Unleashing the Hidden Power of Integrated-GPUs for Database Co-Processing

Edward Ching, Norbert Egi, Masood Mortazavi, Vincent Cheung, Guangyu Shi

FutureWei Technologies

Abstract

Modern high-performance server systems with their wide variety of compute resources (i.e. multi-core CPUs, GPUs, FPGAs, etc.) bring vast computing power to the fingertips of researchers and developers. To investigate modern CPU+GPU co-processing architectures and to discover their relative marginal return on resources ("bang for the buck"), we compare the different architectures with main focus on database query processing, provide performance optimization methods and analyze their performance using practical benchmarking approaches. Discrete GPU devices (d-GPUs) are connected to host memory through PCIe which is relatively slower than the internal bus which connects integrated GPU subsystems (i-GPUs) to the CPU and its caches. Our experimental analysis indicates that while massive processing capabilities have grown in d-GPUs, data locality, memory access patterns and I/O bottlenecks continue to govern and impede the speed of database processing functions. While the use of GPU systems (whether discrete or integrated) lead to significant speed-up in primitive data processing operations, the marginal performance returns on deploying teraflops d-GPU compute resources do not appear to be large enough to justify their cost or power consumption. At this point, integrated GPUs provide better cost-performance and energy-performance results in parallel database query processing.

1 Introduction

Discrete Graphics Processing Units (d-GPUs) have long been used for accelerating the execution of a wide-range of applications, most generally known in the space of scientific high-performance computing (HPC) [WGHP11, GS11]. In recent years, CPU+GPU co-processing have widely been adopted to improve processing performance of data analytic applications handling very large data sets using main memory database management systems [WZHC10, HLH13, HLY+09, PMK11, KLMV12, HSP+13, Bøg13]. While these GPUs posses an enormous amount of compute power, they typically connect to the CPU and the host memory through a relatively slow interconnect, such as PCI-Express (PCIe). Compared to HPC, database query processing requires less computation but on significantly more data where the performance of the interconnect between the CPU, host memory and the GPU has a more significant effect on the application's performance. However, recent CPUs with integrated GPUs (i-GPU) enabled for general-purpose computing provide an alternative architecture. While the computational power of the GPU cores is lower than that of d-GPUs, they are connected to the CPU and its caches through an internal bus

which is significantly faster than PCIe.

Earlier studies in parallel query processing generally focused on two kinds of systems: multi-core CPUs or many-core d-GPUs accessible through a PCI-e connection. Most studies on d-GPU co-processing have indicated PCI-e bandwidth issues as the main bottleneck in off-loading parallel computation to these co-processors [HLY+09, KLMV12, PMK11]. Micro-benchmarks and our own independent investigations in various settings have also confirmed this issue. To resolve this issue, Ocelot uses a memory management facility to transparently "pin" data arrays on d-GPU memory and to dynamically control and optimize the eviction of these data arrays. More recent work has paid greater attention to the use of i-GPU-based, fine-grained co-processing [HLH13, TK13] and have concluded that i-GPUs are promising alternatives to d-GPUs for data-intensive applications and functionalities, such as the JOIN operation. The work we report here is primarily experimental and focuses on the latter approach, i.e. on the use of modern i-GPUs for parallel data processing. We have selected benchmark queries and a set of primitive operations (essentially, micro-benchmarks) to compare the performance of the current, state-of-the-art systems. Our findings indicate a balancing trade-off (in gigaflops, I/O bandwidth and power consumption) that favors modern i-GPUs in query processing.

We focus on the discovery of system profiles and trade-offs that can help guide system designers to select among various architectural options for parallel processing of data analytic queries. We used modern multi-core, i-GPU and d-GPU systems and evaluated each system through the execution of analytic micro-benchmarks and a more complex TPC-H query. We target each system with implementations that are best suited to their architectural requirements. We found that high-gigaflops d-GPU computing systems may be an overkill when it comes to query processing and that performance gains for d-GPU compared to modern i-GPU systems may not justify the energy and capital costs for the purposes of data analytic processing. Next, we will discuss the relevant features of system architectures we investigated in Section 2, present the performance results in Section 3 and conclude in Section 4.

2 Architecture

In this section we analyze the key architectural differences of i-GPUs compared to d-GPUs and highlight their "hidden" potential for database query acceleration to be exploited by applications. Our analysis is based on two recent and widely-used GPUs, that is, Nvidia's GTX780 d-GPU and Intel's HD4600 i-GPU integrated into the Haswell i7-4770K CPU. Figure 1 illustrates the co-processing architecture using d-GPUs, while Figure 2 shows the internals of the Intel CPU with integrated GPU. The most performance critical differences between the two co-processing architectures are (1) the performance of interconnect(s) between the GPU, host memory and the CPU's caches, (2) the cache structure and (3) the computational capacity of the GPUs.

The performance of the interconnect(s) between the GPU and the location of the data prior to (i.e. data input) and right after (i.e. data output) being processed by the GPU becomes highly critical for data-intensive workloads that are collectively processed by the CPU and GPU, in which case the data typically resides in the CPU's cache or in the host memory and has to be moved from there to the GPU and finally back after the GPU processing

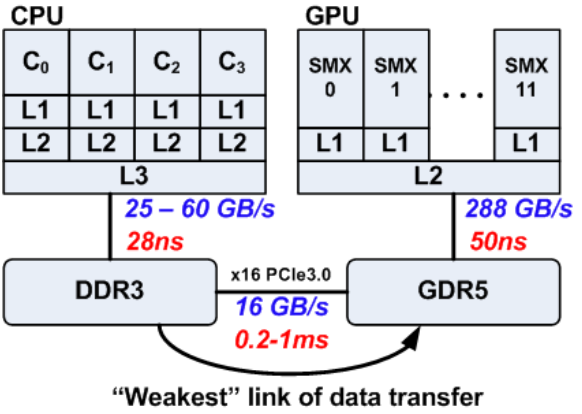


Figure 1: Discrete GPU architecture (NVidia GTX780)

has been completed. CPU+GPU co-processing of database queries demonstrate a data movement pattern very similar to that of above. After a query has been launched, a query execution plan is created and optimized for performance by the database management system (DBMS). The query execution plan determines the order the functionalities (i.e. query primitives) should be executed and the steps used to access data. Depending on the characteristics of functions in the execution plan, they provide better performance either on the CPU or GPU. Based on these characteristics and performance statistics of previous query executions, the optimizer decides which query primitives should be executed on the CPU and which ones on the GPU. The optimizer also aims for minimal amount of data transfer between the CPU and the GPU. However, even in the best case, data that is needed for processing at the GPU traverses over the interconnect between the CPU and GPU twice (once in each direction).

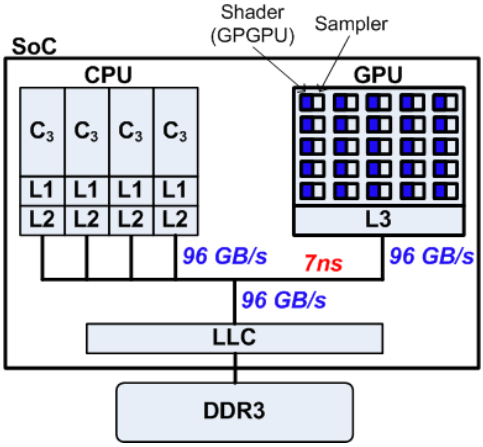


Figure 2: Integrated GPU architecture (Intel i7-4770K)

While d-GPUs connect to the CPU and host memory via a relatively slow (compared to memory speed) PCIe interconnect, i-GPUs connect to the CPU via a fast internal bus. In addition, i-GPUs have direct access to the Last-Level Cache (LLC) within the CPU die, providing even faster access to the data if it is present in the cache, a typical scenario with CPU+GPU co-processing. In our Intel i7-4770K chip the CPU and GPU shares the 8MB of LLC.

In addition to the speed of the interconnect between the CPU and GPU and whether the GPU has direct access to the CPU's cache, the GPU cores' computational capacity is a major contributing factor to applications' performance. In Table 1 we compare the specification of the above mentioned Nvidia GTX780 (d-GPU) and Intel HD4600 GPUs (i-GPU). The important parameters to note are, that the d-GPU outperforms the i-GPU by over a factor of 9x in terms of floating point operations. However, the i-GPU has nearly twice as many data lanes [1] it can run instructions on (i.e. maximum physical occupancy) and its clock speed is 25% higher than that of the d-GPU. However, the d-GPU has access to 3GB of local very fast GDR5 memory while the i-GPU's Last Layer Cache is only 8MB.[2]

	GTX780	**HD4600**
Cores	12	20
Threads/Core	6	7
Data lane/Thread	32	8 / 16 / 32
Max. physical occupancy	2304	4480[a]
Clock (GHz)	1.0	1.25
Power consumption (W)[b]	250	<30[c]
GFLOPS[d]	3977	432

Table 1: Hardware parameter comparison of Nvidia GTX780 d-GPU and Intel HD4600 i-GPU

[a] Using 32 data lanes per thread.
[b] As Thermal Design Power (TDP)
[c] Combined CPU and GPU equals to 84 Watts. [Des]
[d] Database processing typically requires little floating point operations, thus this metric does not directly reflect the real compute power ratio of the two architectures.

2.1 Data Transfer Mechanisms

In order to understand how the speed of interconnect affects the applications' performance, it is critical to understand the data transfer mechanisms supported by the different hardware architectures. In this section, we provide a brief overview of the key aspects of these mechanisms and quantify their effect on the performance of applications in Section 3.

Discrete GPUs support two types of data transfer mechanisms between the CPU and the GPU. On one hand, the more traditional model is using hardware supported Direct Memory Access (DMA) operations to transfer all the data to the GPU's memory (i.e. GDR5 in Figure 1) prior to processing it and DMA the data back after completion. While DMA op-

[1] Assuming SIMD32, the maximum number of data lanes we can have per thread.
[2] Next-generation integrated CPUs are expected to have 128MB of EDRAM (Embedded DRAM) on-chip as an additional layer of cache. [Tak]

erations are executed by hardware and do not need either the CPU's or GPU's intervention, it still happens over the relatively slow PCIe connection. As such, it can add to the total execution time of the application if the GPU cannot be utilized while the DMA operations take place.

On the other hand, a block of the main memory belonging to the user application on the CPU's side can be mapped into the GPU's virtual memory space via the GPU's Memory Management Unit (MMU). Programming the MMU takes only several cycles and thus is a much faster operation than DMA-ing the data between the main memory and the GPU's memory. Data mapped into the GPU's virtual memory space can be referenced directly and when it happens a Programmed I/O (PIO) operation is executed from the GPU's side, which moves the data from the main memory to the GPU's registers for processing. The advantage of this method is that only the part of the data that is processed is moved to the GPU, which might be significantly less than the total amount of application data. However, during a PIO transfer the GPU is stalled until the data transfer completes and because the transfer happens over the relatively slow PCIe connection this can slow the overall processing down except for suitable workloads where the GPU can overlap computation with kernel originated PCI-Express transactions.

In comparison, integrated GPUs, like that of the Intel i7-4770K, share the cache and main memory with the CPU and access memory through the GPU's own virtual memory space that is mapped by the GPU Address Translation Table (GTT) to the physical memory. This mechanism is similar to that of the memory mapped case of d-GPUs with the main exceptions that the data is fetched through a fast internal bus and it can also be retrieved from the shared LLC of the CPU, providing even faster access to the data. In this architecture, data buffers can be allocated by either the GPU or CPU and direct access can then be given to the other through memory address mapping.

3 Performance Analysis

In this section, we provide the results of our performance analysis study that quantifies the architectural differences between i-GPU and d-GPU based co-processing by using artificial and realistic workloads.

3.1 Raw data access

The objective of our first performance measurement was to analyze the speed of raw data access using the different data transfer mechanisms outlined in Section 2.1 (i.e. d-GPU DMA, d-GPU memory mapped, and i-GPU shared access). To this end, we created an artificial workload in OpenCL (i-GPU) and CUDA (d-GPU) where a fixed size array of integers was read through randomly. We selected random read over sequential to avoid prefetching that can hide some of the physical data access delays. Figure 3 shows the execution time of this workload based on the three different data transfer mechanisms.[3] Regardless of the size of the data structure, the i-GPU significantly outperforms both types of d-GPU based mechanisms by a factor of 2.2 to 44 thanks to the high bandwidth, low-latency internal bus and the lower latency of the DDR3 memory. The performance dif-

[3] Please note the logarithmic scaling.

ference compared to the DMA based mechanism varied between 2.2x and 2.6x, but the difference is close to constant regardless of the size of the data set [4], while the difference compared to the memory mapped mechanism is between 20x and 44x and is increasing with the data set size. As for the two d-GPU mechanisms, the DMA based execution provides a 14-fold (on average) performance difference over the memory-mapped execution, clearly demonstrating the drawbacks of memory mapping with d-GPUs in the case when multiple independent accesses are made to the main memory and the GPU cores are stalled while data is moved over the PCIe interconnect.

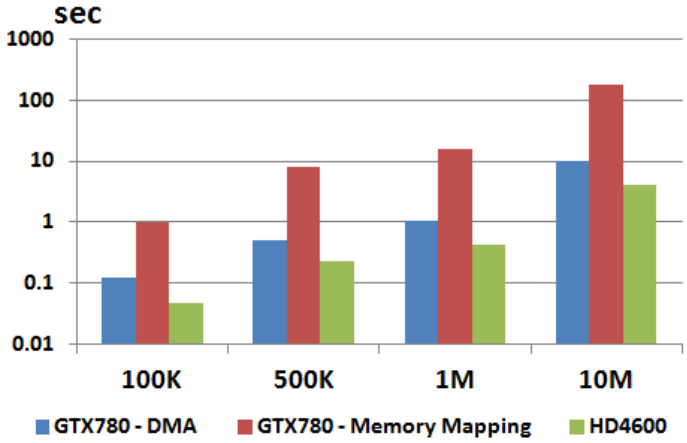


Figure 3: Performance (execution time) of different data transfer mechanisms

3.2 Micro-benchmarks

Previous studies have used database micro-benchmarks to evaluate the performance of database query co-processing [HLY+09]. We have implemented a selected set of these functionalities in OpenCL and CUDA and ran them on both the i-GPU (OpenCL only) and d-GPU platforms to understand their relative performance. Our selected micro-benchmarks represent workloads with:

- Simple optimal memory access patterns: Map, Reduce, Scan (a.k.a. prefix sum) [5];
- Randomized memory access pattern: Gather, Scatter;
- Combination of several sorting operations (i.e. randomized memory access, comparison, branching, limited integer arithmetic): Split, Bitonic and Radix Sort.

Figure 4 shows the execution time of the different micro-benchmarks with 32 Million entries on the i-GPU and d-GPU platforms. On the d-GPU platform we ran the benchmarks in OpenCL for a fairer comparison as well as in CUDA, which is highly optimized for Nvidia GPUs and thus provides the highest achievable performance. We used the DMA

[4] In the case of the DMA based mechanism the reported times include both the DMA transfer to the GPU and the random read of the array. The DMA transfer in all cases takes less than 0.1% of the reported time.

[5] The latter involves a large amount of data movement with some level of randomness though.

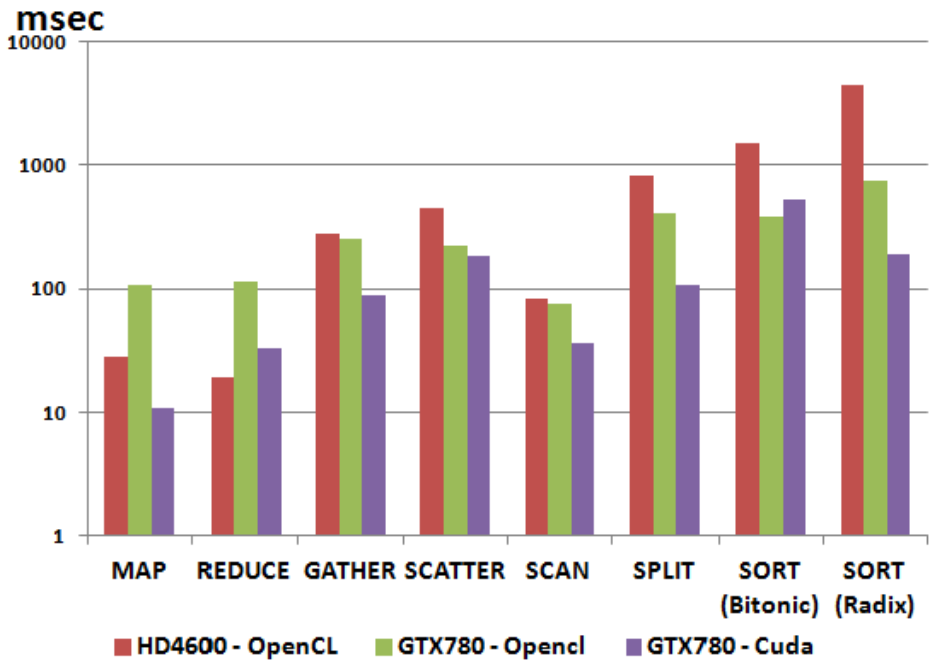


Figure 4: Micro-benchmark execution times

based data transfer mechanism with all the d-GPU measurements. Figure 5 shows the same results, but in terms of data throughput normalized to the power consumption of the GPUs. As far as raw performance is considered (i.e. Figure 4) the i-GPU platform provides better or near equal performance to the d-GPU platform with OpenCL for workloads that demonstrate randomized memory read and simple optimal memory access patterns to localized and coalesced data, which allows maximum leverage of shared memory and caches (i.e. map, reduce, gather and scan). The additional workloads require more computation and multiple iterations through the same data and thus favor the d-GPU platform more for its higher computational capability. For example, in the case of Radix Sort where the performance difference is the most significant in the favor of the d-GPU, the caching advantage of the i-GPU disappears due to the fact that many passes need to be made through a large set of data that does not fit into the cache. That is, the one-time transfer into the large memory unit of d-GPU is amortized over the multiple passes, while the i-GPU loses some of its advantage through frequent cache churns. However, if we take the power consumption of the GPUs into account, as shown in Figure 5, the i-GPU platform outperforms the d-GPU platform even with CUDA in all cases except with Radix Sort. Note again that CUDA uses libraries highly optimized for the Nvidia GPUs, while the OpenCL implementation does not have all of these highly optimized functions available, and hence the lower performance. Given that recent initiatives of new server design [Sea, HP , Dia] demonstrate the growing importance of power consumption in data center infrastructure, our results indicate that i-GPU based co-processing architectures have a high potential to become the

dominant database acceleration platform in the near future.

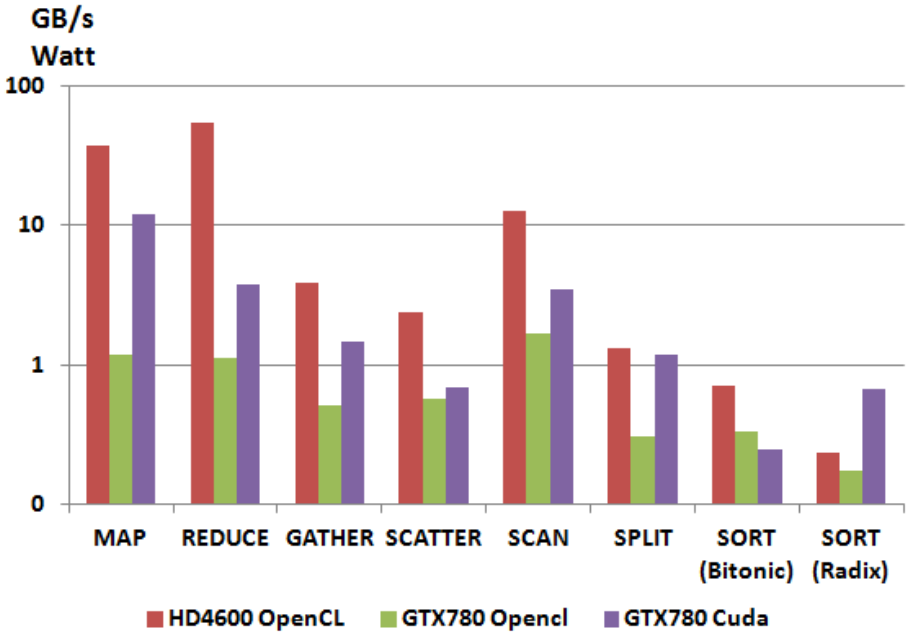


Figure 5: Micro-benchmark data processing throughput, normalized to power consumption

3.3 Database query

Our last set of performance analysis results were gathered by implementing and running Query-1 (Q1) and Query-9 (Q9) of the widely-used TPC-H decision support benchmark. We have implemented both TPC-H Q1 and Q9 as User Defined Functions (UDF). As far as TPC-H Q1 is concerned, we mofified a UDF for MonetDB suggested in [BZN05] by distributing the data set among workitems/threads on the targeted GPU to ensure full resource occupancy. As part of this query's execution plan, each workitem/thread performs sub-aggregation of the results followed by the CPU fully aggregating all the results. The data set was generated by setting the Scaling Factor (SF) to 1, which corresponds to slightly over 6 Million data records. We implemented the UDF in both OpenCL and CUDA. In terms of execution time and data throughput normalized to power consumption, Figures 6 and 7 show the similar trends as we observed earlier with micro-benchmarks. As shown by these results, the HD4600 i-GPU processes the data faster than the GTX780 d-GPU regardless of whether OpenCL or CUDA has been used as the programming environment for the d-GPU. Even more remarkably, if we consider the power consumption as the normalizing factor, the i-GPU outperforms the d-GPU platform by over a magnitude.

In addition to the results discussed above, we also ran TPC-H Query-1 in a hybrid manner in OpenCL, half of the data being processed by the CPU cores and the other half by the

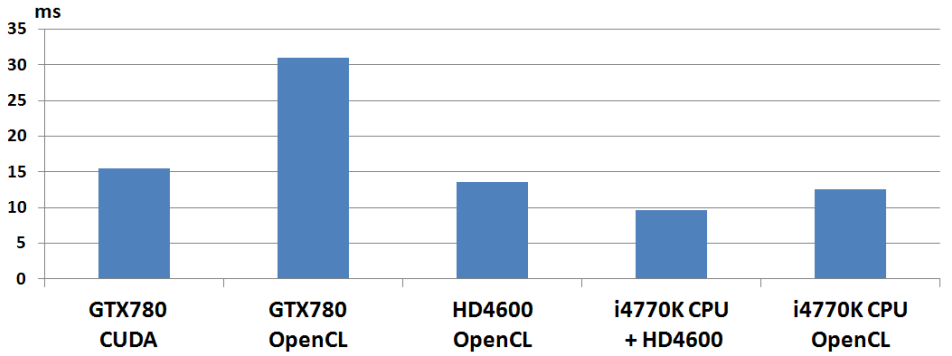


Figure 6: TPC-H Q1 (UDF) execution times

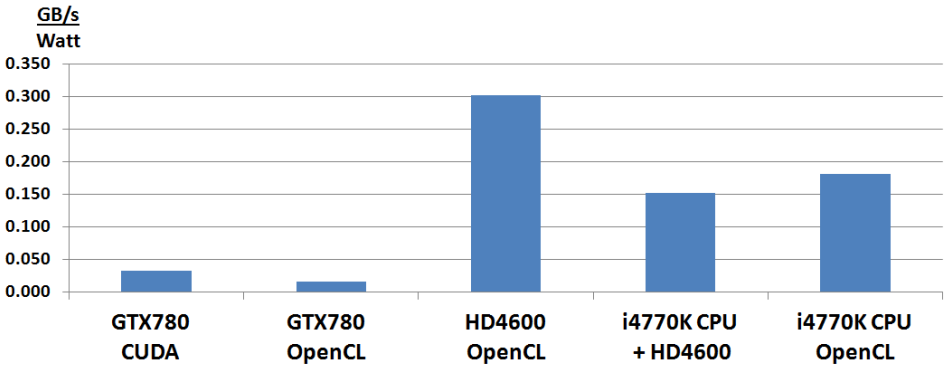
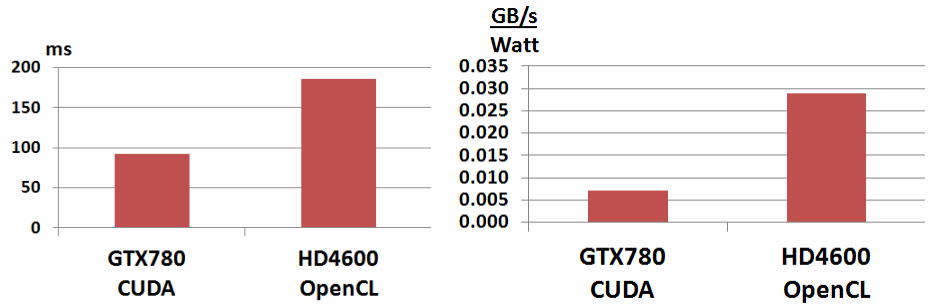


Figure 7: TPC-H Q1 (UDF) data processing throughput, normalized to power consumption

GPU cores, merging the results after completion of the query. As shown by the 4th bar in Figures 6 and 7, even more database processing power can be extracted from the i-GPU based co-processing platform using OpenCL and without increasing the complexity of the program at all. For comparison, we also included (as bar 5) the performance of the CPU-only execution of the TPC-H Q1 UDF which uses the program suggested in [BZN05] for targeting the CPU *only*.

As far as TPC-H Q9 is concerned, we implemented it as a UDF, similarly to Q1, using column based input. The key operations of this query include string matching, join, aggregation, group by and order by. As for string matching, we implemented a naïve linear string search algorithm and the Horspool algorithm [Hor80]. Our micro-benchmarks of this function indicated that the naïve algorithm outperforms the Horspool algorithm and its performance advantage increases with increasing record size. As such, we used the naïve approach in our final implementation. We performed aggregation and grouping using the

sort-merge join algorithm, and used the radix-sort algorithm for ordering. The hash join operation in our experiments took up almost half of the reported Q9 execution time. For hash join, we used an un-partitioned, bucket-chained hash join. It is possible that using a cache optimized partitioned hash join would improve the performance on our i-GPU platform although it has been reported that for off-load mode processing on i-GPU platforms, partitioned hash joins and un-partitioned hash joins show similar results [HLH13].



(a) Execution time

(b) Data processing throughput, normalized to power consumption

Figure 8: Performance comparison of TPC-H Q9 (UDF)

Figure 8 shows the results of our performance evaluation of the TPC-H Q9 UDF implementation, with Scaling Factor set to 1, using all the resources of both the Nvidia GTX780 d-GPU and the Intel HD4600 i-GPU. On the d-GPU platform we used CUDA to execute the query while on our i-GPU platform we used OpenCL. Looking at the performance from the execution time's perspective, the i-GPU based environment requires twice as much time to execute the query with the same number of records. However, in terms of data processing throughtput normalized to the power consumption, the i-GPU outperforms the d-GPU by four times, further strengthening our observation that i-GPUs provide a more power efficient acceleration environment for database processing compared to that of d-GPUs.

4 Conclusions

We have examined query and primitive operation processing on i-GPUs and d-GPUs of comparable performance attached to the system on PCIe 3.0. We used a series of varied micro-benchmarks and a more realistic data-analytic query from the TPC-H standard. We have found that while their compute resources are weaker, i-GPUs excel significantly in the speed of data access from the CPU. Furthermore, i-GPUs behave as "free" resources in some sense and consume far less power. Database management systems require CPUs to perform query parsing, plan generation, plan optimization and plan execution. They can also generate parallel execution plans targeted to multi-core CPU systems. The question of interest is whether dedicated parallel-processing subsystems can be used for off-loading such parallel query executions. Our work confirms earlier findings that such devices can be useful, but draws a further distinction between i-GPUs and d-GPUs, demonstrating that

i-GPUs have better marginal performance return on capital and operating costs. We have arrived at this conclusion through architectural analysis, micro-benchmarks using typical database query primitive operations and a typical complete query when each are optimally implemented for d-GPUs or i-GPUs. It may be worth exploring more complex queries (e.g., skyline queries) on larger data sets to examine whether this advantage of i-GPUs extends to a broader set of query processing scenarios.

References

[Bøg13] Kenneth Sejdenfaden Bøgh. Efficient Skycube Computation on the GPU. Master's thesis, Aarhus University, 2013.

[BZN05] Peter A. Boncz, Marcin Zukowski, and Niels Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *CIDR*, pages 225–237, 2005.

[Des] Desktop 4th Generation Intel Core Processor Family: Datasheet, Vol. 1. http://www.intel.com/content/www/us/en/processors/core/4th-gen-core-family-desktop-vol-1-datasheet.html.

[Dia] Diane Bryant. Architecting for Hyperscale Datacenter Efficiency. http://www.intel.com/newsroom/kits/atom/c2000/pdfs/ Architecting_ for_ Hyperscale_ DC_ Efficiency.pdf.

[GS11] Dominik Göddeke and Robert Strzodka. Cyclic Reduction Tridiagonal Solvers on GPUs Applied to Mixed Precision Multigrid. *IEEE Transactions on Parallel and Distributed Systems (TPDS), Special Issue: High Performance Computing with Accelerators*, 22(1):22–32, Jan 2011.

[HLH13] Jiong He, Mian Lu, and Bingsheng He. Revisiting Co-processing for Hash Joins on the Coupled CPU-GPU Architecture. *Proc. VLDB Endow.*, 6(10):889–900, August 2013.

[HLY+09] Bingsheng He, Mian Lu, Ke Yang, Rui Fang, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational Query Coprocessing on Graphics Processors. *ACM Trans. Database Syst.*, 34(4):21:1–21:39, December 2009.

[Hor80] Nigel R Horspool. Practical Fast Searching in Strings. In *Software - Practice & Experience*, volume 10, pages 501–506, 1980.

[HP] HP Moonshot System. http://www.hp.com/go/moonshot.

[HSP+13] Max Heimel, Michael Saecker, Holger Pirk, Stefan Manegold, and Volker Markl. Hardware-oblivious Parallelism for In-memory Column-stores. *Proc. VLDB Endow.*, 6(9):709–720, July 2013.

[KLMV12] Tim Kaldewey, Guy M. Lohman, Ren Mller, and Peter Benjamin Volk. GPU join processing revisited. In Shimin Chen and Stavros Harizopoulos, editors, *DaMoN*, pages 55–62. ACM, 2012.

[PMK11] Holger Pirk, Stefan Manegold, and Martin L. Kersten. Accelerating Foreign-Key Joins using Asymmetric Memory Channels. In Rajesh Bordawekar and Christian A. Lang, editors, *ADMS@VLDB*, pages 27–35, 2011.

[Sea] SeaMicro SM15000 Fabric Compute Systems and Freedom Fabric Storage. http://www.seamicro.com/products.

[Tak] Taking Advantage of Intel Graphics with OpenCL. http://download-software.intel.com/ sites/ default/ files/ managed/46/ba/ Webinar001_-_ Taking_ Advantage_of_ Intel_Graphics_ with_OpenCL.pdf.

[TK13] Dirk Habich Wolfgang Lehner Tomas Karnagel, Benjamin Schlegel. The HELLS-Join A Heterogeneous Stream join for ExtremeLy Large Windows. In *9th Ninth International Workshop on Data Management on New Hardware (DaMoN 2013)*, June 2013.

[WGHP11] Rick Weber, Akila Gothandaraman, Robert Hinde, and Gregory Peterson. Comparing Hardware Accelerators in Scientific Applications: A Case Study. *IEEE Trans. Parallel Distrib. Syst.*, 22(1):58–68, 2011.

[WZHC10] Ren Wu, Bin Zhang, Meichun Hsu, and Qiming Chen. GPU-Accelerated Predicate Evaluation on Column Store. In Lei Chen, Changjie Tang, Jun Yang, and Yunjun Gao, editors, *Web-Age Information Management*, volume 6184 of *Lecture Notes in Computer Science*, pages 570–581. Springer Berlin Heidelberg, 2010.