SiliconDB - Rethinking DBMSs for Modern Heterogeneous Co-Processor Environments

Kayhan Dursun[†], Carsten Binnig[†], Ugur Cetintemel[†], Robert Petrocelli[*]
[†] Brown University, Providence, USA
[*] Oracle Corporation, Redwood City, USA

ABSTRACT

In the last decade, the work centered around specialized co-processors for DBMSs has largely focused on efficient query processing algorithms for individual operators. However, a major limitation of existing co-processor systems is the PCI bottleneck, which severely limits the efficient use of this type of hardware in current systems.

In recent years, we have seen the emergence of a new class of co-processor systems that include specialized accelerators, implemented as ASICs or FPGAs, which co-reside with the CPU on the same socket. Here we revisit DBMS architectures in this context, and take an initial step towards the design of a new database system called *SiliconDB* that targets these new densely integrated heterogeneous co-processor environments.

1. INTRODUCTION

Motivation: Recent work on specialized co-processors for DBMSs (e.g., FPGAs [8, 13], GPUs [2], etc.) has largely focused on the efficient implementation of query processing algorithms for these devices. The results have not only shown that specialized co-processors are able to provide high speedups for individual database operators but also are more energy efficient in most cases. However, a major limitation of existing co-processors is the PCI bottleneck, which makes the efficient use of these devices in existing DBMSs challenging [11][12]. To address this limitation, the key optimization goal of existing approaches that integrate co-processors in an end-to-end manner is to reduce the overall communication cost such as CoGaDB [1] and Ocelot [2]. Even when using these schemes, the high speedup rates reported for individual operators often vanish when looking at the overall runtime that includes the communication costs between CPU and the co-processor [3].

Recently, a new class of co-processors where accelerators co-reside with the CPU on the same die has emerged. This trend is primarily motivated by the so-called dark silicon [4][7]; i.e., the areas on the CPU die that cannot be populated with general-purpose cores because of their energy consumption. As a result, many CPU manufacturers have announced plans to integrate specialized units densely together with the regular cores. Developing efficient DBMS systems for these new co-processor environments, however, demands a critical rethinking of many of the basic architectural and design assumptions that are not valid anymore. For example, co-processors are no longer attached to the CPU via a slow PCI connection but instead share the access to the same memory bus and sometimes even have direct access to the last level caches of the CPU cores. Consequently, NUMA-awareness and cache-locality become key factors when designing query processing algorithms on these co-processor environments. Moreover, in order to meet the desired energy restrictions and optimally leverage the real estate on the CPU die, the next generation of co-processors are rather specialized processing units that are implemented either as an ASIC or an FPGA. Examples of such systems include the Oracle SPARC M7 [10] processor, which provides specialized database accelerator (DAX) units on the same die with the cores, and the Intel HARP [6], which replaces a normal core on the die with an FPGA unit. For M7, the DAX engines can even write their output to an associated last level cache of the CPU cores.

In addition, these co-processors are not general purpose and can only support a limited set of functions during runtime. While FPGAs are in general reprogrammable, the cost of reconfiguration is high, meaning that the functionality that an FPGA provides at runtime while executing a query is typically fixed as well.

Contributions: In this work, we describe the design towards a new database system called *SiliconDB* that targets state-of-the-art heterogeneous co-processor environments. The challenges that we tackle are two-fold. (1) In order to achieve high performance, our goal is to leverage cache-locality and take into account NUMA effects, which were negated by the high transfer costs incurred by the PCI bus [12] in previous systems. We anticipate that there will be many different configurations of heterogeneous environments that will be specialized for different workloads. To accommodate all these configurations, new DBMSs need to be designed with general and adaptable building blocks. (2) Query execution strategies need to change as well. Instead of using coarse-grained placement strategies that aim to minimize synchronization and data transfers between the CPUs and the co-processors, we develop fine-grained, dynamic workload assignment approaches that can concurrently leverage all computing resources, assigning pieces of the load where they can be executed most efficiently while maximizing the overall resource utilization.

We use the Oracle Sparc M7 processor as an early example of such an environment to rethink the architecture

DaMoN'17, May 15, 2017, Chicago, IL, USA

ISBN 978-1-4503-5025-9/17/05…$15.00

DOI: http://dx.doi.org/10.1145/3076113.3076124

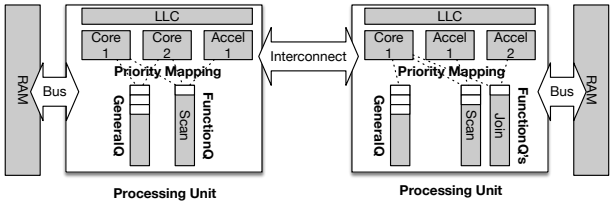


Figure 1: *SiliconDB* Architecture

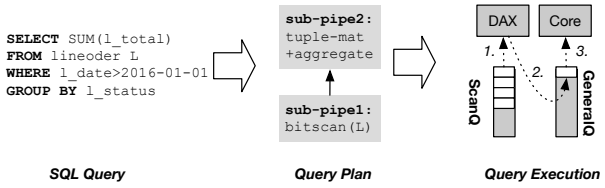


Figure 2: Fine-Grained Query Execution

of DBMSs for analytical column stores, but our approach will be adaptable to other emerging environments as we will study in the future including the support of different analytical workloads.

Outline: In Section 2, we discuss the general architecture of *SiliconDB* that reflects the requirements mentioned above. In Section 3, we present new fine-grained query execution strategies for heterogeneous environments that we developed to leverage all available resources. We present our initial experimental results in Section 4. After summarizing the related work in Section 5, we will conclude the paper in Section 6.

2. SYSTEM ARCHITECTURE

Figure 1 shows the general architecture of *SiliconDB*. A key idea behind the design is grouping of cores and accelerators into so-called *processing units*. In this way, cores and accelerators can access to the same memory regions to attribute to the NUMA effects as well as to leverage effects of cache-locality. For example, in Oracle SPARC M7, DAX engines can write the output of an operation to an associated LLC.

While NUMA-awareness and cache-locality have been extensively studied in the design of main-memory DBMSs, the novel aspect of the architecture is the internal design of each of these processing units: First, the structure of a processing unit is not static; i.e., the grouping of cores and accelerators can be defined in a flexible manner which would allow our approach to adapt to many different hardware configurations with a varying number of cores and accelerators. Second, a processing unit implements a novel fine-grained execution model to maximize resource utilization. In order to achieve this, each processing unit defines a set of function-specific work queues as well as a priority mapping between the cores and the queues for work scheduling. Each function queue contains work elements of a specific work type supported by an accelerator (e.g., a scan queue or a join queue), whereas each work element refers to be a small block of data that needs to be processed (i.e., a fixed number of values of a column in our case). Additionally, each processing unit also has a general queue, which contains all the work elements that are not supported by a general accelerator. The main goal of query execution is then to schedule these work elements such that all available compute resources including the accelerators are leveraged in the most efficient way. Furthermore, in order to support different architectures, *SiliconDB* is designed to be extensible; i.e., new function queues can easily be added.

In the following, we describe the details of the query execution scheme of *SiliconDB* in detail.

3. QUERY EXECUTION

SiliconDB first compiles a given query into a pipelined representation similar to [9] at compile time. However, a difference is that it decomposes the pipelines into smaller sub-pipelines on the basis of the specific functions that are supported by the accelerators of a processing unit. For example, Figure 2 shows the query plan for a TPC-H Q1-like query with two separate sub-pipelines: The first sub-pipeline represents a scan on a bit-compressed column that produces a bit-vector as its output – an operation that is supported by a DAX unit in the Oracle M7. Then the subsequent sub-pipeline uses the bit-vector to materialize the selected tuples and to execute the aggregation on them.

In order to start the execution, *SiliconDB* first adds the work elements for the scan sub-pipelines into a queue of the processing units. Later on, each core/accelerator pulls work elements from specific queues to process them. New work elements are added during execution as a result of a finished work element; e.g., after finishing a scan work element in our sample query, an aggregation work element is added to the general work queue. As mentioned before, while accelerators can only pull work elements from their particular function queue, cores can pull work from any queue.

For cores, the priority mapping defines the order of queues that they should pull work from: in *SiliconDB* this mapping defines that a core should first try to pull work elements from the general queue, since these elements can not be processed by any accelerator. In case no work elements are available in the general queue, the cores can start pulling work elements from other function queues (e.g., to execute scans). As we show in our experiments in the next section, this priority mapping results in a better utilization of all resources (cores/accelerators) than existing execution schemes provide. Additionally, this scheme enables a streaming execution mode between cores and accelerators, which further increases the utilization. For instance, in the case of the query plan shown in Figure 2, the DAX engines and cores start working on the scan elements in the beginning. Then, once the first scan elements are processed and new work elements for the aggregation become available in the general queue, cores start to pick these up for processing while the DAX engines continue working on scans.

A challenge of our execution scheme as described before is that many of the accelerators are typically passive (i.e., a core must assign work to the accelerator). To that end, we defined two different query execution schemes to push work elements to passive accelerators: (1) *Work Handler:* In this scheme, in each processing unit, one core is reserved to pull work elements from different function queues, assign them to the according accelerator and monitor them to create follow-up work elements once they finish processing the assigned work element. (2) *Piggybacked:* In this scheme, there is no dedicated core to assign work to the accelerators. Instead,

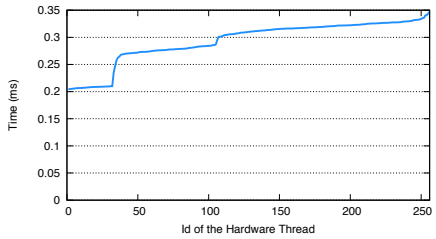


Figure 3: Effect of Cache-Locality

each core assigns work elements to an accelerator just before it starts working on a work element. In order to avoid the accelerators becoming idle in this scheme, a core can assign multiple work elements to an accelerator in every round. The number of elements that are assigned to an accelerator by a core is determined dynamically using queueing theory. Further details are omitted for brevity.

4. EXPERIMENTAL EVALUATION

In the evaluation, we show the results of two experiments for (1) analyzing the effects of cache-locality when sharing caches between cores and accelerators, and (2) comparing the utilization of cores and runtime of queries for our execution strategies compared to existing strategies. In both experiments, we use a TPC-H database of SF=10 and execute the query shown in Figure 2.

The prototype of *SiliconDB* is implemented in C++ and compiled using GCC 4.82. All experiments have been executed on a machine at Oracle with 128GB of RAM and one SPARC M7 processor (32 cores, 8 DAX engines) running Solaris 11.3 as operating system. In the SPARC M7 processor, 4 cores and 1 DAX engine share an 8MB L3 cache. Moreover, each of the cores supports 8 strands (which are similar to hyper-threads) resulting in a total of 256 hardware threads for one server with 32 cores.

4.1 Exp. 1: Effects of Cache-Locality

In this experiment, we executed the query in Figure 2. In order to reveal the effects of cache-locality between a DAX engine and a normal core, we first executed the scan operation that outputs a bit-vector on one fixed DAX engine and then executed the tuple materialization and aggregation in a software thread using a normal core. The output of the scan is stored in the L3 cache assigned to the DAX engine. We used a block size of $1M$ elements for our fine-grained execution scheme in this experiment such that the output bit-vector of the scan can fit completely in the L3 cache. For executing the aggregation, we pinned the software thread to one of the available hardware threads of SPARC M7. We repeated the experiment for all available 256 hardware threads.

The results of the experiment can be seen in Figure 3. The x-axis shows to which hardware thread we pinned the software thread and the y-axis represents the average runtime of the software thread on a block size of $1M$ elements. We see that the runtime of the tuple materialization and the aggregation when the software thread is pinned to one of the first 32 hardware threads is significantly lower. The reason is that these hardware threads are executed by the 4 cores that share the cache with the DAX engine, which produces the bit-vector. In our results, we can also see another increase of runtime when using the hardware threads from 128 to 256. The reason for this are NUMA effects that result from different latencies to access the two different NUMA regions using an on-chip network on the SPARC M7 processor.

4.2 Exp. 2: Efficiency of Execution Strategies

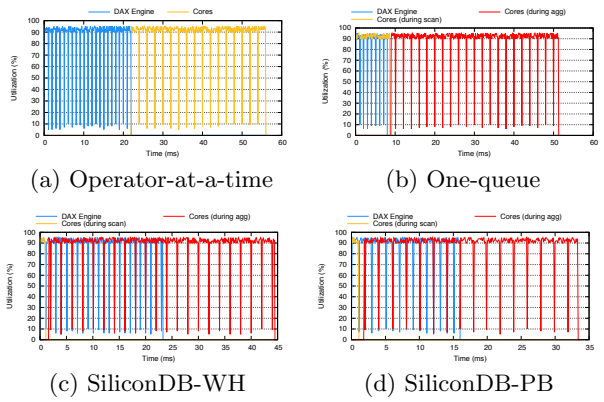
In the second experiment, we show the benefits of our execution strategies discussed in Section 3 when using our two variants; Work Handler (WH) and Piggybacked (PB). In order to show the utilization of cores/accelerator we configured *SiliconDB* to use one processing unit that groups together four cores and one DAX engine (all of which share the same L3 cache). Moreover, in this experiment we used a block-size of $128K$ for our fine-grained execution scheme.

To compare our execution scheme to existing schemes, we also executed the query in Figure 2 using two other execution as baselines. The first baseline is the *operator-at-a-time* scheme that is commonly used in DBMSs for heterogeneous co-processor environments today. In this scheme, the scan is completely executed in the DAX engine and the aggregation in all the available cores of our processing unit. For the second baseline, we implemented a *naive fine-grained scheme* similar to our scheme presented before that only uses one shared queue that contains all work elements instead of separating the queues into function queues and a general queue. In this scheme, work elements are assigned greedily to the next available core/accelerator that is able to execute the work element. Moreover, we reserve one core as a handler to assign work to the DAX engine.

The results of this experiment are shown in Figure 4. In Figure 4(a), we see the resulting utilization over time of the operator-at-a-time scheme. As expected, the DAX engine is first fully-utilized when executing the scan and then drops to 0%-utilization while the cores execute the aggregation. For the naive fine-grained scheme, we can see that both the cores and DAX engine start cooperatively working on the scan. However, once all scan elements are processed, the DAX utilization again drops to 0% while the cores finish the aggregation. In our schemes, which implement separate queues and thus enable a streaming model where the DAX engine works on a scan while the cores process the aggregation, the overall utilization is much better as expected; i.e., the DAX and cores are utilized much more equally over the query execution time. Another important result is that, while the utilization of both of our heterogeneous fine-grained schemes (Work Handler and Piggybacked) seems to be very similar, the overall runtime of the Piggybacked version is approximately 25% lower since all four cores can be used to execute meaningful work instead of reserving one core as a DAX handler.

5. RELATED WORK

There exists a large body of work on how to leverage specialized co-processors for analytical database workloads (e.g., FPGAs [8, 13], GPUs [2], etc.). The main limitation of existing co-processors is the PCI bottleneck, thus the main goal of existing approaches that integrate co-processors in an end-to-end manner into a DBMS has been to reduce the overall communication [1, 2]. However, even when using these schemes, the high speed-up rates reported for individual database operators often become negligible when considering the overall runtime that includes the communication costs between CPUs and the co-processors [3]. A key contribution of our work is to revisit the design of a co-processor



(a) Operator-at-a-time (b) One-queue

(c) SiliconDB-WH (d) SiliconDB-PB

Figure 4: Efficiency of Execution Strategies

accelerated DBMS where the accelerators are tightly integrated with the cores.

In [11] and [12], authors address a similar problem for an architecture where a CPU and GPU are integrated on a single chip. To show the effects of using an integrated GPU, they propose specialized scan and aggregation operations. They were able achieve a 3× performance boost compared to an architecture where the GPU sits at the end of the PCI bus, even though the integrated GPU has 4× lower computational power than a discrete GPU. While we share the high-level motivation of this work, our approach differs in various aspects. First, our focus is not on specialized operator implementations but on query scheduling to leverage all compute resources in the best possible manner. Second, another goal of the architecture of *SiliconDB* and our scheduling strategies is to be able to adapt to new architectures with other types of accelerators without the need to re-design the entire DBMS stack.

Finally, our scheduling scheme also has similarities with the approach used in QPipe [5]. While QPipe uses a similar queue-based concept to schedule work for different database operators (similar to our function-specific work queues), there are some important differences: First, the focus of the scheduling strategies in QPipe is on the sharing of data and work between queries, whereas our focus is to maximize the utilization of all accelerators and cores. Second, different processing units might share the same queue to enable dynamic scheduling decisions, which is a key aspect of *SiliconDB* that allows it to adapt to different co-processor architectures.

6. CONCLUSIONS

Previous studies on specialized co-processor environments to support analytical database workloads mostly focused on addressing the data transfer bottleneck. The emergence of new co-processor environments creates a new category of optimization opportunities which were not addressed in past work, as these were primarily concerned with other bottlenecks, such as the expensive data movement costs between the CPU and accelerator units over relatively slow PCI-e links.

In this paper, we present *SiliconDB*, which we are building to address these new bottlenecks and present our initial results in comparison to existing solutions. Our approaches take cache locality and NUMA effects into account with the support of an underlying architecture that is designed to target these issues. We are using Oracle Sparc M7 processor as an early example, but our work and results are more general, since the adaptive nature of our design will render our solutions applicable to different types of heterogeneous processor environments.

7. ACKNOWLEDGMENTS

This work is partially supported by Oracle Corporation by a research gift. We also thank Angelo Rajadurai for providing us access to the necessary hardware for our benchmarks.

8. REFERENCES

[1] S. Breß. The design and implementation of cogadb: A column-oriented gpu-accelerated DBMS. *Datenbank-Spektrum*, 14(3):199–209, 2014.

[2] S. Breß et al. Ocelot/hype: Optimized data processing on heterogeneous hardware. *PVLDB*, 7(13):1609–1612, 2014.

[3] S. Breß et al. Robust query processing in co-processor-accelerated databases. In *SIGMOD*, pages 1891–1906, 2016.

[4] N. Hardavellas et al. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, 2011.

[5] S. Harizopoulos et al. Qpipe: A simultaneously pipelined relational query engine. In *SIGMOD*, pages 383–394, 2005.

[6] Intel. Harp. https://www.ece.cmu.edu/˜calcm/carl/lib/exe/fetch.php?media=carl15-gupta.pdf.

[7] R. Johnson et al. The bionic DBMS is coming, but what will it look like? In *CIDR*, 2013.

[8] R. Müller et al. Fpgas: a new point in the database design space. In *EDBT*, pages 721–723, 2010.

[9] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9):539–550, 2011.

[10] Oracle. Sparc m7. www.oracle.com/SPARC-M7.

[11] J. Power et al. Implications of emerging 3d gpu architecture on the scan primitive. *SIGMOD Record*, 44(1):18–23, 2015.

[12] J. Power et al. Toward gpus being mainstream in analytic processing: An initial argument using simple scan-aggregate queries. In *DaMoN*, pages 1–8, 2015.

[13] J. Teubner et al. *Data Processing on FPGAs*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013.