Adaptive Data Processing in Heterogeneous Hardware Systems

Bala Gurumurthy
Otto-von-Guericke-Universität
bala.gurumurthy@ovgu.de

Tobias Drewes
Otto-von-Guericke-Universität
tobias.drewes@ovgu.de

David Broneske
Otto-von-Guericke-Universität
david.broneske@ovgu.de

Gunter Saake
Otto-von-Guericke-Universität
gunter.saake@ovgu.de

Thilo Pionteck
Otto-von-Guericke-Universität
thilo.pionteck@ovgu.de

ABSTRACT

In recent years, Database Management System have seen advancements in two major sectors, namely functional and hardware support. Before, a traditional DBMS was sufficient for performing a given operation, whereas a current DBMS is required to perform complex analytical tasks like graph analysis or OLAP. These operations require additional functions to be added to the system for processing. Also, a similar evolution is seen in the underlying hardware. This advancement in both functional and hardware domain of DBMS requires modification of its overall architecture. Hence, it is evident that an adaptable DBMS is necessary for supporting this highly volatile environment. In this work, we list the challenges present for an adaptable DBMS and propose a conceptual model for such a system that provides interfaces to easily adapt to the software and hardware changes.

Keywords

Heterogeneous database system, Device parallelism, Query planning

1. INTRODUCTION

In recent years, a traditional Database Management System (DBMS) is required to also perform various complex operations that are not directly supported by it. To perform these special analytical operations, several tailor-made functions are integrated into the DBMS [12]. Further, the efficiency of a DBMS depends on the throughput of the underlying hardware. DBMS operations are ported to different specialized hardware to achieve better throughput. This heterogeneity of functionalities and hardware systems require modifications of the existing DBMS structure incurring additional complexities and challenges.

In the current context, various analytical tasks such as graph processing, or data mining are executed in DBMSs [8, 9]. These functionalities are ported to a DBMS with an overhead of altering the overall architecture of the system. However, modification of the complete system structure is time consuming and also not all components are tuned for efficiency [12].

Further, the throughput of any software system depends on its underlying hardware. Many specialized compute devices are fabricated to perform certain specific tasks efficiently. These devices trade-off higher efficiency in certain tasks for generality. They are used as additional co-processors along with the CPU for better throughput. One of the commonly used co-processor is a GPU, which is mainly used for enhancing graphical processing in a system. The parallelism in GPU has been already exploited extensively for several DBMS operations [3, 1]. Similarly, other devices are available such as MIC (Many Integrated Cores), APU (Accelerated Processing Unit), FPGA (Field Programmable Gate Array), etc., that could be exploited for efficiently executing DBMS operations. The major challenge in integrating this hardware, is the execution of the device specific variant of the same DBMS operation optimized for the given hardware.

Thus, the availability of different hardware enables a new level of parallelism that we call cross-device parallelism. In this level, a user given query is executed in parallel among different devices for concurrent execution. Along with cross-device parallelism, we also have the traditional pipeline and data parallel execution of functions to increase the efficiency. These dimensions of parallelism incurs additional complexity of efficiently traversing them to determine the optimal execution path for executing a given query.

Along with optimization, the heterogeneity of hardware requires concepts for reducing the data transfer cost among different devices. In a main memory DBMS, the device transfer bottleneck exists between main-memory and co-processing devices. Hence, it is also crucial to minimize the data transfer time for improving the efficiency of DBMS processing.

Hence, heterogeneity of functions and hardware has multiple challenges to be addressed and requires a system that is adaptable for these changes. In this work, we provide our insights on the challenges present in developing an adaptive database system and the techniques on overcoming these challenges. As there could be more functionalities and hardware available in future to be integrated into DBMS, we focus on a plug'n'play architecture that enables addition of these newer functions and hardware with considerably lesser

30^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 22.05.2018 - 25.05.2018, Wuppertal, Germany.

overhead than upgrading the complete architecture. This architecture provides interfaces for integrating different functionalities and hardware into DBMS with less effort.

The main contributions from our work are,

- The existing challenges for an adaptive DBMS in the context of hardware and software heterogeneity.
- The concepts for developing an adaptable DBMS with plug'n'play capabilities.

The subsequent paper is structured as follows. In Section 2, we provide an overview of the different devices used for DBMS and list out the challenges in using them. Then in Section 3, we discuss about the various challenges present due to functional and hardware heterogeneity in DBMS and in Section 4, we provide our concepts on developing an adaptable DBMS that addresses these challenges. The conceptual discussion in this paper are already explored in different works and we detail about them in Section 7. Finally, we provide the summary of the paper in Section 8.

2. DBMS IN HETEROGENEOUS HARDWARE ENVIRONMENT

Relying on CPUs as the working horse is approaching the limits of their efficiency [2]. There are multiple works conducted to port the existing database operations to different hardware. We discuss these devices and their DBMS support below.

GPU

Single core of the GPU has a lower clock frequency compared to a CPU core, a GPU features several hundreds of them compared to several tens of cores that current CPUs offer. However, especially memory accesses have high latency that needs to be hidden by processing. To this end, they spawn multiple threads for a given function and do context switching to hide the latency. This massive parallelism in GPU are useful in performing data intensive DBMS operations. Some of the DBMS using GPU are, CoGaDB, GPUDB, etc,. [3, 7].

The major open challenges in using GPU for DBMS are,

1. Cost model for determining the executable operator in GPU during runtime
2. Combined query compilation and execution strategies for CPU and GPU.

FPGA

Another hardware that has gained much attention in recent years is a FPGA (Field Programmable Gate Array). They are programmed either using RTL (Register Transfer Level) languages (VHDL, Verilog) or via HLS (High-Level Synthesis), where the circuits are extracted for example from C or OpenCL code. This provides a platform that can be tuned to perfection for any given domain specific operation providing higher throughput.

The open challenges in using FPGA are,

1. Selection and placement of operators for partial reconfigurable implementations
2. Efficient pipelining between different operations at runtime

Other Devices

There are other hardware used for DBMS are MICs (Many Integrated Core) and APUs (Accelerated Processing Units). In case of MIC, there are multiple CPU cores available for processing connected with each other using an on-chip bus system. These processors are capable of performing complex computations. Whereas, APUs have both CPU and GPU in a single die. Here, both both CPU and GPU have access to the same memory space (i.e. main memory).

3. CHALLENGES IN HETEROGENEOUS ENVIRONMENTS

To have a DBMS adaptable to both changing hardware and software the following challenges has to be addressed.

3.1 Device Features

Adding DBMS operation to a new processing device requires novel ways to exploit the device without compromising the overall system design. Hence, one of the major challenge is to reorganize the processing functions based on the hardware features available and must also adapt the underlying functions for efficient execution in the device.

3.2 Abstraction Hierarchy

It is shown that speedup gains for any particular database operation can be achieved by performing device-specific parameter tuning of the given operation [5]. Removing these device-specific parameters aids adaptability but makes it hard to tune for optimal execution as each device has its own advantage. Due to this polarity in abstraction versus specialization between functions and devices, it is required that we find a good abstraction level for the operations that provides both an interface to write new functions and also exploits the hardware for optimal efficiency.

3.3 Parallelism Complexity

The growth of DBMS in both functional and hardware level provides various parallelization opportunities. Presence of multiple devices creates an additional paradigm : cross-device parallelization. Using this type of parallelism, the given query is divided into granular parts based on the level of abstraction selected and these functional primitives are distributed among the different processing devices for parallel processing. We detail the different types of parallelization below,

Functional Parallelism

In multiple instances, the incoming queries have various suboperations that run independent to each other. One common example is the availability of multiple selection predicates combined using logical operations. These predicates can be executed in parallel among the different devices and the results are combined in next steps. Thus, the other way round: identifying and dissecting and identifying these parallel operations provide additional capabilities for simultaneous execution in the form of functional parallelism. The major challenge in this parallelism is the intermediate step of materialization of the results to be processed in the next operator in the pipeline. There is also a synchronization overhead present in this parallelism due to the differences in the execution time for different processing devices.

Data Parallelism

In contrast to functional parallelism, data parallelism does not split an operation into to different functions but executes same operation on different partitions of the data concurrently. This method also has a similar synchronization overhead of waiting for all the devices to finish processing. The major disadvantage of this parallelism is the additional step to merge results from different devices.

Cross-Device Parallelism

The above mentioned functional and data level parallelism are decided after the selection of processing devices. As we mentioned earlier, each devices have their own perks and must be utilized to the maximum extent. Hence, it is necessary to decide on the implementation details for the given device that exploits the hardware for efficient execution. Moreover, the above mentioned parallelization strategies can also be realized in the device level. In terms of device-level functional parallelism, it could be a multiple operator running in parallel in different devices or in a pipeline with communication within the devices. Similarly, the data parallelism could also be realized via suitable cost functions for operations on devices.

3.4 Optimization Strategies

The different levels of parallelism for execution of a query provide additional opportunities for fine tuning the operations but has the complexity of selecting optimal execution path. As the decision of top level parallelism influences the subsequent levels, selection of the right execution path for a given query is critical. However, the important drawback of this multi-level parallelism model is the search space explosion. There are various options available for any given level thereby having multiple combinations in total for selection. This search space of parallelism has to be traversed for finding the optimal execution path. Deciding the optimal path of a single operation in a query can be complex (e.g, join order optimization) which in addition with new dimensions of multiple devices increases the complexity further. Hence, newer methods for exploring the various optimization opportunities are to be determined.

4. ADAPTABLE DBMS

The mentioned challenges require a DBMS architecture that efficiently handles the diversity in both functionality and hardware. Based on the challenges, we have found areas to be explored for designing an adaptive DBMS.

For a better explanation of the challenges we use the TPC-H query6 as our motivating example. The query selects data from multiple columns, performs multiplication of results and outputs the aggregate. These three operations are in turn executed using multiple granular primitive functions. The different primitives used for processing the given query are,

- **Selection** primitive selects the values from the given column. Bitmaps are used as output format to reduce the data transfer size, as each bit carries the selection information of single value.
- **Logical Operation** primitives performs logical functions on the bitmaps produced by the different selections.

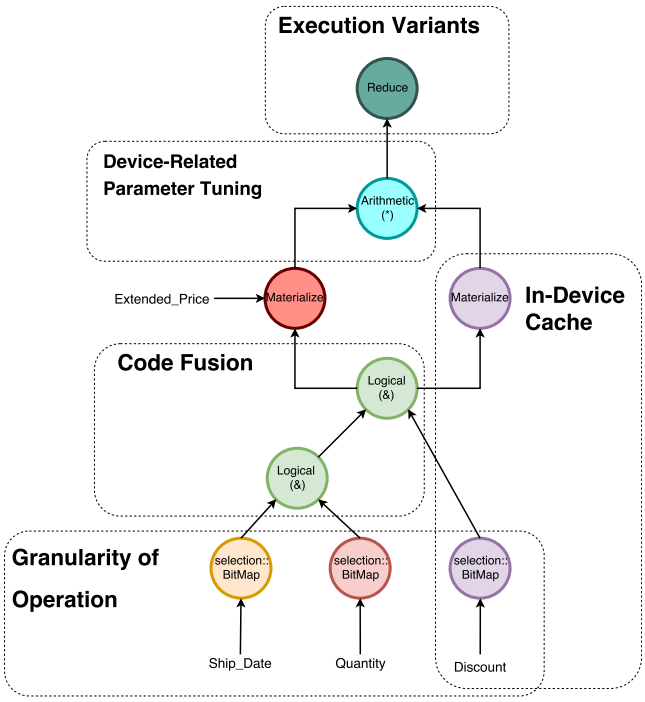

Figure 1: Optimization strategies for TPC-H Q6

- **Materialize** extracts the selected column values using the given bitmap input.
- **Arithmetic** performs arithmetic operation over column values.
- **Reduce** performs aggregation of values.

The flow of execution of query6 using these primitives is given in in Figure 1. The figure also illustrates different optimizations that can be done for a simple query like in the figure. We discuss about the various optimization strategies in the subsequent sections.

4.1 Granularity of Operation

One of the main challenges in the proposed adaptable system is the level of granularity required for optimized processing. Based on the capabilities of the devices, we could either run a few complex operation or split them into more granular sub-operations and then also execute those ins parallel among multiple devices.

At the top level, each database operation acts as a set of primitives connected together to provide a final result. The more granular a function is split, the more hardware sensitiveness comes into play. For example, the access patterns in CPU and GPU are different for efficient processing. Further, database operations are data centric where every operation is applied to a massive amount of data. To aid parallel data processing, we propose the use of explicitly data parallel primitives to be combined into complete DBMS operations. There are many works on primitive based DBMS query processing. He et al., propose multiple primitives such as Split, Filter, etc., for GPUs [7]. Other primitives such as prefix-sum and its variants, scatter and gather are also proposed for efficient data parallel

execution [6]. This approach provides a fail safe: when a newer device is added the primitives could still run on them with minor changes to the functionality. This availability of different granular levels provide additional benefit enabling developer to replace the inefficient fine-granular primitives with custom coarse-granular ones.

4.2 Code Fusion

Implementing primitives in multiple granular levels becomes time consuming. Hence, code could be generated at runtime for the given granularity level of the operation. This code for execution in an individual device is generated by combing the primitives for the corresponding device into single execution process. This reduces the overhead of materializing data from intermediate steps.

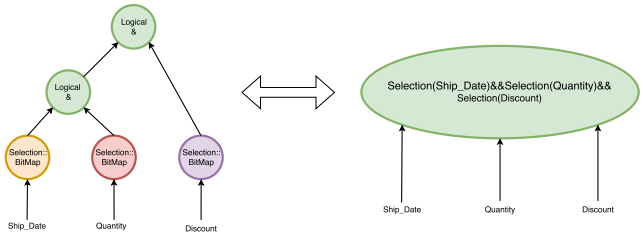

Figure 2: Code synthesis - selection

For example, three selection predicates as shown in Figure 2 can be either run in different devices (left) and the results are combined using the logical operations, or the predicates are all combined into single execution (right).

4.3 In-Device Cache

The current data-transfer bottleneck is between the main memory and the processing devices itself. CPU has faster access than other devices as it is directly linked to the main memory, whereas in case of the co-processors, data must be transported via connections with higher latency and possibly more limited bandwidth, such as PCIexpress. Thus, even highly efficient GPUs can have sub-optimal performance than CPUs due limited access capabilities to main memory. Hence, using device memory as data cache is crucial for high compute throughput. In contrast to this, these external devices have limited memory. Hence, it is not always possible to store all the necessary data on the device itself. Thus, the host system must determine the hot set of data to be stored in the device memory using the execution plan for the given query and monitoring the data transfer to the device and .

4.4 Execution Variants

Each primitive selected for executing a given query can have different characteristics to choose from based on the executing device. For example, complex branching statements are handled efficiently by CPUs, whereas GPUs are capable of massive thread level parallelism with less control flow. In addition, the data access pattern must be selected the memory architecture of the given device. For example, coalesced data access provides efficient memory access in GPU. Finally, hardware specific vectorization of DBMS operations (SIMD) is also an important parameter in database processing to exploit the hardware capabilities.

Also in an abstract level, characteristics of the primitive itself can affect system throughput. The choice output format and the number of intermediate steps are some of the characteristics that influence the overall system. For example, using bitmap results from selection in external devices will be generally more efficient than transferring complete column.

4.5 Device-Related Parameter Tuning

Finally, once we have decided on the device and its corresponding function to execute, certain device related parameters like global and local work group sizes have to be tuned for further improvement of the overall efficiency. These device related parameters are tuned for efficiency by monitoring the performance of execution. There is a feedback loop from the devices, providing execution specific informationused for tuning the primitive for higher efficiency.

Other than these above mentioned challenges, one of the major challenge is to formulate an order for using the strategies to extract an efficient execution plan. Since all the strategy mentioned above are inter-dependent, selection of one depends on the other. In order to have a standardized execution flow, we propose an architecture that has all the necessary components used for using the above strategies.

5. CONCEPTUAL ARCHITECTURE

As mentioned earlier, the overall efficiency of processing a query in a heterogeneous environment requires all the mentioned optimization strategies to be applied to a given query. To aid this, we propose a DBMS architecture that provides a structure to handle the optimization from global abstraction to local device specific levels. The structure is shown in Figure 3.

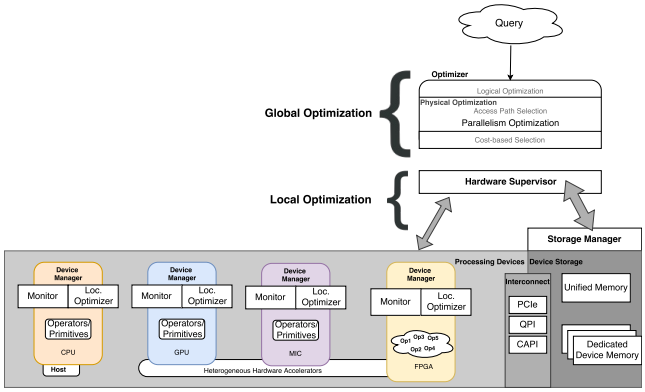

Figure 3: Conceptual Architecture

The given query is first subjected to the general logical optimization and access path selection steps. Global optimization is done over the resultant query from logical and access path selection steps. This step determines the level of granularity for the given query. once selected, these granular operations are provided to their respective device based on decision given using hardware supervisor. Finally, local optimization is done the granular operations to tune for their respective devices and the kernels that always work together are combined together. The components used for these optimizations done are discussed below,

Global Optimizer: The global optimizer processes the complete query and provides the best execution plan for the whole query. It decides on the level of granularity to be used as primitive. In addition to the different granularities, the parallelism strategies (i.e. pipeline and data) are also selected here.

The different schema available for executing a single query leads to search space explosion. Traversing the whole design space might be time consuming and hence a machine learning based cost estimation algorithm is used [4].

Hardware Supervisor: The hardware supervisor provides statistical information about the underlying devices. This helps in improving the decisions made by the global optimizer. It combines the characteristics of individual devices into a single integrated system view. This also communicates with the devices and supervises execution of operations in the individual devices.

Storage Manager: The storage manager provides information about the location and availability of data to be processed. This aids in determining the transfer costs in order to aid selecting the execution device. Also, it is evident that not all devices have direct access to the main memory. Hence, it is the task of storage manager to partition and transfer data to the respective devices.

Device Manager: Each device manager has two subcomponents: Monitor and Local optimizer. Monitors provide device specific information and the local optimizer holds information about the primitives implemented in the corresponding device and also about the current workload. It uses these information to perform device specific optimizations to further increase the processing efficiency of a given operation.

6. PRELIMINARY EVALUATION

To evaluate the efficiency of different parallelism mechanisms, we executed the TPC-H query 6 by combining five different primitives namely, Bitmap, Logical, Materialize, Arithmetic and Reduce. All these primitives are data parallel and are implemented using OpenCL. The execution path for the query is shown in Figure 1. For our evaluation, we considered four different execution models as explained below.

Baseline linear execution: In the baseline version, we execute the linear Q6 compiled query without parallelism or primitives. The result of this execution are used as a benchmark to compare with other parallel implementations.

Single Device Primitives (SDP): In singular device primitive version, the parallel primitives mentioned above are executed in a single device. The results for complete execution of parallel primitives in both CPU and GPU are recorded for analysis.

Multiple Device Pipelined (MDP): In the multiple device pipelined variant, we split the query into two phases: selection and aggregation and execute them in a pipeline. We perform selection in CPU and aggregation in GPU (MDP - CPU + GPU) and vice versa (MDP - CPU + GPU) recording their results.

Cross-Device Functional Parallel (CDFP): Finally, the given query is split into functional units and the independent units are executed concurrently in the devices.

All these models are executed on a machine running Ubuntu OS version 16.04 and gcc version 5.4.0 with Intel Core i5 CPU and Nvidia Geforce 1050 Ti GPU.

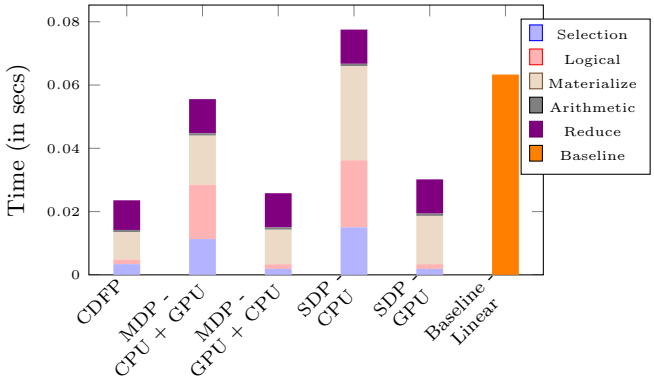

Figure 4: Execution model Variants- results

From the results, we see that the single device execution model for CPU has the lowest efficiency for processing Q6 and is even slower than the linear execution variant. This is due to the additional materialization step to be performed. In case of single device execution of the query in GPU, the system is nearly 2.5x faster than the CPU variant and 2x faster than the scalar version.

For the multi device pipelined model, we see the CPU selection with GPU reduce variant is 2x slower than its counterpart. The selection phase in CPU takes considerable time for processing the select, logical and materialize primitives, whereas GPU selection higher execution time only for materializing the values.

Finally, we see that cross-device functional parallelism model has the highest efficiency in processing the query. This is mainly due to the multiple selection predicates available in the query. The latency of execution is reduced by executing the selection and materialization steps in parallel. The detailed information of the execution of individual primitives in this variant is shown in Figure 5.

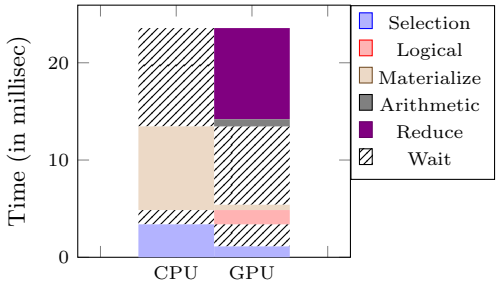

Figure 5: Cross-device parallelism - results

From the chart, we see the devices wait at multiple instances for the other to finish to continue processing the query. In case of selection and materialization, GPU waits until CPU has processed its values before executing the next results. Also, the CPU is idle when the GPU is computing the results of arithmetic, logical and aggregation operations.

From these results, we infer that using functional parallelism enhances the efficiency of query processing. The advantage of functional parallelism comes with the disadvantage of synchronization overheads due to differences in processing speed among the different devices.

7. RELATED WORK

Karnagel et al., have explored the adaptivity in DBMS using primitives for executing a query [10]. They group a subset of primitives to be executed in a single device into an execution island and process them. Their also use device level caching to reduce transfer overhead. Once the intermediate result for an island is computed, an intermediate estimation step is done to select the subsequent devices. In our method, the execution path is given by an optimizer and is executed in by the devices.

In terms of granularity of operators, He et al,. have given a comprehensive set of data parallel primitives that can be ported into various hardwares [7]. Our research complements theirs by adding new primitives and additional functionalities to the already defined ones. Similarly, Pirk et al,. have also given an abstracted set of primitives that could be used in various platforms [11].

8. CONCLUSION

We detailed in this paper, the need for an adaptive architecture for DBMS that can be easily modified based on the underlying hardware and the software functionalities. In this adaptable DBMS the executable operations must be generalized for high interoperability whereas, device specific operations are needed for higher efficiency. Along with challenge in selecting the right abstraction level, there are multiple challenges available for an adaptable DBMS in a heterogeneous environment. Our main contribution in this work is the framework for overcoming these challenges with the concepts listed below,

- Granular levels for DBMS operations
- Device specific code generation
- In-device data caching techniques
- Device and functional variants of operator
- Hardware and functionality based tunable parameters

The interfacing of different components of DBMS is a challenging task in itself. A plug'n'play architecture in DBMS removes these overheads by providing interfaces for supporting additional hardwares and softwares. Also, an adaptable DBMS could additionally help in optimizing a new functionality that is formed by combing the given set of granular primitives as the primitives are in itself tuned for efficiency. Finally, this adaptive architecture of DBMS de-couples the functional and device based execution layers thereby providing independence between the operation and its corresponding execution unit.

9. ACKNOWLEDGMENTS

This work was partially funded by the DFG (grant no.: SA 465/51-1 and PI 447/9)

10. REFERENCES

[1] P. Bakkum and K. Skadron. Accelerating SQL database operations on a GPU with CUDA. *Proceedings of the Workshop on General-Purpose Computation on Graphics Processing Units*, pages 94–103, 2010.

[2] S. Borkar and A. A. Chien. The Future of Microprocessors. *Communications of the ACM*, pages 67–77, 2011.

[3] S. Breß, M. Heimel, N. Siegmund, L. Bellatreche, and G. Saake. GPU-accelerated database systems: Survey and open challenges. *Transactions on Large-Scale Data- and Knowledge-Centered Systems*, pages 1–35, 2014.

[4] S. Breß and G. Saake. Why it is time for a HyPE. *Proceedings of the International Conference on Very Large Databases*, 6(12):1398–1403, 2013.

[5] D. Broneske, S. Breß, M. Heimel, and G. Saake. Toward hardware-sensitive database operations. In *Proceedings of the International Conference on Extending Database Technology*, pages 229–234, 2014.

[6] Y. Dotsenko, N. K. Govindaraju, P.-P. Sloan, C. Boyd, and J. Manferdelli. Fast Scan Algorithms on Graphics Processors. In *Proceedings of the Annual International Conference on Supercomputing*, pages 205–213, 2008.

[7] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, and P. V. Sander. Relational Query Coprocessing on Graphics Processors. *ACM Transactions on Database Systems*, pages 21:1—-21:39, 2009.

[8] J. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The MADlib Analytics Library or MAD Skills, the SQL. *Proceedings of the International Conference on Very Large Databases*, pages 1700–1711, 2012.

[9] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li. Big data processing in cloud computing environments. *Proceedings of the International Symposium on Pervasive Systems, Algorithms and Networks*, pages 17–23, 2012.

[10] T. Karnagel, D. Habich, and W. Lehner. Adaptive Work Placement for Query Processing on Heterogeneous Computing Resources. *Proceedings of the International Conference on Very Large Databases*, pages 733–744, 2017.

[11] H. Pirk, O. Moll, M. Zaharia, and S. Madden. Voodoo - A vector algebra for portable database performance on modern hardware. *Proceedings of the International Conference on Very Large Databases*, pages 1707–1718, 2016.

[12] K.-U. Sattler and O. Dunemann. Sql database primitives for decision tree classifiers. In *Proceedings of the International Conference on Information and Knowledge Management*, pages 379–386, 2001.