtering [31], duplicate detection [11], and MapReduce [17]. Due to the properties of GPUs, a GPU algorithm is not necessarily faster than its CPU counterpart, because data has to be copied from the CPU's main-memory to the GPU's device-memory to be processable by the GPU. Hence, these copy operations introduce a significant overhead and may lead to a slow down in system performance [16]. Therefore, it is important to use the GPU only, when it is beneficial for query processing. To this end, the query optimizer has to be aware of the properties of different processing devices, such as CPU and GPU, and has to make suitable scheduling decisions to maximize system performance [7].

Proceedings of the VLDB Endowment, Vol. 6, No. 12

1.1 Problem Statement

The *research challenge* is, how to combine scheduling decisions with the physical optimization process in database optimizers to maximize performance. Since applications have different requirements, it should be possible to support response-time or throughput optimization, which in turn, may require completely different scheduling heuristics. Therefore, the engine has to carefully evaluate a logical query plan and create a suitable *hybrid query plan*; a physical query plan utilizing possibly multiple processing devices. We identify hybrid query processing and optimization as a challenging task in prior work [7]: The physical optimization process becomes more complex, because the optimization space increases with the possibility of using the CPU or the GPU for executing an operator. Additionally, changing the processing device may require data transfers. Gregg and Hazelwood investigated the performance impact of data transfers and concluded that the storage location has a significant impact on performance, because data transfers pose a major bottleneck [16].

1.2 Research Goals

The goal of our research is to develop a *hybrid query processing engine* (HyPE) that handles creation and optimization of hybrid query plans w.r.t. an optimization criterion.[1] We aim at a portable solution, so HyPE can be easily applied to different GPU-accelerated DBMSs, such as GDB [18], Ocelot [21], or Virginian [2]. To achieve our goals, we need to

- review the literature for suitable heterogeneous scheduling approaches and GPU-accelerated data management to be able to build on state-of-the-art results.
- create a flexible architecture for HyPE and apply it at several hybrid DBMSs to prove applicability.

[1] `http://wwwiti.cs.uni-magdeburg.de/iti_db/research/gpu/hype/`

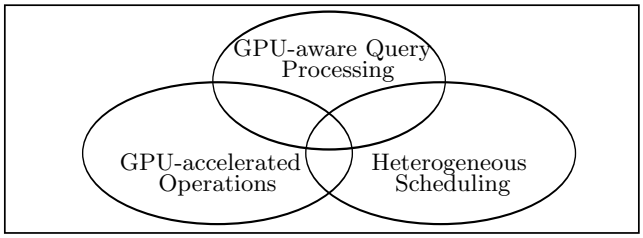

Figure 1: Relationship of research fields

- implement and evaluate different query processing techniques for different optimization criteria.

1.3 Contribution and Paper Structure

In this paper, we contribute: (1) A literature review on database query processing on GPUs and heterogeneous scheduling approaches. (2) Our design of HyPE, as well as (3) a detailed research plan to achieve our goals.

The remaining article is organized as follows. In Section 2, we discuss preliminary considerations and summarize our preliminary results for our PhD project in Section 3. We present our design of HyPE in Section 4. Finally, we propose our research plan in Section 5.

2. PRELIMINARY CONSIDERATIONS

GPU-accelerated data management can highly benefit from heterogeneous scheduling approaches, whereas GPU-accelerated data management may allow custom optimizations. Therefore, in this section, we briefly describe the basics of GPUs and explore the literature for GPU-accelerated query processing and heterogeneous scheduling. Figure 1 illustrates the relationship between the research areas.

2.1 Graphics Processing Units

GPUs are specialized processing devices with the following properties: (1) They have a significant higher theoretical computational power than CPUs for the same monetary costs. (2) GPUs have a highly parallel architecture based on threads, designed for scenarios where thousands to ten thousands of threads run in parallel to solve a problem. (3) They are optimized for numeric computation. Therefore, a lot of control flow in a GPU algorithm brakes the performance, because several computing units share one control unit in a symmetric multiprocessor. (4) Data can only be processed, if it is dormant in the GPU RAM. Hence, we have to transfer data between CPU RAM and GPU RAM when the input data is not present in the memory of the current processing device. However, these data transfers slow down the system, because the PCIe Bus is significantly slower than the memory bandwidth of the CPU or GPU RAM [37].

2.2 Literature Review

We now discuss related work in the field of GPU-accelerated data management as well as heterogeneous scheduling and hybrid query processing.

2.2.1 GPU-accelerated Data Management

In 2004, Govindaraju and others were the first to investigate the GPU as coprocessor for database query processing [15]. They accelerated database operators such as selections and aggregations. They represented data as textures in GPU RAM and used fragment programs on those textures to perform computations. With the capability of GPUs to perform general purpose computations using CUDA or other frameworks, new approaches were developed. He and others investigated efficient algorithms for join processing in GPUs [19], as well as for the remaining relational operators [18] and implemented them in their prototype *GDB*. Furthermore, they proposed an analytical cost model for their relational operators and developed a coprocessing technique that considers the resources of CPU and GPU as well as the cost of data transfers to utilize suitable processors for an operator [18]. He and others also investigated throughput optimization of transactions using GPUs in their prototype *GPUTx*. They group multiple transactions into bulks and execute them concurrently on the GPU while maintaining conflict serializability. They observe an improvement of throughput by 4-10 times compared to a CPU-based counterpart [20]. Diamos and others designed and implemented GPU algorithms for relational operators as well [10]. The authors state that their algorithms are 1.69 to 3.54 times faster than the algorithms of He and others [18]. Bakkum and others extend the SQLite command processor by GPU algorithms with focus on accelerating a subset of possible SQL queries [3]. Furthermore, they investigated efficient data management on GPUs in their prototype *Virginian* and propose the data structure *tablet*, which utilizes a major column format of data, supporting coalescing memory access on GPU and cache awareness on CPU side, respectively [2]. The approach does not assume that input or output results fit completely in GPU memory, supporting arbitrarily large databases. Virginian represents query plans as a sequence of opcodes, which are executed by a virtual machine. This avoids that data has to be read and written from global device memory and can be kept in local and shared memory, when further operations are performed on the GPU, leading to significant performance improvements.

Pirk and others studied the acceleration of foreign key joins. Their approach streams the keys to the GPU RAM and performs the random lookups on the GPU [34]. Pirk and others argue that operation-based scheduling may overutilize a processing device, while keeping the other devices idle. They propose the coprocessing technique *bitwise distribution*, where a compressed approximating version of the data is stored in the GPU's RAM while the CPU RAM contains the complete data. The query processing has two steps: (1) In the *GPU preselection phase*, the data is prefiltered on a best effort base, keeping some false positives. After transferring the result in the CPU RAM, the CPU constructs the final tuple values and removes the false positives in the *refinement phase* to obtain the final result [33, 35]. Ghodsnia proposed a new generation of in-memory databases: a column store kept entirely in GPU RAM to minimize the data transfer overheads [13]. He argues that GPU RAM size increases with each generation and large data sets can be distributed on multiple GPUs. However, he did not yet provide a qualitative evaluation of his concepts. Kaczmarski compared OLAP cube creation on CPUs and GPUs and identified advantages and challenges of GPU coprocessing in databases [26]. Riha and others proposed an architecture for adaptive hybrid OLAP systems [36]. The idea is to utilize CPUs and GPUs and exploiting their specific properties to reduce system response time. They use data formats optimized for the characteristics of the respective processing

devices and consider data locality information. Riha and others found that (1) their approach achieves the fourfold performance compared to the best related solutions and (2) a GPU increases the systems speed by a factor of two for 10% higher monetary costs [36]. Zhong and He proposed to use the GPU as additional in-memory buffer to avoid costly I/O operations. They applied their solution on the virtualization platform Xen and observed a reduction of average I/O cost by up to 68%.

2.2.2 *Heterogeneous Scheduling and Hybrid Query Processing*

In contrast to the following approaches, HyPE is a library designed for effective database query processing and optimization. The design goal is that HyPE handles physical cost-based query optimization, a property state of the art approaches lack. Query processing techniques for distributed database systems do not consider special properties of a hybrid CPU/GPU system (e.g., bus topology, highly differing processing devices, or a tight coupling of the processing devices).

Therefore, we argue that concepts for distributed query processing are not applicable to hybrid CPU/GPU systems without modifications. In contrast to parallel database systems, hybrid CPU/GPU systems possess heterogeneous processing devices. In hybrid CPU/GPU systems, it is more difficult to decide, which processor should be allocated for which operation, because of the heterogeneous nature of such systems. Consequently, research identified the need for heterogeneous scheduling frameworks, which allocate for each task the most suitable processor [1, 24].

Malik and others introduce a scheduling approach for hybrid OLAP systems [30]. They use a tailor-made calibration-based analytical cost model to estimate execution times of CPU and GPU algorithms. Since the cost model is operation and system specific, it is not easily portable to other database operations and DBMS. Kerr and others propose a model for static selection of CPU and GPU algorithms [27]. Therefore, the approach does not introduce run-time overhead, but does not allow inter-device parallelism between operations of the same type (e.g., sort operations), and cannot exploit possible break even points of CPU and GPU algorithms to increase system performance compared to dynamic approaches (e.g., [5, 25]). Iverson and others [25] introduce an approach based on an estimation of task execution times using a learning-based approach. Similar to our model [5], their solution requires no hardware specific information. Augonnet and others develop StarPU, a run-time system with unified execution environment that is able to distribute parallel tasks in systems with heterogeneous processors [1]. Experiments with StarPU showed super-linear efficiency, meaning computational power the hybrid system is higher than the sum of each heterogeneous processors' computational power. Similar to our approach, StarPU is capable of creating performance models for tasks automatically as well. Ilić and others introduce CHPS, an execution environment for platforms including heterogeneous processors such as hybrid CPU/GPU systems [24]. CHPS supports a flexible task description mechanism and overlapping of processor computation and data transfers. Furthermore, it can automatically construct performance models for tasks, similar to StarPU or our approach. Ilić and others applied CHPS on query Q3 and Q6 of the TPC-H benchmark and observed significant performance improvements [23]. However, the queries where implemented using tailor-made optimizations for each query and were integrated into CHPS. We target a portable solution, where a query plan is automatically optimized to run in a heterogeneous environment. Furthermore, our decision can be easily integrated in an existing DBMS by design.

3. PRELIMINARY RESULTS

Most heterogeneous scheduling approaches work as extensible frameworks for task-based scheduling. But in a DBMS, the heterogeneous scheduling approach has to be integrated in the query optimizer, which may pose a severe constraint.

Therefore, we developed a heterogeneous scheduling approach, which distributes database operations on CPU and GPU processing devices with minimal response time [8]. Cost models are the base of such decisions, but estimating the cost of an operation for heterogeneous processors is a complex task. Mostly, an analytical cost model is built for the processing device and has to be calibrated according to the specific hardware in a system [18, 30]. Some approaches use learning-based strategies, which map feature values (e.g., input data size, selectivity, data skew) to execution times [5, 25]. According to Wu et al., well calibrated cost models with sufficiently accurate cardinality estimation are either equally good or better than approaches using machine learning [40]. However, the performance estimation with analytical models has the drawback that the models have to be refined with each new hardware generation and their parameters have to be maintained and properly adjusted by a database administrator. Therefore, we use a learning-based approach to estimate the execution times of operations on different processing devices [5]. In this section, we discuss our preliminary results: our decision model, the challenges for hybrid query processing and our approach for hybrid query optimization.

3.1 Decision Model

We developed a self-tuning decision model that distributes database operations on processing devices w.r.t. an optimization criterion [8]. A set of algorithms can execute an operation O. The idea is to learn an algorithms execution behavior depending on the features of input datasets. In other words, the model observes the correlation between input datasets and resulting execution times and learns the correlation using a statistical method. This statistical method computes an approximation function, when it has collected enough observations.

The model selects the optimal algorithm for an operation by (1) computing estimated execution times for all available algorithms capable of executing O and (2) using the estimated execution times and an optimization heuristic to select the optimal algorithm. For response-time optimization, the system selects the algorithm with lowest estimated execution time.

We investigated suitable use cases of operator-based offloading to coprocessors in prior work and evaluated the model on three use cases: update merging in in-memory column stores, lookup operations on index structures, and sort operations. Depending on use case and use case specific parameters, we observed improvements in overall execution time up to 30% [6].

3.2 Challenges for Hybrid Query Processing

As the second step, we developed a query-optimization approach using our decision model. Hybrid query processing and optimization introduces new challenges that a query optimizer has to consider to achieve maximal performance.

We identified five challenges [7]: (1) The *Pipelining Challenge* states that regular pipelining is not possible between two GPU algorithms, because inter-kernel communication is undefined [32]. However, it is possible to compile several operations in one kernel using OpenCL [21]. (2) The *Execution-Time-Prediction Challenge* declares that estimating execution times becomes problematic when several GPU kernels are executed in parallel, because the impact of multi-kernel execution on a kernels performance is difficult to predict. (3) The *Copy-Serialization Challenge* states that data needed by different operations cannot be transferred concurrently between CPU and GPU, because concurrent copy operations are not allowed [32]. (4) The *Critical-Query Challenge* declares that an optimizer needs a heuristic that decides which queries should be accelerated by the GPU. Not every query can benefit from GPU acceleration, because it depends on the operation, the number of operations concurrently executable on the GPU and the PCIe bandwidth. (5) The *Optimization-Impact Challenge* states the difficulty to estimate the impact of an optimization of one query Q_1 on another query Q_2 or even the overall system performance.

3.3 Hybrid Query Optimization using Decision Model

We proposed a simple algorithm for physical optimization of query sequences [9] and query trees [7]. We considered two basic approaches: (1) The greedy algorithm utilizes the operator-based decision model to select the optimal algorithm for each operation in a logical query plan. Afterwards, the algorithm inserts copy operations whenever the model decides to change the processing device. (2) The n-copy operation heuristic, which is a cost-based algorithm. It generates a set of candidate plans and uses the estimated execution times of our decision model to compute the cost of the hybrid query plans. The algorithm chooses the query plan with lowest cost for execution. Based on the observations of Gregg and Hazelwood [16], the heuristic only considers query plans with less than n-copy operations. This substantially reduces the optimization space, but the heuristic cannot find the optimal query plan when it has more than n copy operations [7].

4. THE VISION: A QUERY PROCESSING ARCHITECTURE

Query processing and optimization is highly specific to the architecture of a database management system. Hence, when we investigate a query processing engine for hybrid query processing, we have to investigate two things: (1) a GPU-aware database architecture and (2) a hybrid query processing engine.

The hybrid query processing engine should be independent of database architecture and database model *independent* to ensure generality, because it handles common concepts such as considering the data location, utilizing intermediate result size and selectivity estimation to choose a suitable query plan.

However, the estimation of intermediate result size or selectivity estimation *is* dependent on database architecture and database model. To close the gap of these contradicting limitations, we split the hybrid query processing engine in two layers: (1) a database system independent layer, which implements query optimization strategies, cost estimation and query plan selection and (2) a database system dependent layer, which basically maps the first layer to a concrete database management system. Since we need a hybrid DBMS as back-end, we are currently working on a GPU-aware database architecture, which is, in essence, a column-oriented-GPU-accelerated DBMS (CoGaDB).

4.1 HyPE

The database architecture independent layer of our hybrid query processing engine has the following tasks: (1) cost estimation of operators in consideration of data locality, (2) implementing hybrid query optimization strategies, and (3) selecting a query plan w.r.t. an optimization criterion.

HyPE consists of three components, which are the *self-tuning execution time estimator* (STEMOD), the *algorithm selector* (ALRIC) and the *hybrid query optimizer* (HOPE [Hybrid OPtimizEr]). STEMOD is responsible to provide accurate, reliable, database architecture and model independent *estimated execution times* for database algorithms. For an incoming algorithm A and a dataset D, STEMOD computes an estimated execution time $T_{est}(D, A)$. ALRIC uses STEMOD to obtain estimated execution times for all available algorithms to execute an operation. For an operator $Op(D, O)$, composed of a dataset D and an operation O, ALRIC looks up all available algorithms for O and utilizes STEMOD to compute estimated execution times ($T_{est}(D, A)$) for them and then decides on the optimal algorithm A_{opt}. HOPE utilizes STEMOD and/or ALRIC to construct a promising physical query plan Q_{phy} for a given logical query plan Q_{log}. Depending on the optimization algorithm, HOPE generates query plans and uses STEMOD to compute their costs or it directly relies on the decisions provided by ALRIC. Figure 2 summarizes the architecture of our hybrid query processing engine. The system dependent and independent layers are highlighted in different gray shades. CoGaDB can utilize all components of HyPE separately. Therefore, it can query STEMOD to obtain an estimated execution time for an algorithm, let ALRIC decide on an algorithm or utilizes HOPE to take care of the physical optimization. This loose coupling between the components allows an existing system to choose, which functionality it likes to use. A system with a well working optimizer might only be interested in estimated execution times, whereas a system without a physical optimizer such as CoGaDB might rely completely on HyPE.

4.2 CoGaDB

CoGaDB[2] is a column-oriented-GPU-accelerated DBMS with the purpose of investigating new possible database architectures and coprocessing schemes. It utilizes the capabilities of HOPE to construct for each logical query plan an optimized hybrid query plan. Furthermore, it implements a coprocessing scheme, where columns belonging to the working set are cached in the GPU RAM. Intermediate results

[2] `http://wwwiti.cs.uni-magdeburg.de/iti_db/research/gpu/cogadb`

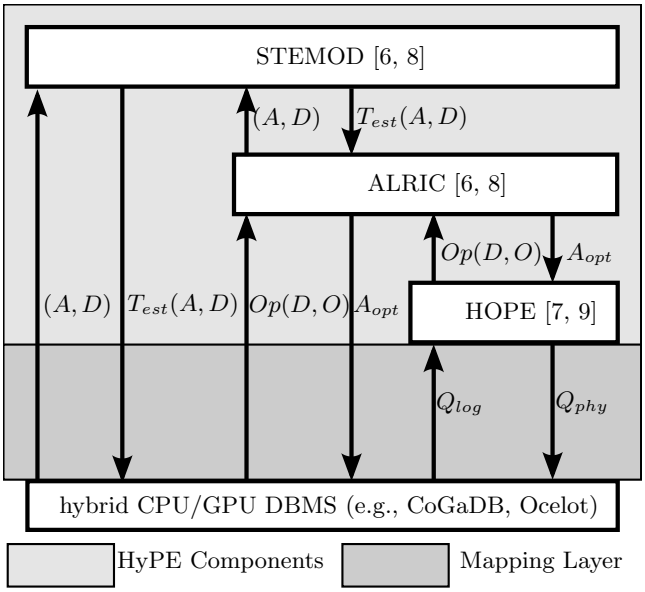

Figure 2: Architecture of HyPE

are represented as list of tuple identifiers (TIDs). If a column is cached in GPU memory, then only the tid list representing the intermediate result has to be transferred to the GPU RAM. Then, the intermediate result is constructed by applying the TID list to the cached column. Finally, the data is processed by the operation, which returns a TID list representing the result. Again, only the TID list has to be transfered between the CPU RAM and the GPU RAM. This minimizes the performance penalty of the data transfers, which were identified as major bottleneck [16]. Furthermore, data is often encoded differently in CPU RAM and GPU RAM. By transferring only the TIDs, no data transcoding has to be done, as long as the relative positions of values in a column is not changed by the coding technique.

5. RESEARCH PLAN

According to the challenges for hybrid query processing described in Section 3.2 and our discussions, we propose the following research plan:

- We will develop alternative optimization heuristics for ALRIC to allow a fine-granular customization of HyPE's behavior.
- We will implement and evaluate different query evaluation approaches, such as streaming-based operator processing, single-query-plan optimization, and global-query-graph optimization:
  streaming-based operator processing: n query plans are serialized in an operator sequence and the streams are merged to one stream, which is scheduled by ALRIC.
  single-query-plan optimization: Query plans are optimized separately from each other by HOPE.
  global-query-graph optimization: Query plans are merged into one global query graph and HOPE performs a global optimization.
- For the single-query-plan and global-query-graph optimization, we will investigate and evaluate different query optimization algorithms for HOPE using greedy and cost-based approaches, such as [7].
- We will apply HyPE on at least two GPU-accelerated database management systems. We are currently working on integrating HyPE in CoGaDB and Ocelot [22].
- The aforementioned concepts should be combined and evaluated to identify the pitfalls for designing, implementing and using a hybrid database management system as well as proving their applicability.

6. CONCLUSION

Effective GPU coprocessing in database systems is still an open challenge yet to overcome. In this paper, we identify the need for a hybrid query processing engine, which supports a broad range of database management systems regardless of their architecture, data model, specifics of their algorithms or the hardware in the deployment environment. We summarize the goals, preliminary results and future work of our PhD project. We expect to accelerate database operations further using (1) advanced optimization criteria for operator stream scheduling, (2) cost-based query optimization based on our decision model, and (3) a combination of HyPE and several GPU-accelerated DBMS as a symbiosis of an advanced database architecture and a learning-based-hybrid query optimizer.

Acknowledgements

We thank Norbert Siegmund and David Broneske from University of Magdeburg as well as Tobias Lauer from Jedox AG and Ladjel Bellatreche from University of Poitiers for helpful feedback and discussions.

7. REFERENCES

[1] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice & Experience*, 23(2):187–198, 2011.

[2] P. Bakkum and S. Chakradhar. Efficient Data Management for GPU Databases. 2012. `http://pbbakkum.com/virginian/paper.pdf`.

[3] P. Bakkum and K. Skadron. Accelerating SQL Database Operations on a GPU with CUDA. In *GPGPU*, pages 94–103. ACM, 2010.

[4] F. Beier, T. Kilias, and K.-U. Sattler. GiST Scan Acceleration using Coprocessors. In *DaMoN*, pages 63–69. ACM, 2012.

[5] S. Breß, F. Beier, H. Rauhe, K.-U. Sattler, E. Schallehn, and G. Saake. Efficient Co-Processor Utilization in Database Query Processing. *Information Systems*, 38(8):1084–1096, 2013.

[6] S. Breß, F. Beier, H. Rauhe, E. Schallehn, K.-U. Sattler, and G. Saake. Automatic Selection of Processing Units for Coprocessing in Databases. In *ADBIS*, pages 57–70. Springer, 2012.

[7] S. Breß, I. Geist, E. Schallehn, M. Mory, and G. Saake. A Framework for Cost based Optimization of Hybrid CPU/GPU Query Plans in Database Systems. *Control and Cybernetics*, 41(4):715–742, 2012.

[8] S. Breß, S. Mohammad, and E. Schallehn. Self-Tuning Distribution of DB-Operations on Hybrid CPU/GPU Platforms. In *GvD*, pages 89–94. CEUR-WS, 2012.

[9] S. Breß, E. Schallehn, and I. Geist. Towards Optimization of Hybrid CPU/GPU Query Plans in Database Systems. In *GID*, pages 27–35. Springer, 2012.

[10] G. Diamos, H. Wu, A. Lele, J. Wang, and S. Yalamanchili. Efficient Relational Algebra Algorithms and Data Structures for GPU. Technical report, Center for Experimental Research in Computer Systems (CERS), 2012.

[11] B. Forchhammer, T. Papenbrock, T. Stening, S. Viehmeier, U. Draisbach, and F. Naumann. Duplicate Detection on GPUs. In *BTW*, pages 165–184. Köllen-Verlag, 2013.

[12] V. Garcia, E. Debreuve, and M. Barlaud. Fast k Nearest Neighbor Search using GPU. In *CVPRW*, pages 1–6. IEEE, 2008.

[13] P. Ghodsnia. An In-GPU-Memory Column-Oriented Database for Processing Analytical Workloads. In *The VLDB PhD Workshop*. VLDB Endowment, 2012.

[14] N. Govindaraju, J. Gray, R. Kumar, and D. Manocha. GPUTeraSort: High Performance Graphics Coprocessor Sorting for Large Database Management. In *SIGMOD*, pages 325–336. ACM, 2006.

[15] N. K. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha. Fast Computation of Database Operations using Graphics Processors. In *SIGMOD*, pages 215–226. ACM, 2004.

[16] C. Gregg and K. Hazelwood. Where is the Data? Why You Cannot Debate CPU vs. GPU Performance Without the Answer. In *ISPASS*, pages 134–144. IEEE, 2011.

[17] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang. Mars: A MapReduce Framework on Graphics Processors. In *PACT*, pages 260–269. ACM, 2008.

[18] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, and P. V. Sander. Relational Query Co-Processing on Graphics Processors. In *ACM Trans. Database Syst.*, volume 34. pp. 21:1–21:39. ACM, 2009.

[19] B. He, K. Yang, R. Fang, M. Lu, N. Govindaraju, Q. Luo, and P. Sander. Relational joins on graphics processors. In *SIGMOD*, pages 511–524. ACM, 2008.

[20] B. He and J. X. Yu. High-Throughput Transaction Executions on Graphics Processors. *PVLDB*, 4(5):314–325, 2011.

[21] M. Heimel. Investigating Query Optimization for a GPU-accelerated Database. Master's thesis, Technische Universität Berlin, Electrical Engineering and Computer Science, Department of Software Engineering and Theoretical Computer Science, 2011.

[22] M. Heimel, M. Saecker, H. Pirk, S. Manegold, and V. Markl. Hardware-oblivious parallelism for in-memory column-stores. In *VLDB*. VLDB Endowment, 2013.

[23] A. Ilić, F. Pratas, P. Trancoso, and L. Sousa. *High Performance Scientific Computing with Special Emphasis on Current Capabilities and Future Perspectives*, chapter High-Performance Computing on Heterogeneous Systems: Database Queries on CPU and GPU, pages 202–222. IOS Press, 2011.

[24] A. Ilić and L. Sousa. CHPS: An Environment for Collaborative Execution on Heterogeneous Desktop Systems. *International Journal of Networking and Computing*, 1(1):96–113, 2011.

[25] M. Iverson, F. Ozguner, and L. Potter. Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment. In *HCW*, pages 99–111, 1999.

[26] K. Kaczmarski. Comparing GPU and CPU in OLAP Cubes Creation. In *SOFSEM*, pages 308–319. Springer, 2011.

[27] A. Kerr, G. Diamos, and S. Yalamanchili. Modeling GPU-CPU Workloads and Systems. GPGPU, pages 31–42. ACM, 2010.

[28] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt, and P. Dubey. FAST: Fast Architecture Sensitive Tree Search on Modern CPUs and GPUs. In *SIGMOD*, pages 339–350. ACM, 2010.

[29] T. Lauer, A. Datta, Z. Khadikov, and C. Anselm. Exploring Graphics Processing Units as Parallel Coprocessors for Online Aggregation. In *DOLAP*, pages 77–84. ACM, 2010.

[30] M. Malik, L. Riha, C. Shea, and T. El-Ghazawi. Task Scheduling for GPU Accelerated Hybrid OLAP Systems with Multi-core Support and Text-to-Integer Translation. In *26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pages 1987–1996. IEEE, 2012.

[31] R. Moussalli, R. Halstead, M. Salloum, W. Najjar, and V. J. Tsotras. Efficient XML Path Filtering Using GPUs. In *ADMS*, 2011.

[32] NVIDIA. NVIDIA CUDA C Programming Guide. `http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf`, 2013. pp. 30–34, Version 5.0, [Online; accessed 16-Feb-2013].

[33] H. Pirk. Efficient Cross-Device Query Processing. In *The VLDB PhD Workshop*. VLDB Endowment, 2012.

[34] H. Pirk, S. Manegold, and M. Kersten. Accelerating foreign-key joins using asymmetric memory channels. pages 585–597, 2011.

[35] H. Pirk, T. Sellam, S. Manegold, and M. Kersten. X-Device Query Processing by Bitwise Distribution. In *DaMoN*, pages 48–54. ACM, 2012.

[36] L. Riha, M. Malik, and T. El-Ghazawi. An Adaptive Hybrid OLAP Architecture with optimized Memory Access Patterns. In *Cluster Computing*, pages 1–15. Springer, 2012.

[37] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. pages 2–6, 186, Addison-Wesley Professional, 1st edition, 2010.

[38] G. Wang and G. Zhou. GPU-Based Aggregation of On-Line Analytical Processing. In *ICCIP*, pages 234–245. Springer, 2012.

[39] W. Wang and L. Cao. Parallel k-Nearest Neighbor Search on Graphics Hardware. In *PAAP*, pages 291–294. IEEE, 2010.

[40] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacıgümüş, and J. F. Naughton. Predicting Query Execution Time: Are Optimizer Cost Models Really Unusable? In *ICDE*. IEEE, 2013.