Energy-Efficient Query Processing on Embedded CPU-GPU Architectures

Xuntao Cheng
LILY, Interdisciplinary Graduate School
Nanyang Technological University, Singapore
xcheng002@ntu.edu.sg

Bingsheng He
School of Computer Engineering
Nanyang Technological University, Singapore
bshe@ntu.edu.sg

Chiew Tong Lau
School of Computer Engineering
Nanyang Technological University, Singapore
asctlau@ntu.edu.sg

ABSTRACT

Energy efficiency is a major design and optimization factor for query co-processing of databases in embedded devices. Recently, GPUs of new-generation embedded devices have evolved with the programmability and computational capability for general-purpose applications. Such CPU-GPU architectures offer us opportunities to revisit GPU query co-processing in embedded environments for energy efficiency. In this paper, we experimentally evaluate and analyze the performance and energy consumption of a GPU query co-processor on such hybrid embedded architectures. Specifically, we study four major database operators as micro-benchmarks and evaluate TPC-H queries on CARMA, which has a quad-core ARM Cortex-A9 CPU and a NVIDIA Quadro 1000M GPU. We observe that the CPU delivers both better performance and lower energy consumption than the GPU for simple operators such as selection and aggregation. However, the GPU outperforms the CPU for sort and hash join in terms of both performance and energy consumption. We further show that CPU-GPU query co-processing can be an effective means of energy-efficient query co-processing in embedded systems with proper tuning and optimizations.

1. INTRODUCTION

In the era of Internet of Things (IoT), embedded devices are pervasive. In 2011, the number of embedded devices overtakes the human population and it is expected to reach 24 billion by 2020 [6]. Databases are an integral component on embedded devices. For example, embedded applications usually use database systems such as SQLite and InnoDB as their storage and query processing backend. Energy-efficient query processing has been a major factor for the design and optimization of database systems on embedded devices, which are essential to many IoT applications due to the battery limitation. Recently, because of the evolution of GPUs' programmability and computational capability driven by the need to offer high performance for more general-purpose applications, GPUs have been incorporated in new-generation embedded devices. The CARMA development board coupling a quad-core ARM Cortex-A9 CPU and a 96-core NVIDIA Quadro 1000M GPU is an embedded device of this kind. Previously, GPUs have also been an effective means to improve the performance of main-memory databases in servers and workstations. Such emerging CPU-GPU architectures bring exciting research opportunities on CPU-GPU query co-processing in embedded environments for energy efficiency.

Energy efficiency of databases has been a recent hot research topic [19, 17]. However, most studies focus on CPU-based architectures. Unlike CPUs, there have been few studies on energy saving technologies for GPUs like Dynamic Voltage and Frequency Scaling (DVFS). GPUs are considered to be very power hungry. The embedded GPU consumes at least about 3 to 4 times more power than the embedded CPU on the CARMA board. It is worthwhile to revisit query co-processing on embedded CPU-GPU architectures for energy efficiency carefully.

In this paper, we experimentally evaluate and analyze the performance and energy consumption of a GPU query co-processor on such hybrid embedded architectures. Specifically, we select four such query operators: selection, aggregation, sort and hash join as micro-benchmarks and evaluate TPC-H queries on such a GPU query co-processor. We adopt the state-of-the-art open-source implementations of query operators on both CPUs and GPUs [7, 23, 2] with modifications and improvements required to adapt to embedded CPU-GPU architectures. We make the following contributions:

- We have studied the impacts of GPUs in query co-processors on embedded devices regarding of performance and energy efficiency by studying four common database operators: selection, aggregation, sort and hash join as micro-benchmarks and TPC-H queries on such an architecture.
- We have found that the CPU delivers both better performance and lower energy consumption than the GPU for simple operations such as selection and aggregation. The GPU outperforms the CPU in other cases. The CPU-GPU query co-processing can be an effective means for energy-efficient query co-processing on embedded devices with proper tuning and optimizations.
- We have discussed a number of opportunities for query co-processing on future embedded CPU-GPU architec-

DaMoN'15, June 1, 2015, Melbourne, VIC, Australia
Copyright 2015 ACM 978-1-4503-3638-3/15/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2771937.2771939.

tures including new memory transfer technologies and potential optimizations.

The rest of the paper is organized as follows. In Section 2, we introduce background and related work on energy savings on embedded systems and GPU query co-processing. Next, we present implementation details of the query co-processor on embedded architectures in Section 3. Experimental results are presented in Section 4. We discuss future architectures based on our findings in Section 5 and finally conclude this paper in Section 6.

2. BACKGROUND AND RELATED WORK

In this section, we introduce the background of energy savings on embedded devices and GPU query co-processing.

2.1 Energy Savings on Embedded Systems

Many embedded devices are often considered as *wimpy* nodes since they have limited memory, low co-processing power and low power consumptions. Recently, *wimpy* nodes are used to build more energy-proportional and large-scale systems. The FAWN cluster made from low power embedded CPUs was proposed for key-value store [1]. It connects a large number of low power embedded processors coupled with flash memories to achieve high energy efficiency [20]. Schall et al. [18, 17] demonstrated a DBMS built on *wimpy* nodes which can dynamically adjust the power of nodes based on its current workload. Dumitrel et al. recently observed that database query co-processing is energy-efficient on an ARM based cluster at the cost of marginally slower throughput [3]. The Pedraforca project [13] represented the cases for embedded nodes of CPU-GPU architectures towards energy-efficient clusters.

Another approach towards energy savings is to use *beefy* nodes and *wimpy* nodes in a hybrid fashion, where the normal servers and workstations are considered as beefy nodes. Mühlbauer et al. [14] demonstrated a DBMS applicable for both ARM processors and x86-64 processors. Lang et al. proposed the heterogeneous cluster as a new design point [11] for the energy efficiency of data warehousing workloads. The advantage of their design is that workloads which can not fully utilize the *beefy* nodes can be offloaded to *wimpy* nodes in order to improve the energy efficiency. Wong et al. [21] proposed a hybrid approach by attaching a *wimpy* node to a *beefy* node in order to shift the workload to the *wimpy* node and put the beefy node into the "sleep" state when the load is light.

Most existing studies of this kind are focusing on applications of *wimpy* CPU, predominantly ARM processors. Less attention has been paid to energy savings on embedded GPUs. Anuj et al. took advantage of the recently available DVFS capabilities of some embedded GPUs to achieve a flexible power-performance trade-off [15]. Kai et al. proposed a two-tier design coordinating the CPU and the GPU to let them finish at the same time achieving reasonable energy savings [12].

2.2 GPU Query Co-Processing

GPUs are considered as an effective means for improving query co-processing performance on main memory databases. Database queries and relational operators on GPUs have been extensively studied in previous work. He et al. designed a set of highly optimized data-parallel primitives and common relational query algorithms on GPUs [7]. Yuan et al. debunked reasons why GPUs were not used for warehousing queries and showed that a hybrid CPU-GPU can maximize the hardware combination efficiency with optimized task scheduling [22]. Tim et al. [10] and He et al. [8] revisited relational joins on GPUs taking advantage of emerging new technologies and achieved significant performance improvements. Peters et al. proposed a hybrid soring algorithm resulting in a high performance comparison-based sort for GPUs [16]. In the literature, most studies on GPUs are focusing on performance improvements. In our work, we address both the performance and energy efficiency of query co-processing on embedded CPU-GPU architectures.

Table 1: Specifications of CARMA and a GPU workstation

	CARMA	GPU Workstation
CPU	NVIDIA Tegra 3 Quad-core ARM Cortex-A9 1.3 GHz peak power $\sim$ 9W	Intel Xeon 6-core E5-2620 2 GHz peak power $\sim$ 95W
GPU	NVIDIA Quadro 1000M 96-core peak power $\sim$ 45W	NVIDIA Tesla K40C 2880-core peak power $\sim$ 245W
Memory	2GB	16GB
Storage	4GB eMMC	256GB SSD
PCI	4x PCIe Gen 1	16x PCIe Gen 3
Idle power	$\sim$ 10W	$\sim$ 80W
Peak power	$\sim$ 50W	$\sim$ 400W

There are many emerging embedded CPU-GPU devices. Most of them share a common architecture in which the GPU is connected to the CPU through the PCIe interface. We select the CARMA board in this study. We compare specifications of CARMA and a GPU workstation in Table 1. The embedded CPU and GPU on CARMA have much lower power than those in this GPU workstation. The CARMA board also features smaller memories, storage and other components. As a result, the CARMA board has a much lower idle and peak power than the workstation at the machine level.

3. IMPLEMENTATIONS

In this section, we give an overview of our query engine followed by considerations and details of the implementation.

3.1 Overview

Our query engine is based on the same layered design as that in GDB, a GPU-based query co-processor [7]. Its design on primitives and query operators is for exploiting the data parallelism of query co-processing. Figure 1 illustrates the four layers in the query engine: storage, data-parallel primitives, access methods and operators. At the storage layer, we use the column-store layout for relations. A set of commonly used primitives are defined and implemented in a data-parallel way. The engine further supports table scan and hash index based access. We adopt the same design from GDB [7] for most primitives, access methods and operators. We have also improved some of the implementations by porting the OpenCL implementation of OmniDB [23] and adopted implementations of main-memory hash joins in previous studies with modifications [2, 9].

In the following, we briefly describe our implementations of operators and the CPU-GPU query co-processing scheme

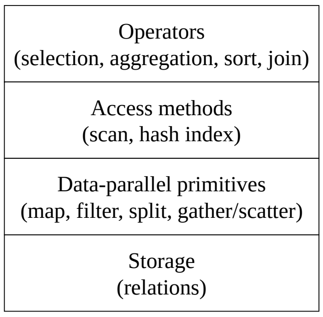


Figure 1: The layered design of the query engine (adopted from GDB [7])

including details specific to embedded CPU-GPU architectures.

3.2 Query Operators

We select four query operators as micro-benchmarks for our study. On both the CPU and the GPU, the implementations are with multi-threading parallelism, in Pthreads and CUDA, respectively.

Selection and aggregation. Both selection and aggregation perform sequential scans on input relations when using scan as the memory access method by default. When using the hash indexes for memory accesses, they perform sequential scans inside each hash bucket. For a CPU thread, this is implemented as a streaming access to the memory using hardware and software prefetching. For all threads within a GPU thread group, their accesses to GPU's memory are coalesced to utilize GPU's high memory bandwidth.

Sort. We select quick-sort as our sort algorithm as it only requires $O(logN)$ auxiliary space where N is the cardinality of the input relation. This relatively small memory footprint is important for embedded devices because of the limited memory capacity on both the CPU and the GPU.

Hash Join. A hash join algorithm works on two input relations, R and S, where R is the build relation and S is the probe relation. We adopt the partitioned hash join because it allows each partition to fit into either caches of CPUs or shared memories of GPU thread groups avoiding random memory access when joining each pair of partitions [10].

3.3 CPU-GPU Co-processing

We support a fine-grained query co-processing at the operator level. However, this co-processing in our current work is still simple and premature. More advanced query co-processing techniques will be investigated in the future. For each query operator, we partition input relations in a way that a certain percentage of the workload is distributed to the GPU and the rest is distributed to the CPU. This percentage is denoted as σ hereafter. Specifically, the query engine offers three execution approaches for each query operator:

- *CPU-only* (i.e., $\sigma = 0$): Only cores of the embedded CPU are used and the GPU remains idle.
- *GPU-only* (i.e., $\sigma = 100\%$): Only the GPU is used while CPU cores remain idle (a single CPU core is temporarily used to feed data to GPU and launch GPU threads).
- *CPU-GPU co-processing* (i.e., $0 < \sigma < 100\%$): All CPU and GPU cores are used. The workload distribution between the CPU and the GPU is decided by the σ value.

For selection and sort, the final result is achieved by concatenating partial results from the GPU and the CPU. For sort, a final N-way merge operation is executed on the CPU. For hash join, partial results from each CPU thread or GPU thread group can be directly linked together because of the partition based algorithm.

In the experiments, we configure the σ value from 0 to 1 and find the specific σ values for each query operator achieving the best performance and lowest energy consumption. The relative position of this best σ configuration in the range $[0, 1]$ shows the relative advantage of either the CPU or the GPU in terms of performance and energy consumption for each query operator. If the best σ configuration is closer to 0, it means that the CPU outperforms the GPU for this operator. The GPU outperforms the CPU otherwise.

3.4 Architecture Aware Implementations

Embedded CPU-GPU architectures have different features compared with the CPU-GPU architectures in servers or workstations. In this work, we port and tune the existing GPU codes [4, 7, 23] and CPU codes [2] to make them compatible with the embedded environment and fully utilize embedded processors.

In our previous study of main-memory hash joins on many-core processors [9], we added materializations and architecture-aware tuning into their implementations using architecture specific SIMD intrinsics which are not compatible with ARM CPUs. In this work, we modify the code to replace all architecture specific parts with C++ implementations compatible with the embedded CPU. For example, SIMD-based writes to memories are replaced with direct "memcpy" operations and SIMD gather & scatter intrinsics are replaced with batches of random memory accesses. For selection, aggregation and sort, we base our implementations on the Standard Template Library (STL) of C++ and each CPU thread takes an equal share of the input. We discuss future studies on optimizations for the embedded processors in Section 5.3.

Embedded GPUs have relatively fewer co-processing cores with smaller shared memories. To fully utilize this embedded GPU, we tune the scale of thread groups to maximize the utilization of GPU cores and the memory hierarchy.

We adopt the intermediate results management from our previous work [9] and add materializations of final results to our query engine. We use a late materialization strategy because it eliminates many unnecessary memory operations.

4. EVALUATIONS

In this section, we present experimental results on a single CARMA board. We discuss the extension to multiple boards in Section 5.

4.1 Setup

Our experiments are performed on a CARMA node as introduced in Table 1. Input relations are pre-loaded to the main memory before the execution of query algorithms. Programs and required libraries are cross-compiled using the CUDA toolkit 5.0 with the "-O3" optimization option in a 64-bit virtual machine running Ubuntu 12.04 LTS with kernel version 3.8.0-31. Managed by the Tegra "cpuidle" driver, power states of unused CPU cores are adaptively adjusted (e.g., in the GPU-only approach, three CPU cores are switched to the low-power "offline" state) and they are waken up to become "online" when workloads are scheduled to execute on them. The GPU remains idle when no GPU kernel is called. We set the CPU scaling mode to "performance" forcing all "online" CPU cores to run at their maximum frequency.

Workloads. Each record in input relations contains two 8-byte integer attributes, namely the key and the record ID. By default, key values are randomly generated. For all algorithms except hash join, we fix the relation size to 50 million records which is the maximum size for main-memory query co-processing on CARMA. Two relations with the same size of 5 million records are used as the input relation R and S for hash join. These settings of relation sizes are large enough to represent common data sets in embedded applications. The selectivity is 100 percent for selection and the hash join. We further select two queries (i.e., Q9 and Q14) from the TPC-H benchmark with different complexities to evaluate our CPU-GPU co-processing query engine. Q9 contains multiple complex join operators and a sort. Q14 mainly requires a single join. TPC-H query experiments are conducted based on data generated from the TPC-H data generator. Because of the limited memory capacity available on both the CPU and the GPU on the CARMA board, we set the scale factor to only 0.5, meaning that the total size of all the tables are approximately 500MiB.

Measurements and Comparisons. We use a power meter, *"Watts up? Pro"*, to measure the real-time power readings of the entire machine during the execution and sum them up to get the total energy consumption in joules. For all operators, we compare the execution time and total energy consumption by varying the σ values as introduced in Section 3.3. Specifically, we show the *normalized* execution time and energy consumption by normalizing measurements to that of the CPU-only approach.

4.2 Results on Operators

In this section we report and analyze the execution time and the total energy consumption of each query operator.

4.2.1 Selection and Sum

Selection performs simple comparisons after relation scans. Thus, it is a memory-intensive operation. In Figure 2a, both the execution time and energy consumption are increasing when the σ value rises. The CPU-only approach is the best and it achieves 80% better performance with 82% less energy consumption than the GPU-only approach. This significant difference is caused by the memory transfer bottleneck through the PCIe. Breaking down the execution time of the GPU for all σ values, we find that 98% of the total execution is spent in the PCIe memory transfer and the rest 2% accounts for the actual selection on the GPU.

In Figure 2b, as the σ value rises, both the execution time and energy consumption of sum increases. The CPU-only approach is also the best and it achieves 68% better performance with 70% less energy consumption than the GPU-only approach. For sum, the memory transfer on the PCIe takes 99% of the total time in average and thus it bounds the overall performance.

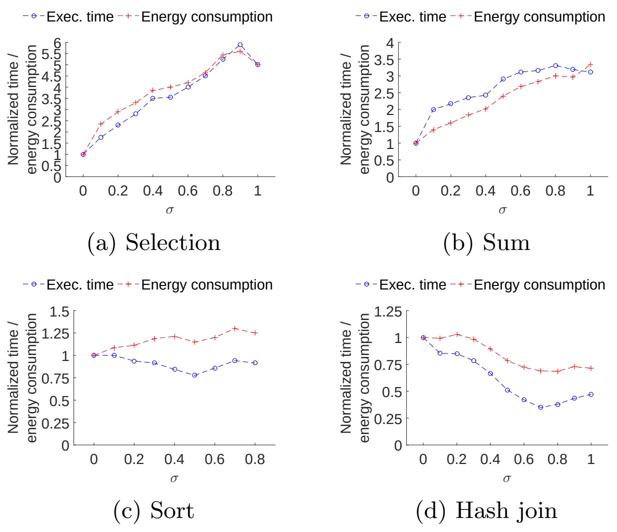


Figure 2: Normalized execution time and energy consumption of four query operators (the σ value ends at 0.8 for sort because of GPU's limited memory capacity)

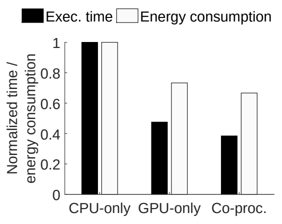
4.2.2 Sort

Because of the limited memory capacity on GPU, we end the σ value at 0.8 instead of 1 for sort. Figure 2c shows that the shortest execution time is achieved when $\sigma = 0.5$ and the CPU-only approach achieves the lowest energy consumption. Specifically, the CPU-only approach is 9% slower with 21% less energy consumption than the GPU-only approach. Different from selection and sum, the memory intensive part of sort (sort conducts more than one pass over input relations) happens on the GPU only and does not require additional PCIe transfers after the input relations are transfered to the GPU. This is why the PCIe transfer overhead does not dominate the performance of sort on the GPU. Another reason for this stable execution time and energy consumption for all σ values is that a final N-way merge accounts for about 20% of the total execution time and this significant part is processed by the low power CPU alone which is more energy-efficient for this memory intensive operation as observed in above experiments.

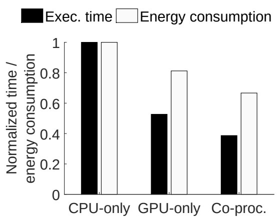
4.2.3 Hash Join

In Figure 2d, the execution time and energy consumption of hash join is reported. When the σ value increases, the execution time keeps decreasing indicating that the GPU is more efficient than the CPU until the σ value reaches 0.7. When the σ value is larger than this break-even point, the CPU takes insufficient workload required to mitigate the PCIe data transfer overhead. The GPU-only approach is about 2.1 times faster than the CPU-only approach with about 29% less energy consumption. The best execution time and the lowest energy consumption is achieved when $\sigma = 0.7$.

We observe an interesting performance-energy consumption trend that the energy consumption is more stable than the execution time. The slowest execution is about 2.8 times slower than the fastest one with only about 49% more energy consumption. When the σ value increases, the GPU takes over more workload from the CPU which reduces the overall



(a) Q9



(b) Q14

Figure 3: Normalized execution time and energy consumption of TPC-H Q9 and Q14

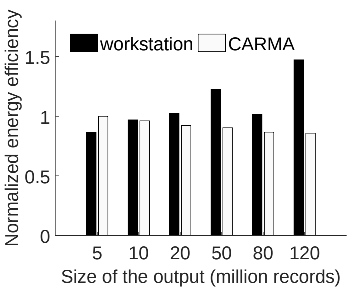


Figure 4: Normalized energy efficiency of hash join on the workstation and the CARMA board

execution time greatly. However, this also makes the power hungry GPU become busier resulting in limited reduction on the overall energy consumption.

4.3 Evaluation of TPC-H Queries

In Figure 3, we present results achieved in executing TPC-H Q9 and Q14 in three approaches by normalizing them to that of the CPU-only approach. The GPU-only approach reduces the overall execution time by about 50% for both queries, with 27% and 19% reductions in energy consumption than the CPU-only approach for Q9 and Q14, respectively. Our tuned co-processing approach with the best σ configuration further reduces the execution time by 9% and 14% with 7% and 15% less energy consumption for Q9 and Q14, respectively. Thus, the CPU-GPU co-processing delivers the best performance for both these two join based queries. These results are in line with our previous observations that the GPU is a better choice than the CPU for sort and hash join.

5. DISCUSSION

In this section we discuss some interesting issues and future directions.

5.1 Comparison with a Workstation

As a sanity check, we have compared the energy efficiency of the CARMA board and the workstation listed in Table 1. The energy efficiency refers to the number of result tuples can be produced per joule. Thus, it is calculated as the cardinality of the output divided by the total energy consumption. To make this a fair comparison, we run the same hash join code on both platforms deployed on all CPU and GPU cores and vary the cardinality of relations R and S from 5 million to 120 million. The selectivity is set to 100%, and the size of outputs equals to the size of either relation R or S. Due to the limited memory capacity, the CARMA board cannot process hash joins in a single run when the size of two input relations are larger than 5 million. Thus, we partition the input relations first and execute multiple runs of hash join on each pair of partitions to process large inputs on the CARMA board.

For each input size on both the workstation and the CARMA board, we tune the σ configuration experimentally and report the highest energy efficiency achieved. Energy efficiencies are normalized by that achieved on the CARMA board when the output size is 5 million in Figure 4. We can see that the CARMA board achieves 15% higher energy efficiency than the workstation when the size is 5 million. However, the workstation achieves higher energy efficiency than the CARMA board for relations larger than 20 million records. This confirms that embedded devices are more energy-efficient than workstations when co-processing small inputs and high-end workstations outperform for large inputs.

5.2 Extending to Cluster Settings

Although the embedded device is theoretically more energy-efficient for a range of small input relations, more devices are needed to process larger relations at the same level of energy efficiency. This calls upon a cluster of multiple embedded devices to work in a distributed manner. Dumitrel et al. proposed a similar cluster based on ARM big.LITTLE CPUs [3]. Although we do not have sufficient hardware resources due to budget constraints and we leave this as our future work, we discuss several necessary components for an energy-efficient cluster of this kind and also for future embedded architectures: network, PCIe and storage.

Network. Network is essential for any cluster and software optimizations for the network interface in such embedded devices is necessary. On the CARMA board, we evaluate the potential of this optimization by cross-compiling and testing the UDT high performance network transfer library [5]. The network port available on CARMA is 1Gbps Ethernet, with peak theoretical transfer rate of 128 MB/s. A single UDT connection utilizing only 1 CPU core with default settings can only maintain a speed of 29.8 MB/s on average. We then tune the network packet size in UDT transfer, add one more CPU core to write network buffers into memory and enable the Ethernet Jumbo Frames. The speed of a single connection rises to 41.2 MB/s and the aggregated bandwidth rises to 83.4 MB/s on the two CPU cores. Further increasing the number of CPU cores from two to three leads to underutilization of these cores and does not increase the network bandwidth.

Storages. In previous experiments, we exclude the access to the storage to focus on the main-memory co-processing. To study the impact of the storage on the overall performance as well as the total energy consumption, we incorporate the storage access into sort using an eMMC and a SSD on the CARMA board. Both the eMMC and the SSD in use are based on flash memory which is low in power consumptions. Figure 5 shows the breakdown of the total energy consumption of sort on CARMA. When using the eMMC, storage access accounts for more than 50% of the total execution time and bounds the execution. After replacing the eMMC with an SSD, this storage overhead is reduced to about 20%. As a result, SSD enabled sort increases the

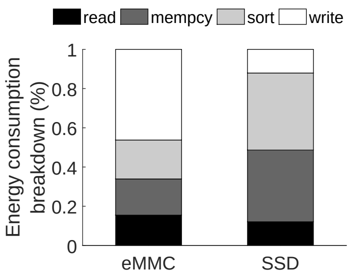


Figure 5: Energy consumption breakdown of sort on CARMA using eMMC and SSD

energy efficiency for the CPU-only approach and the GPU-only approach of sort by 7% and 25%, respectively. This demonstrates that SSDs can potentially help embedded systems to increase the overall energy efficiency especially when frequent read and write I/O operations are involved.

5.3 Future Work

SIMD (Single Instruction Multiple Data) instruction set has been widely used in multi-core CPUs. Many applications such as video encoding/decoding and image rendering have been taking advantage of SIMD to deliver high performance. In our past work [9], we have evaluated the performance impact of SIMD on main-memory hash joins and have obtained significant speedup. ARM NEON is a SIMD technology designed for some of the Cortex-A series CPUs. It supports up to 128-bit SIMD intrinsics. It is our future work to study the strategy for SIMD optimizations and its impact on embedded platforms in terms of both performance and energy consumption.

As observed in Section 4, memory transfer is a major performance bottleneck for the GPU. New memory transfer technologies have been proposed and are about to be released such as the NVIDIA NVLink technology that can link two GPUs or a CPU and a GPU directly with a higher throughput and energy efficiency than conventional PCIe buses.

The σ setting in this paper is only at the operator level. In complex queries, a more fine-grained approach may be required to fully harness the power of CPU-GPU co-processing.

6. CONCLUSION

Energy-efficient query co-processing in databases on embedded architectures requires novel hardware and software co-design. We take the opportunity enabled by the evolution of general-purpose GPUs of new-generation embedded devices to study the impact of CPU-GPU co-processing on such architectures from both the performance and energy consumption perspectives. By studying query operators as micro-benchmarks, we find that the CPU is more energy-efficient than the GPU when co-processing simple operators such as selection and aggregation. However, the GPU outperforms the CPU for sort and hash join. With proper tuning and optimizations, the CPU-GPU query co-processing scheme can be an effective means for energy-efficient query co-processing in embedded systems. We have also discussed a number of opportunities for query co-processing on future embedded CPU-GPU architectures.

Acknowledgment

We would like to thank NVIDIA for the hardware donation. This work is supported by the National Research Foundation, Prime Ministers Office, Singapore under its IDM Futures Funding Initiative and administered by the Interactive and Digital Media Programme Office (Grant No.: MDA/IDM/2012/8/8-2 VOL 01), Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly (LILY), Interdisciplinary Graduate School (IGS) of NTU and a MoE AcRF Tier 2 grant (MOE2012-T2-2-067) in Singapore.

7. REFERENCES

[1] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating Systems Principles*, pages 1–14. ACM, 2009.

[2] C. Balkesen, G. Alonso, J. Teubner, and M. T. Ozsu. Multi-core, main-memory joins: Sort vs. hash revisited. *Proceedings of the VLDB Endowment*, 7(1):85–96, 2013.

[3] B. M. T. Dumitrel Loghin, H. Zhang, B. C. Ooi, and Y. M. Teo. A performance study of big data on small nodes. *Proceedings of the VLDB Endowment*, 8(7), 2015.

[4] R. Fang, B. He, M. Lu, K. Yang, N. K. Govindaraju, Q. Luo, and P. V. Sander. Gpuqp: query co-processing using graphics processors. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1061–1063. ACM, 2007.

[5] Y. Gu and R. Grossman. Udtv4: Improvements in performance and usability. In *Networks for Grid Applications*, pages 9–23. Springer, 2009.

[6] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[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 (TODS)*, 34(4):21, 2009.

[8] J. He, M. Lu, and B. He. Revisiting co-processing for hash joins on the coupled cpu-gpu architecture. *Proceedings of the VLDB Endowment*, 6(10):889–900, 2013.

[9] S. Jha, B. He, M. Lu, X. Cheng, and P. H. Huynh. Improving main memory hash joins on intel xeon phi processors: An experimental approach. *Proceedings of the VLDB Endowment*, 8(6), 2015.

[10] T. Kaldewey, G. Lohman, R. Mueller, and P. Volk. Gpu join processing revisited. In *Proceedings of the Eighth International Workshop on Data Management on New Hardware*, pages 55–62. ACM, 2012.

[11] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis. Towards energy-efficient database cluster design. *Proceedings of the VLDB Endowment*, 5(11):1684–1695, 2012.

[12] K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang. Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *Parallel*

Processing (ICPP), 2012 41st International Conference on, pages 48–57. IEEE, 2012.

[13] F. Mantovani. High performance computing based on embedded processors. In *High Performance Computing & Simulation (HPCS), 2014 International Conference on*, pages 1034–1034. IEEE, 2014.

[14] T. Mühlbauer, W. Rödiger, R. Seilbeck, A. Reiser, A. Kemper, and T. Neumann. One dbms for all: the brawny few and the wimpy crowd. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 697–700. ACM, 2014.

[15] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated cpu-gpu power management for 3d mobile games. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6. IEEE, 2014.

[16] H. Peters, O. Schulz-Hildebrandt, and N. Luttenberger. A novel sorting algorithm for many-core architectures based on adaptive bitonic sort. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 227–237. IEEE, 2012.

[17] D. Schall and T. Härder. Energy-proportional query execution using a cluster of wimpy nodes. In *Proceedings of the Ninth International Workshop on Data Management on New Hardware*, page 1. ACM, 2013.

[18] D. Schall and V. Hudlet. Wattdb: an energy-proportional cluster of wimpy nodes. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1229–1232. ACM, 2011.

[19] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 231–242. ACM, 2010.

[20] V. Vasudevan, L. Tan, M. Kaminsky, M. A. Kozuch, D. Andersen, and P. Pillai. Fawnsort: Energy-efficient sorting of 10gb. *Sort Benchmark final*, 2010.

[21] D. Wong and M. Annavaram. Knightshift: scaling the energy proportionality wall through server-level heterogeneity. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, pages 119–130. IEEE, 2012.

[22] Y. Yuan, R. Lee, and X. Zhang. The yin and yang of processing data warehousing queries on gpu devices. *Proceedings of the VLDB Endowment*, 6(10):817–828, 2013.

[23] S. Zhang, J. He, B. He, and M. Lu. Omnidb: Towards portable and efficient query processing on parallel cpu/gpu architectures. *Proceedings of the VLDB Endowment*, 6(12):1374–1377, 2013.