Query Processing on Gaming Consoles

Wei Cui, Qianxi Zhang, Spyros Blanas, Jesus Camacho-Rodriguez, Brandon Haynes,
Yinan Li, Ravi Ramamurthy, Peng Cheng, Rathijit Sen, Matteo Interlandi
Microsoft
weicu@microsoft.com,{firstname.lastname}@microsoft.com

ACM Reference Format:
Wei Cui, Qianxi Zhang, Spyros Blanas, Jesus Camacho-Rodriguez, Brandon Haynes,, Yinan Li, Ravi Ramamurthy, Peng Cheng, Rathijit Sen, Matteo Interlandi. 2023. Query Processing on Gaming Consoles. In *19th International Workshop on Data Management on New Hardware (DaMoN '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3592980.3595313

1 INTRODUCTION

The impending end of Moore's Law has generated a surge of interest within the database community in exploring alternative hardware solutions for efficiently handling database analytical workloads [5]. This is driven by the realization that traditional CPU performance is no longer scaling as it used to, making it necessary to investigate other hardware options. Our community has identified this opportunity, and over the years we have seen substantial work both in improving CPU execution using newly available hardware instructions [11, 24] or compilation [16], and exploring new hardware such as GPUs [19], FPGAs [10], or other accelerators [9, 22].

In this short paper, we add a new chapter in the quest of searching for the Database Machine [8] and show some early results obtained by running selected TPC-H queries on an Xbox console. While using an Xbox to execute SQL queries might initially seem like a bizarre choice, Xbox has an interesting hardware configuration since it is equipped with an AMD chiplet with integrated CPU and GPU. This means an Xbox has a memory bandwidth of around 560GB/s—about 2-4× what is currently obtainable from a CPU server, or about 25% less than a NVIDIA P100 discrete GPU (see Table 1). Furthermore, as far as we know, Xbox is one of the few devices with an integrated CPU and GPU that is already deployed at cloud scale thanks to the Xbox Cloud Gaming offering [4].

Until recently, chiplet devices with an integrated CPU and GPU were mostly confined into lower-end laptops. In recent years, however, we have been witnessing a renewed interest in high-end chiplet devices such as Apple Silicon [1], AMD MI300 [20], and Xbox [3]. This new wave of chiplet devices offer similar characteristic as GPUs, but without having to transmit data through the PCIe bus. Quite surprisingly, only a handful of works [12, 17] recognized this opportunity early on. Our hope with this paper is to increase awareness of this new hardware trend. To show the capabilities that these chiplet devices can attain, we have implemented an extension of TQP [13] to run SQL queries on Xbox with performance that is comparable to an NVIDIA P100 GPU, but without the overhead of moving data from CPU to GPU memory. For the selected queries we picked from TPCH, we can achieve almost up to 20× improvement over state-of-the-art CPU-only open-source solutions (DuckDB), while being only marginally slower than a P100. As far as we know, this paper is the first showing that is possible to run SQL queries on a gaming console, with performance that is orders of magnitude better than CPU-only solutions.

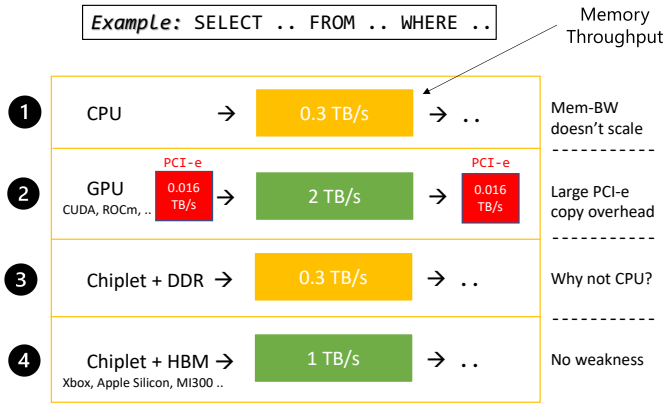

Figure 1: Executing a SQL query on different hardware backends with different memory bandwidth properties.

Summarizing, the contributions of this paper are: (1) identifying a set of limitations of query processing over discrete CPUs and GPUs, while highlighting chiplets with integrated CPU and GPU as interesting devices for analytical workloads (Section 2); (2) showing experimentally the performance of integrated CPU and GPU devices by running a few selected TPCH queries on Xbox, and comparing against a discrete GPU (NVIDIA P100) and CPU baselines (Section 3).

2 CPU VS GPU: A QUALITATIVE ANALYSIS

In this Section we are going to provide a qualitative assessment on Query Processing (QP) of analytical queries on CPUs and GPUs. We will also discuss the strengths and limitations of these hardware options when it comes to QP. Figure 1 summarizes our investigation. Let us assume we have a SQL query that is memory bound (as is typically the case for analytical queries). When executing the query on a CPU (❶), ideally we can expect to achieve a throughput of about 300GB/s—the maximum throughput currently achievable by the fastest DDR memory [21]. Next, processing on GPUs (❷) brings several advantages compared to CPUs.

- *High memory bandwidth*: The availability of High Bandwidth Memory (HBM) [2] on a GPU offers an aggregate bandwidth in

DaMoN '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 979-8-4007-0191-7/23/06…$15.00
https://doi.org/10.1145/3592980.3595313

the order of TBs/sec (about one order of magnitude higher than the CPU case), tremendously benefit analytical queries. While recent CPUs [14] have started to incorporate HBM in their designs, their lower core counts and memory-level parallelism can result in a lower utilization of the available bandwidth.

- *Massive parallelism*: While high-end CPU servers can have more than a hundred cores [6, 7], GPUs are equipped with at least an order of magnitude more, that is, thousands of (simpler) CUDA cores (NVIDIA) or Stream Processors (AMD). Moreover, GPUs can efficiently handle hundreds of thousands of concurrent thread contexts. This architecture supporting massive parallelism is a good fit for executing well-parallelizable analytics operations.
- *User-controlled coherence and synchronization*: GPUs offer more control than CPUs to users for data placement, and data coherence is explicitly controlled by the user program. This eliminates the hardware complexity that would otherwise be required to guarantee coherence through transparent mechanisms as on CPUs, and can be efficiently handled for analytics queries.

Because of the above characteristics, GPUs outperform CPUs for data analytics workloads if the data fits in the (HBM) device memory [19]. However, data is often fetched from higher-capacity locations, such as CPU memory, local or remote storage. This data copy operation requires to pass through the PCI-e bus, with limited bandwidth both compared to the device and the host memory.

We next focus on low-end CPU/GPU chiplet devices (❸). These devices have the same memory bandwidth as CPUs, therefore they suffer from the same limitations as CPU-only QP. Indeed GPUs can be leveraged here to improve computation peformance, but in general the memory bandwidth accessed by these GPUs are low and will limit the data processing throughput of QP. These limitations however disappear in high-end CPU/GPU chiplets devices (❹) such as Apple M1, AMD MI300A, and Xbox. In this case, in fact, QP can leverage the high bandwidth of the HBM memory, while not be limited by PCI-e. Additionally, these devices have adequate computation power to fully utilize the high memory bandwidth [1], and they don't suffer from the CPU synchronization overhead since we can use the SIMT-based execution of the GPU to guarantee low synchronization overhead. These devices are also reasonably priced compared to discrete GPUs because each core is designed with the critical computation instructions only. While the per-core performance is weaker than regular CPUs due to simper instructions, limited speculation execution and no hyper-threading, more cores can be packed in the chip, therefore enabling better utilization of the high memory bandwidth. Finally, synchronization between the CPU and the GPU for data access is more fine-grained and cheaper compared to data copy over PCI-e, although careful design is still required for correct concurrent execution.

3 EXPERIMENTS

3.1 Implementation

For this evaluation we have integrated TQP with Antares [15]: a state-of-the-art compiler for Deep Learning models able to tune and code-generate kernels for different hardware. Furthermore, we have extended Antares to support Xbox. The compilation workflow is as follows: (1) each input query is first parsed and optimized using Apache Spark [23]; (2) TQP converts the Spark physical plan into a tree composed of PyTorch programs, one for each physical operator in the original plan; (3) PyTorch `jit.trace` is used to freeze the computation graph composed by the PyTorch programs; (4) The frozen computation graph (in TorchScript format) is lowered into the Antares IR; (5) Antares fuser and tuner are used to generate optimal programs. The above compilation process is fully automatic.

[1]Note that the GPU must be used in order to fully utilize the available bandwidth, i.e., using the CPU cores is not enough to drain all the bandwidth.

3.2 Settings

Hardware Setup. For the experiments we use three different hardware configurations. A CPU-only machine with a Xeon E5-2690 v4 with 2 sockets, and where each socket has 14 cores supporting up 28 threads. The host memory in this machine is a DD4 2400 with 8 banks. The GPU machine has an NVIDIA Tesla P100 with PCIe v3. Finally, we used a standard Xbox Series X. The hardware performance of each machine is summarized in Table 1.

	Xeon E5-2690	P100	Xbox Series X
Memory Bandwidth (GB/s)	154	732	560
Unidirectional PCIv3 (GB/s)	-	16	-
Theoretical TFLOps	1.4	9.5	12.0

Table 1: Experiment Setup for CPU/GPU/Xbox

Experimental Setup. We selected four queries from TPCH. These queries cover a large set of use cases: filter with simple aggregation (query 6); case statement and PK-FK join (14); join, simple aggregation and subquery (17); complex filters involving string match and in statements, join and simple aggregation (19). We run each query using TQP and Antares on the CPU and GPU machine, and on the Xbox. We compare the TQP performance against DuckDB [18]: a popular vectorized databases. We use DuckDB version 6.1. For each query, we run it 10 times and report the median of the last 5 runs. We use TPCH `dbgen` at scale factor 10. Since TQP does not support decimals yet, we convert all decimals columns into doubles.

Query	CPU		NVIDIA P100	Xbox Series X
	DuckDB	TQP	TQP (+ PCIe3)	TQP
TPCH-6	73	56.281	4.294 (+ 251.229)	4.205
TPCH-14	116.523	131.092	10.656 (+ 272.135)	17.302
TPCH-17	789.905	829.936	118.174 (+ 177.088)	155.109
TPCH-19	225.485	194.114	10.403 (+ 322.746)	13.767

Table 2: Latency (in ms) of selected TPCH queries (SF 10).

3.3 Results

We report the end-to-end latency results in Table 2. Starting from the CPU numbers, TQP outperforms DuckDB for query 6 and query 19, and is within 15% from DuckDB for queries 17 and 14. If we focus now on the P100 numbers where the data is already pre-loaded in GPU memory (i.e., no PCIe overhead), TQP outperforms DuckDB by up to 22× (query 19). However, if we add the time to move data from CPU to GPU memory, only query 17 is faster on the P100 than DuckDB on the CPU-only machine. In fact, for query 17, 60% of the total time is spent on data movements over the PCIe, while the

other three queries spend no less than 96% of the total end-to-end time on moving data over the PCIe bus.

Finally, if we look at Xbox numbers, TQP numbers are quite in line with the P100 performance (with data already pre-loaded in GPU memory), but it does not suffer from any PCIe overhead since the CPU and GPU memory are unified. The difference between Xbox and P100 performance is mostly driven by the difference in memory bandwidth (around 23%) between the two devices.

4 CONCLUSION

The Xbox console has an interesting hardware configuration where the CPU and the GPU are integrated and share 29GB of HBM. In this paper we run few selected TPCH queries on an Xbox, and see that the Xbox performance can be comparable to that of a discrete GPU while avoiding the PCIe data transfer bottleneck. Based on these encouraging early results, we plan to support the full TPCH soon and share more details of our approach in an upcoming paper.

REFERENCES

[1] 2023. Apple Silicon. https://en.wikipedia.org/wiki/Apple_silicon. [Online; accessed Dec 2023].

[2] 2023. HBM. https://en.wikipedia.org/wiki/High_Bandwidth_Memory. [Online; accessed Dec 2022].

[3] 2023. XBox. https://en.wikipedia.org/wiki/Xbox. [Online; accessed Dec 2023].

[4] 2023. XCloud. https://en.wikipedia.org/wiki/Xbox_Cloud_Gaming. [Online; accessed Dec 2023].

[5] Gustavo Alonso. 2023. Data Processing in the Hardware Era. https://www.youtube.com/watch?v=KekKAKI0Aho&feature=youtu.be.

[6] ampere. 2023. Ampere Altra Max. https://amperecomputing.com/processors/ampere-altra.

[7] anandtech. 2021. AMD gives details on EPYC Zen4 Genoa and Bergamo, up to 96 and 128 cores. amd-gives-details-on-epyc-zen4-genoa-and-bergamo-up-to-96-and-128-cores.

[8] Haran Boral and David J. DeWitt. 1983. Database Machines: An Idea Whose Time has Passed? A Critique of the Future of Database Machines. In *Database Machines*, H.-O. Leilich and M. Missikoff (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 166–187.

[9] Helena Caminal, Yannis Chronis, Tianshu Wu, Jignesh M. Patel, and José F. Martínez. 2022. Accelerating Database Analytic Query Workloads Using an Associative Processor. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 623–637. https://doi.org/10.1145/3470496.3527435

[10] Anshuman Dasgupta, Zaid Al-Ars, Gustavo Alonso, and Timothy Roscoe. 2010. Database Acceleration Using Reconfigurable Hardware. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (Indianapolis, Indiana) *(SIGMOD '10)*. ACM, New York, NY, USA, 1119–1122. https://doi.org/10.1145/1807167.1807294

[11] Gruber, Ferdinand; Bandle, Maximilian; Engelke, Alexis; Neumann, Thomas; Giceva, Jana. 2023. Bringing Compiling Databases to RISC Architectures. https://db.in.tum.de/~gruber/p791-gruber.pdf. [Online; accessed March 2023].

[12] Bingsheng He, Xiaoyi Lu, Wenjian Wang, Fan Wu, and Rui Zhang. 2015. In-Cache Query Co-Processing on Coupled CPU-GPU Architectures. *Proceedings of the VLDB Endowment* 8, 4 (2015), 329–340.

[13] Dong He, Supun C Nakandala, Dalitso Banda, Rathijit Sen, Karla Saur, Kwanghyun Park, Carlo Curino, Jesús Camacho-Rodríguez, Konstantinos Karanasos, and Matteo Interlandi. 2022. Query Processing on Tensor Computation Runtimes. *PVLDB* (2022), 2811–2825.

[14] hpcwire. 2023. Intel Officially Launches Sapphire Rapids and HPC-optimized Max Series. https://www.hpcwire.com/2023/01/10/intel-officially-launches-sapphire-rapids-and-max-series/.

[15] Microsoft. 2023. Antares. https://github.com/microsoft/antares.

[16] Thomas Neumann. 2011. Efficiently Compiling Efficient Query Plans for Modern Hardware. *Proc. VLDB Endow.* 4, 9 (jun 2011), 539–550. https://doi.org/10.14778/2002938.2002940

[17] Jason Power, Yinan Li, Mark D. Hill, Jignesh M. Patel, and David A. Wood. 2015. Toward GPUs Being Mainstream in Analytic Processing: An Initial Argument Using Simple Scan-Aggregate Queries. In *Proceedings of the 11th International Workshop on Data Management on New Hardware* (Melbourne, VIC, Australia) *(DaMoN'15)*. Association for Computing Machinery, New York, NY, USA, Article 11, 8 pages. https://doi.org/10.1145/2771937.2771941

[18] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1981–1984. https://doi.org/10.1145/3299869.3320212

[19] Anil Shanbhag, Samuel Madden, and Xiangyao Yu. 2020. A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics. In *SIGMOD*. 1617–1632.

[20] Ryan Smith. 2023. CES 2023: AMD Instinct MI300 Data Center APU Silicon In Hand - 146B Transistors, Shipping H2'23. https://www.anandtech.com/show/18721/ces-2023-amd-instinct-mi300-data-center-apu-silicon-in-hand-146b-transistors-shipping-h223.

[21] Versus. 2023. Memory bandwidth. https://versus.com/en/glossary/memory-bandwidth.

[22] Lisa Wu, Andrea Lottarini, Timothy K. Paine, Martha A. Kim, and Kenneth A. Ross. 2014. Q100: The Architecture and Design of a Database Processing Unit. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 255–268. https://doi.org/10.1145/2541940.2541961

[23] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Spark: Cluster Computing with Working Sets. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA) *(OSDI'12)*. USENIX Association, 10–10. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/zaharia

[24] Jingren Zhou and Kenneth A. Ross. 2002. Implementing Database Operations Using SIMD Instructions. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, Wisconsin) *(SIGMOD '02)*. Association for Computing Machinery, New York, NY, USA, 145–156. https://doi.org/10.1145/564691.564709