An Energy-Efficient Stream Join for the Internet of Things

Adrian Michalke[†] Philipp M. Grulich[‡] Clemens Lutz[†*] Steffen Zeuch[†‡] Volker Markl[†‡]
[†]DFKI GmbH [‡]TU Berlin

ABSTRACT

The Internet of Things (IoT) combines large data centers with (mobile, networked) edge devices that are constrained both in compute power and energy budget. Modern edge devices contribute to query processing by leveraging accelerated processing units with multicore CPUs or GPUs. Therefore, data processing in the IoT presents the challenges of 1) minimizing the energy consumed while sustaining a given query throughput, and 2) processing increasingly complex queries within a given energy budget.

In this paper, we investigate how modern edge devices can reduce the energy requirements of stream joins as a common data processing operation. We explore three dimensions to save energy: workload characteristics, computational efficiency, and heterogeneous hardware. Based on our findings, we propose the ecoJoin that 1) reduces energy consumption by 81% at a given join throughput, and 2) enables scaling the throughput by two orders-of-magnitude within a given energy budget.

ACM Reference Format:
Adrian Michalke, Philipp M. Grulich, Clemens Lutz, Steffen Zeuch, and Volker Markl. 2021. An Energy-Efficient Stream Join for the Internet of Things. In *International Workshop on Data Management on New Hardware (DAMON'21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3465998.3466005

1 INTRODUCTION

The Internet of Things (IoT) [3] enables a wide range of new application scenarios. These scenarios cover different domains such as biosensors for medical patients [2, 30], concrete structure monitoring [4], or smart grid management [10, 11] and perform complex analytical workloads, e.g., joins and aggregations, on geographically distributed data streams [13]. To manage such data streams efficiently, a new class of IoT data management systems has emerged [6, 35, 54]. These systems offload data processing tasks to *edge devices*, which are located close to the data sources, e.g., sensors. Edge devices gather data from the sensors, perform preprocessing, and send intermediate results to the cloud. However, edge devices are usually battery-powered and thus *energy-constrained* [9, 13, 44, 53]. As a result, it is crucial to take energy consumption into account. To this end, an energy-aware IoT data management system must address the following three challenges:

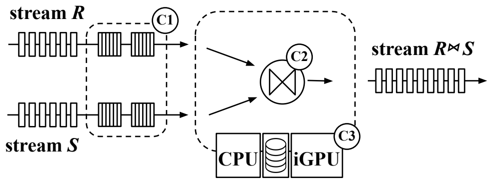

Figure 1: EcoJoin combines three energy optimizations.

C1: Workload Characteristics. Modern processors balance performance and power consumption. They sleep in a low power state to save energy. When there is work to do, processors *race-to-idle* by ramping up their clock speed, process the data quickly, and return to the idle state. However, predicting the optimal time point to switch states and set clock frequency is difficult. Consequently, an energy-aware IoT data management system must take workload characteristics into account to dynamically adjust power states.

C2: Computational efficiency. Due to the race-to-idle paradigm, high-throughput directly translates into energy-efficiency. For relational databases, research proposed a variety of energy-efficient operator implementations [14–16, 22, 24, 39, 42, 48]. In contrast, the energy-efficiency of stream processing workloads has not been addressed yet. In particular, streaming queries inherently have different semantics compared to relational workloads, and optimize latency in addition to throughput. Consequently, an energy-aware IoT data management system should revisit the design and implementations of common operators.

C3: Heterogeneous Hardware. Modern edge devices are highly integrated System-on-a-Chip circuits (SoC) [32, 41, 50]. They offer a diverse set of heterogeneous computational resources with different energy and performance profiles, e.g., multi-core CPUs and GPUs. Current stream processing systems are usually optimized for general-purpose hardware, e.g., they use a JVM to abstract from hardware details [7]. As a result, they cannot fully exploit the hardware resources of edge devices [54]. Consequently, an energy-aware IoT data management system should take advantage of the available heterogeneous hardware resources of modern edge devices.

In this paper, we investigate these challenges for an energy-aware IoT data management system. We focus on stream joins, as joins are a commonly-used, computation-intensive operator in IoT workloads [55]. In particular, we propose ecoJoin[1], a novel energy-efficient stream join approach.

2 ENERGY-EFFICIENT STREAM JOIN

In this section, we present ecoJoin, an energy-efficient stream join. EcoJoin combines three techniques to reduce energy consumption, shown in Figure 1: ① Exploiting workload characteristics, ② reducing algorithmic complexity, and ③ utilizing hardware resources.

[*] currently employed at TU Berlin.

[1]https://github.com/TU-Berlin-DIMA/ecoJoin

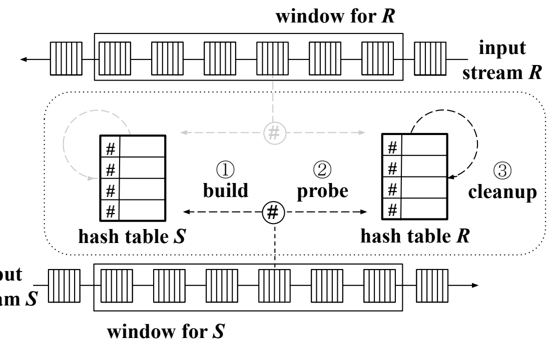

Figure 2: The hash partitioning scheme of ecoJoin.

2.1 Exploiting Workload Characteristics

Tuple-wise processing prevents the processor from entering low power states, because the tuple stream generates a steady load on the processor over a long period of time. With ecoJoin, we propose to optimize power consumption by racing the join to idle in a two-step approach. In the first step, we batch tuples to split processing into idle and race phases. The idle phase enables the processor to enter a low power state to save energy. Larger batch sizes lead to longer idle phases and to higher join throughput due to cache locality. However, large batch sizes increase processing latency. To avoid high processing latencies, ecoJoin adaptively adjusts batch sizes based on stream input rates and the latency tolerances provided by the user. In a second step, we adjust the processor's clock rate to conserve energy. Our ecoJoin determines the clock rate based on the processing phase and the input rate. It sets the idle frequency to the lowest value supported by the processor and the race frequency to the optimal clock rate. We obtain the optimal clock rate by pre-computing it based on the observation that the join is a memory-bounded operator [18, 38]. Thus, processors achieve their peak join throughput without boosting to the maximum frequency.

2.2 Increasing Computational Efficiency

State-of-the-art stream joins are based on nested loops because the time interval is typically a range predicate. Furthermore, nested loop joins are easily parallelizable for multi-threaded execution. However, they do not scale to fast input rates on large windows, due to their low computational efficiency.

EcoJoin improves computational efficiency by following a three-phase approach, shown in Figure 2. We describe the steps for the first stream. The second stream follows the same process with the streams mirrored. After presenting the core algorithm, we show how a multi-core processor can run the phases in parallel.

(1) Build phase. The build starts when a batch of tuples is ready to be processed. Thus, we assume that tuples arrive in an input queue, e.g., via the network, while the processor is idle. When the join starts, it first inserts all tuples of the batch into a hash table, which is partitioned on the tuple's key value. Because the second stream follows the same procedure, ecoJoin maintains two hash tables. Within a hash table, each bucket contains a buffer with tuples from the current join interval. To insert tuples into an interval buffer, the ecoJoin increments a fill-state counter that points to the next free slot.

(2) Probe phase. The same batch is probed with tuples in the second stream's hash table. For each tuple in the batch, the join looks up the bucket with the corresponding key. If the bucket exists, the join evaluates the join predicate for each tuple in the bucket. The predicate matches tuples by key equivalence and by the time interval. If they match, ecoJoin emits matching tuples into a buffered result stream. To allocate a slot, the join increments the result buffer's fill-state counter.

(3) Cleanup phase. After the probe is complete, the join garbage-collects invalid tuples in bulk. Invalid tuples are tuples that have exceeded the range of the last observed time interval — they will not match any further tuples. Invalid tuples are removed by iterating through the bucket from both ends, and replacing each invalid tuple in the front with a valid tuple from the back. As a full scan of the hash table is expensive and invalidates only one batch of tuples, ecoJoin tries to reduce the number of garbage collection passes. To this end, ecoJoin counts the number of invalid tuples that it encounters per hash table bucket in the probe phase. If a bucket exceeds a specified threshold of invalid tuples, the join marks the bucket in a bitmap. Then, in the cleanup phase, the garbage collector passes only over the marked buckets.

After every algorithm cycle, the join switches the streams and thus processes batches from both streams.

Parallelization. To speed-up the join, ecoJoin parallelizes all three phases. To this end, ecoJoin evenly distributes the tuple batch over all cores. Each core independently runs build and probe phases. Within those phases, cores modify the fill-state counters with atomic increment instructions. Before proceeding to the cleanup phase, all cores wait on a global barrier until the probe phase is complete. In the cleanup phase, ecoJoin assigns each bucket to a core. In contrast to the other phases, the cleanup requires no synchronization among the cores. Then, the cores again wait on a barrier before starting the next batch.

2.3 Exploiting Heterogeneous Processors

Modern edge devices feature processors that integrate GPUs. We tune our implementation to run efficiently on CPUs and on GPUs. This enables us to switch to the most energy-efficient processor, depending on the available hardware and the given workload. We describe how our ecoJoin takes advantage of the integrated GPU.

Overview. In our design, the CPU orchestrates the GPU's execution, in three steps. First, the CPU receives the input data and batches the tuples. When a batch is ready, the CPU sets the GPU to its race frequency, and launches a GPU kernel that performs the build and probe phases. The GPU performs coalesced accesses to load the tuples from memory. After the build and probe are complete, the CPU checks if any buffer is marked for cleaning. If yes, the CPU launches the cleanup kernel on the GPU. After that, the CPU sets the GPU to its idle frequency and waits for the next batch.

Memory Zones. The integrated GPU provides direct access to main-memory [34], albeit with restrictions. The CPU and the GPU are not cache-coherent on current edge devices [5, 33], as cache-coherence increases hardware complexity [37]. Instead, main-memory is segmented into three zones [34]: CPU-only memory, zero-copy memory, and GPU-only memory. Each zone has different access

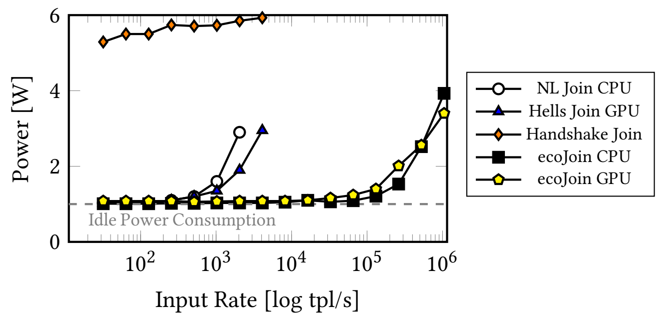

Figure 3: Power consumption of stream join variants.

characteristics. The GPU cannot directly access CPU-only memory. Zero-copy memory is directly accessible by both processors, but only with reduced bandwidth for the GPU. Finally, GPU-only memory provides the full main-memory bandwidth to the GPU, but cannot be directly accessed by the CPU.

As we assume that the streams are ingested via the network, the join accesses the input and result queues in zero-copy memory. Thus, we avoid copying data between main-memory zones. In contrast, we store the hash table in GPU-only memory, to take full advantage of the memory bandwidth.

3 EVALUATION

In this section, we conduct a set of experiments to evaluate our ecoJoin in terms of power consumption, throughput and latency. First, we introduce our experimental setup (see Section 3.1). After that, we present and discuss our results (see Section 3.2).

3.1 Experimental Setup

In the following, we describe our hardware and software setup, the workload characteristic, and the measurement methodology.

Hardware and Software. We conduct all experiments on an NVIDIA Jetson Nano as a representative IoT edge device. It includes a quad-core ARM A57 at 1.43GHz, a 128-core NVIDIA Maxwell GPU at 921MHz, and 4GB RAM. The system runs Ubuntu 18.04. Furthermore, we use GCC 7.4.0 and CUDA 10.0 for compilation.

Workload. We join two streams R: (int X, float Y) and S: (int A, float B) on the join predicate $R.X = S.A$. We use synthetically generated data. The keys follow a uniform distribution between 0 and INT_MAX. Tuples arrive with an symmetric input rate and are processed in a fixed window size of 10 seconds. We use a hash bucket size of 64 across all experiments.

Measurement Methodology. In general, we conduct all experiments over a runtime of 2 minutes. We define throughput as the highest input rate for which all tuples could be processed in the given runtime. We define latency as the time difference between the moment the younger tuple arrives and the moment its join pair is emitted. For energy profiling, we use an ARM Energy Probe [1] to measure power consumption in real-time. We direct the current through a shunt resistor with 0.1Ω and measure it at a sample rate of 2kHz. For the Jetson Nano, we measure an idle power consumption of 1.03W. Across all experiments, we denote the average power consumption over the measured time frame as $Power[W]$.

3.2 Experiments

Our evaluation consists of three experiments. First, we compare throughput and energy consumption to state-of-the-art stream join algorithms (see Section 3.2.1). Second, we discuss the impact of batching and frequency scaling on the energy consumption of our ecoJoin (see Section 3.2.2). Third, we investigate the impact of adaptive batching on energy consumption and latency (see Section 3.2.3).

3.2.1 Stream Join Variants. We compare the energy-efficiency among two variants of our ecoJoin and three baselines. As representative state-of-the-art stream join algorithms, we consider the Handshake Join (CPU) [45] and the HELLS Join (GPU/CPU Co-processing) [21]. To this end, we ported the Handshake Join to ARM, disabled SIMD, and adjust its join predicate. We reimplemented the HELLS Join in CUDA and extended it to support our adaptive batching technique. For our ecoJoin, we conduct the experiment with a CPU and GPU version. Additionally, we evaluate a naive nested loop join algorithm to assess the impact of the hash-based processing strategy of our ecoJoin. Across all experiments, we vary the input rate from 10^2 to 10^6 tuples/s, leading to a join selectivity up to 0.9%.

Results. Figure 3 shows the throughput and power consumption for different ingestion rates across the individual stream join variants. In general, we observe that the maximum achievable throughput varies significantly across the individual stream join algorithms. The nested loop variant achieves the lowest throughput of 2K tuples/s. Both HELLS Join and Handshake Join achieve a similar throughput of around 4K tuples/s. In contrast, our hash-based ecoJoin achieves a significantly higher throughput of up to 1M tuples/s on both the CPU and iGPU.

Additionally, we observe that GPU-based HELLS Join outperforms the CPU-based NL Join. This indicates a performance advantage of the iGPU for nested-loop-based join. In contrast, our ecoJoin reaches the same throughput on both CPU and iGPU. In terms of energy-efficiency, we observe that overall the Handshake Join has a high power consumption of over 5.5W, even for low input rates. This indicates that the Handshake Join is optimized for high performance instead of energy-efficiency. All other implementations scale gracefully with the rising workload and reach idle power consumption for low input rates. In comparison to baselines, we observe that our ecoJoin reduces power consumption significantly, up to 81% compared to Handshake Join and 65% compared to HELLS Join.

Summary. Overall, ecoJoin outperforms state-of-the-art algorithms in terms of throughput and energy consumption due to its higher computational efficiency.

3.2.2 Batching and Frequency Scaling. In this experiment, we study the impact of adaptive batching and frequency scaling (see Section 2.1). We measure three variants of our CPU-based ecoJoin. *FB* uses a fixed batch size of 1024 tuples, *AB* adapts the batch size to the input rate, and *AB+FS* uses adaptive batching in combination with fine-grained frequency scaling. Furthermore, we depict the *idle power* and the *max power* consumption[2]. Across the experiment, we vary the input rate from 10^2 to 10^6 tuples/s.

[2]idle power = power consumption without any workload; max power = power consumption if the CPU uses always the highest clock frequency.

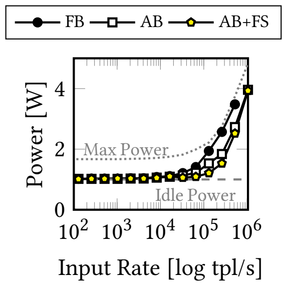

Figure 4: Adaptive batching and frequency scaling.

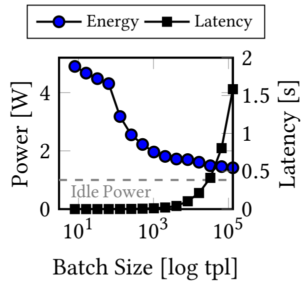

Figure 5: Impact of batch size as power-latency trade-off.

Results. Figure 4 shows the power consumption for increasing ingestion rates across all three variants. Overall, fixed batching (*FB*) achieves the lowest throughput with 0.5M tuples/s. In contrast, both adaptive batching-based methods (*AB* and *AB+FS*) achieve up to 1M tuples/s. Additionally, we observe that adaptive batching reduces the power consumption across all ingestion rates by up to 28% compared to *FB*. Furthermore, we see that for high ingestion rates, frequency scaling (*AB+FS*) further reduces the power consumption by up to 20%. As a result, adaptive batching and frequency scaling reduce the power consumption of our ecoJoin by over 1W.

Summary. We conclude that adaptive batching and frequency scaling significantly improves the energy-efficiency.

3.2.3 Energy-Latency Trade-Off. In this experiment, we investigate the impact of the batch size on power consumption and latency. To this end, we use a fixed ingestion rate of 10k tuples per second and vary the batch size within the range of 10 and 100k tuples.

Results. In Figure 5, we observe that latency and power consumption are inversely proportional. Thus, with increasing batch sizes, the latency increases from 1.28ms up to 1.59s. At the same time, the power consumption decreases from 4.93W to 1.46W. Furthermore, batches of 10k tuple are a crossover point below which latency is less than 100ms, and above which power consumption nearly stops to decrease. As a result, a batch size of 10k tuples is a good trade-off between power consumption and latency.

Summary. In sum, a large batch size reduces power consumption but increases latency. However, there is a sweet-spot at which power consumption is close to the minimum while latency remains low.

3.3 Discussion

Our experiments have studied the processing performance and energy-efficiency of our ecoJoin. Our analysis has shown that ecoJoin outperforms state-of-the-art stream join algorithms significantly in throughput and power consumption. Furthermore, we have shown that adaptive batching and frequency scaling improve energy-efficiency. As a result, we argue that energy-aware data processing approaches consider workload characteristics, computational efficiency, and heterogeneous hardware to achieve high energy-efficiency.

4 RELATED WORK

Our energy-efficient ecoJoin combines research from the areas of *Energy-Aware Data Management Systems* and *Efficient Stream Join Processing*. In the following, we categorize related work accordingly.

Energy-Aware Data Management. Energy awareness of data management systems has been an important area of research over the last years [12, 15, 24, 28, 47, 51]. Based on initial experiments, Harizopoulos et al. [15] formulated the vision for energy-aware databases management systems. Tsirogiannis et al. [47], evaluated different database configurations and argued that energy-efficiency always correlates with processing efficiency. Following these findings, we proposed our ecoJoin as an energy-efficient stream join. Our work reveals that energy-efficiency is an important design aspect for data processing algorithms on edge devices. Similar to Götz et al. [12] and Kissinger et al. [24] our ecoJoin applies race-to-idle to stream processing workloads. Compared to relational workloads, stream processing workloads perform long-running queries over unbounded data streams, directly on the battery-powered edge devices. As a result, even small energy savings would accumulate to longer battery life-times and thus longer operation times.

Further research utilized heterogeneous hardware to improve the energy-efficiency of data processing workloads, e.g., by using clusters of low-power SoCs [27, 29, 42], by investigating energy-optimized CPU architectures [49], or by leveraging SoCs with integrated GPUs [8]. We follow this line of research and leverage a heterogeneous SoC to investigate the energy-efficiency trade-off between CPUs and GPUs for stream processing in detail.

Efficient Stream Join Processing. Joining data streams has been extensively studied over the last years. Research has been conducted on stream joins in distrusted systems [17, 20], index structures for state materialization [23, 43, 46], and efficient utilization of modern hardware [21, 25, 26, 31, 36, 40, 55]. Our ecoJoin, which relates to these works in two aspects. First, our ecoJoin extends the symmetric hash-join [52] with a record eviction policy. In contrast to Kang et al. [19], we reduce the cleanup overhead by only evicting records if the hash-table is close to its maximum capacity. Second, our ecoJoin focuses on the efficient utilization of SoCs with integrated CPUs and GPUs. Karnagel et al. [21] and Körber et al. [25] demonstrated performance advantages of GPUs on similar hardware. In contrast to this work, our ecoJoin leverages a hash-based join if the stream consists of multiple keys, which shifts the trade-off between GPU and CPU. Further research proposed stream joins for FPGAs [26, 31, 36] to improve energy efficiency. However, FPGAs are still not common on edge hardware.

5 CONCLUSION

We conclude that combining orthogonal dimensions leads to higher energy-efficiency when joining streams. The most significant energy savings come from increased computational efficiency, followed by adaptive batching with frequency scaling. iGPUs are competitive with CPUs. Overall, with ecoJoin, we make the case that IoT environments present new challenges for energy-aware data stream processing. In future work, we will incorporate ecoJoin in our new data processing platform NebulaStream [54].

Acknowledgments

This work was funded by the DFG Priority Program (MA4662-5), the EU Horizon 2020 programme (780245), the German Federal Ministry for Economic Affairs and Energy (01MD19002B), and the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A).

REFERENCES

[1] ARM 2013. *ARM Energy Probe*. ARM. Retrieved Mar 25, 2021 from https://developer.arm.com/documentation/dui0482/l/the-energy-probe/energy-probe-overview

[2] Ian F. Akyildiz and Josep Miquel Jornet. 2010. The Internet of nano-things. *IEEE Wirel. Commun.* 17, 6 (2010), 58–63. https://doi.org/10.1109/MWC.2010.5675779

[3] Kevin Ashton. 2009. That "Internet of Things" thing. *RFID Journal* (June 2009).

[4] Norberto Barroca, Luis Borges, Fernando Velez, Filipe Monteiro, Marcin Górski, and João Castro-Gomes. 2013. Wireless sensor networks for temperature and humidity monitoring within concrete structures. *Construction and Building Materials* 40 (03 2013), 1156–1166. https://doi.org/10.1016/j.conbuildmat.2012.11.087

[5] Jeremy Bentham. 2020. *Raspberry Pi DMA programming in C*. Retrieved Mar 25, 2021 from https://iosoft.blog/2020/05/25/raspberry-pi-dma-programming/

[6] Flavio Bonomi, Rodolfo A. Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *MCC@SIGCOMM*, Mario Gerla and Dijiang Huang (Eds.). ACM, 13–16. https://doi.org/10.1145/2342509.2342513

[7] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).

[8] Xuntao Cheng, Bingsheng He, and Chiew Tong Lau. 2015. Energy-Efficient Query Processing on Embedded CPU-GPU Architectures. 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 10, 7 pages. https://doi.org/10.1145/2771937.2771939

[9] Peijin Cong, Junlong Zhou, Liying Li, Kun Cao, Tongquan Wei, and Keqin Li. 2020. A Survey of Hierarchical Energy Optimization for Mobile Edge Computing: A Perspective from End Devices to the Cloud. *ACM Comput. Surv.* 53, 2 (2020), 38:1–38:44. https://doi.org/10.1145/3378935

[10] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. 2012. Smart Grid - The New and Improved Power Grid: A Survey. *IEEE Commun. Surv. Tutorials* 14, 4 (2012), 944–980. https://doi.org/10.1109/SURV.2011.101911.00087

[11] Hassan Farhangi. 2010. The path of the smart grid. *IEEE Power and Energy Magazine* 8, 1 (2010), 18–28. https://doi.org/10.1109/MPE.2009.934876

[12] Sebastian Gotz, Thomas Ilsche, Jorge Cardoso, Josef Spillner, Thomas Kissinger, Uwe Assmann, Wolfgang Lehner, Wolfgang E. Nagel, and Alexander Schill. 2014. Energy-Efficient Databases Using Sweet Spot Frequencies. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing* (London, United Kingdom, 2014-12). IEEE, 871–876. https://doi.org/10.1109/UCC.2014.142

[13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* 29, 7 (2013), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

[14] Sebastian Haas and Gerhard P. Fettweis. 2017. Energy-Efficient Hash Join Implementations in Hardware-Accelerated MPSoCs. In *International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures, ADMS@VLDB 2017, Munich, Germany, September 1, 2017*, Rajesh Bordawekar and Tirthankar Lahiri (Eds.). 26–33. http://www.adms-conf.org/2017/camera-ready/adms17_p3_haas.pdf

[15] Stavros Harizopoulos, Mehul A. Shah, Justin Meza, and Parthasarathy Ranganathan. 2009. Energy Efficiency: The New Holy Grail of Data Management Systems Research. In *Fourth Biennial Conference on Innovative Data Systems Research, CIDR 2009, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings*. www.cidrdb.org. http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_112.pdf

[16] Ahmad Hassan, Hans Vandierendonck, and Dimitrios S. Nikolopoulos. 2015. Energy-Efficient In-Memory Data Stores on Hybrid Memory Hierarchies. In *DaMoN*, Ippokratis Pandis and Martin L. Kersten (Eds.). ACM, 1:1–1:8. https://doi.org/10.1145/2771937.2771940

[17] Gabriela Jacques-Silva, Ran Lei, Luwei Cheng, Guoqiang Jerry Chen, Kuen Ching, Tanji Hu, Yuan Mei, Kevin Wilfong, Rithin Shetty, Serhat Yilmaz, et al. 2018. Providing streaming joins as a service at facebook. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1809–1821.

[18] Christopher Jonathan, Umar Farooq Minhas, James Hunter, Justin J. Levandoski, and Gor V. Nishanov. 2018. Exploiting Coroutines to Attack the "Killer Nanoseconds". *Proc. VLDB Endow.* 11, 11 (2018), 1702–1714. https://doi.org/10.14778/3236187.3236216

[19] Jaewoo Kang, Jeffrey F. Naughton, and Stratis Viglas. 2003. Evaluating Window Joins over Unbounded Streams. In *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman (Eds.). IEEE Computer Society, 341–352. https://doi.org/10.1109/ICDE.2003.1260804

[20] Jeyhun Karimov, Tilmann Rabl, and Volker Markl. 2019. AJoin: ad-hoc stream joins at scale. *Proceedings of the VLDB Endowment* 13, 4 (2019), 435–448.

[21] Tomas Karnagel, Dirk Habich, Benjamin Schlegel, and Wolfgang Lehner. 2013. The HELLS-join: a heterogeneous stream join for extremely large windows. In *Proceedings of the Ninth International Workshop on Data Management on New Hardware, DaMoN 2013, New York, NY, USA, June 24, 2013*, Ryan Johnson and Alfons Kemper (Eds.). ACM, 2. https://doi.org/10.1145/2485278.2485280

[22] Alexey Karyakin and Kenneth Salem. 2019. DimmStore: Memory Power Optimization for Database Systems. *Proc. VLDB Endow.* 12, 11 (2019), 1499–1512. https://doi.org/10.14778/3342263.33422629

[23] Hyeon Gyu Kim. 2013. A Structure for Sliding Window Equijoins in Data Stream Processing. In *16th IEEE International Conference on Computational Science and Engineering, CSE 2013, December 3-5, 2013, Sydney, Australia*. IEEE Computer Society, 100–103. https://doi.org/10.1109/CSE.2013.25

[24] Thomas Kissinger, Dirk Habich, and Wolfgang Lehner. 2018. Adaptive Energy-Control for In-Memory Database Systems. In *Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18* (Houston, TX, USA). ACM Press, 351–364. https://doi.org/10.1145/3183713.3183756

[25] Michael Körber, Jakob Eckstein, Nikolaus Glombiewski, and Bernhard Seeger. 2019. Event Stream Processing on Heterogeneous System Architecture. In *Proceedings of the 15th International Workshop on Data Management on New Hardware* (Amsterdam, Netherlands) *(DaMoN'19)*. Association for Computing Machinery, New York, NY, USA, Article 3, 10 pages. https://doi.org/10.1145/3329785.3329933

[26] Charalabos Kritikakis, Grigorios Chrysos, Apostolos Dollas, and Dionisios N. Pnevmatikatos. 2016. An FPGA-based high-throughput stream join architecture. In *26th International Conference on Field Programmable Logic and Applications, FPL 2016, Lausanne, Switzerland, August 29 - September 2, 2016*, Paolo Ienne, Walid A. Najjar, Jason Helge Anderson, Philip Brisk, and Walter Stechele (Eds.). IEEE, 1–4. https://doi.org/10.1109/FPL.2016.7577354

[27] Willis Lang, Stavros Harizopoulos, Jignesh M. Patel, Mehul A. Shah, and Dimitris Tsirogiannis. 2012. Towards Energy-Efficient Database Cluster Design. *Proc. VLDB Endow.* 5, 11 (2012), 1684–1695. https://doi.org/10.14778/2350229.2350280

[28] Willis Lang, Ramakrishnan Kandhan, and Jignesh Patel. 2011. Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race. *IEEE Data Eng. Bull.* 34 (01 2011), 12–23.

[29] Dumitrel Loghin, Bogdan Marius Tudor, Hao Zhang, Beng Chin Ooi, and Yong Meng Teo. 2015. A Performance Study of Big Data on Small Nodes. *Proc. VLDB Endow.* 8, 7 (2015), 762–773. https://doi.org/10.14778/2752939.2752945

[30] Mubarak Mujawar, Hardik Gohel, Sheetal Bhardwaj, Sesha Srinivasan, Nicolerta Hickman, and Ajeet Kaushik. 2020. Aspects of nano-enabling biosensing systems for intelligent healthcare; towards COVID-19 management. *Materials Today Chemistry* 17 (06 2020), 100306. https://doi.org/10.1016/j.mtchem.2020.100306

[31] Mohammadreza Najafi, Mohammad Sadoghi, and Hans-Arno Jacobsen. 2016. SplitJoin: A Scalable, Low-latency Stream Join Architecture with Adjustable Ordering Precision. (2016), 493–505. https://www.usenix.org/conference/atc16/technical-sessions/presentation/najafi

[32] Nvidia. 2019. *Nvidia announces Jetson Nano*. Retrieved Mar 25, 2021 from https://nvidianews.nvidia.com/news/nvidia-announces-jetson-nano-99-tiny-yet-mighty-nvidia-cuda-x-ai-computer-that-runs-all-ai-models

[33] Nvidia 2019. *Nvidia Tegra X1 Mobile Processor*. Nvidia. https://developer.download.nvidia.com/assets/embedded/secure/jetson/TX1/docs/Tegra_X1_TRM_DP07225001_v1.3p.pdf DP-07225-001_v1.3p.

[34] Nvidia 2021. *CUDA for Tegra*. Nvidia. https://docs.nvidia.com/cuda/pdf/CUDA-for-Tegra-AppNote.pdf DA-06762-001_v11.2.

[35] Dan O'Keeffe, Theodoros Salonidis, and Peter Pietzuch. 2018. Frontier: Resilient edge processing for the internet of things. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1178–1191.

[36] Fei Pan and Hans-Arno Jacobsen. 2018. PanJoin: A Partition-based Adaptive Stream Join. *CoRR* abs/1811.05065 (2018). arXiv:1811.05065 http://arxiv.org/abs/1811.05065

[37] Jason Power, Arkaprava Basu, Junli Gu, Sooraj Puthoor, Bradford M. Beckmann, Mark D. Hill, Steven K. Reinhardt, and David A. Wood. 2013. Heterogeneous system coherence for integrated CPU-GPU systems. In *MICRO*, Matthew K. Farrens and Christos Kozyrakis (Eds.). ACM, 457–467. https://doi.org/10.1145/2540708.2540747

[38] Georgios Psaropoulos, Thomas Legler, Norman May, and Anastasia Ailamaki. 2017. Interleaving with Coroutines: A Practical Approach for Robust Index Joins. *Proc. VLDB Endow.* 11, 2 (2017), 230–242. https://doi.org/10.14778/3149193.3149202

[39] Iraklis Psaroudakis, Thomas Kissinger, Danica Porobic, Thomas Ilsche, Erietta Liarou, Pinar Tözün, Anastasia Ailamaki, and Wolfgang Lehner. 2014. Dynamic fine-grained scheduling for energy-efficient main-memory queries. In *DaMoN*, Alfons Kemper and Ippokratis Pandis (Eds.). ACM, 1:1–1:7. https://doi.org/10.1145/2619228.2619229

[40] Pratanu Roy, Jens Teubner, and Rainer Gemulla. 2014. Low-Latency Handshake Join. *Proc. VLDB Endow.* 7, 9 (2014), 709–720. https://doi.org/10.14778/2732939.2732944

[41] Samsung. 2019. *Exynos Auto V9*. Retrieved Mar 25, 2021 from https://www.samsung.com/semiconductor/minisite/exynos/products/automotiveprocessor/exynos-auto-v9

[42] Daniel Schall and Theo Härder. 2013. Energy-proportional query execution using a cluster of wimpy nodes. In *DaMoN*, Ryan Johnson and Alfons Kemper (Eds.). ACM, 1. https://doi.org/10.1145/2485278.2485279

[43] Amirhesam Shahvarani and Hans-Arno Jacobsen. 2020. Parallel Index-based Stream Join on a Multicore CPU. (2020), 2523–2537. https://doi.org/10.1145/3318464.3380576

[44] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* 3, 5 (2016), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

[45] Jens Teubner and René Müller. 2011. How soccer players would do stream joins. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis (Eds.). ACM, 625–636. https://doi.org/10.1145/1989323.1989389

[46] Quoc-Cuong To, Juan Soto, and Volker Markl. 2018. A survey of state management in big data processing systems. *VLDB J.* 27, 6 (2018), 847–872. https://doi.org/10.1007/s00778-018-0514-9

[47] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. [n.d.]. Analyzing the energy efficiency of a database server. In *Proceedings of the 2010 international conference on Management of data - SIGMOD '10* (Indianapolis, Indiana, USA, 2010). ACM Press, 231. https://doi.org/10.1145/1807167.1807194

[48] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. 2010. Analyzing the energy efficiency of a database server. In *SIGMOD*, Ahmed K. Elmagarmid and Divyakant Agrawal (Eds.). ACM, 231–242. https://doi.org/10.1145/1807167.1807194

[49] Annett Ungethüm, Dirk Habich, Tomas Karnagel, Wolfgang Lehner, Nils Asmussen, Marcus Völp, Benedikt Noethen, and Gerhard P. Fettweis. 2015. Query processing on low-energy many-core processors. In *31st IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2015, Seoul, South Korea, April 13-17, 2015*. IEEE Computer Society, 155–160. https://doi.org/10.1109/ICDEW.2015.7129569

[50] Eben Upton. 2019. *Raspberry Pi 4 on sale now from $35.* Retrieved Mar 15, 2021 from https://www.raspberrypi.org/blog/raspberry-pi-4-on-sale-now-from-35

[51] Han Wang, Setareh Rafatirad, and Houman Homayoun. 2019. A+ Tuning: Architecture+Application Auto-Tuning for In-Memory Data-Processing Frameworks. In *25th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2019, Tianjin, China, December 4-6, 2019*. IEEE, 163–166. https://doi.org/10.1109/ICPADS47876.2019.00032

[52] Annita N. Wilschut and Peter M. G. Apers. 1993. Dataflow Query Execution in a Parallel Main-memory Environment. *Distributed Parallel Databases* 1, 1 (1993), 103–128. https://doi.org/10.1007/BF01277522

[53] Li Da Xu, Wu He, and Shancang Li. 2014. Internet of Things in Industries: A Survey. *IEEE Trans. Ind. Informatics* 10, 4 (2014), 2233–2243. https://doi.org/10.1109/TII.2014.2300753

[54] Steffen Zeuch, Ankit Chaudhary, Bonaventura Del Monte, Haralampos Gavriilidis, Dimitrios Giouroukis, Philipp M. Grulich, Sebastian Breß, Jonas Traub, and Volker Markl. 2020. The NebulaStream Platform for Data and Application Management in the Internet of Things. (2020). http://cidrdb.org/cidr2020/papers/p7-zeuch-cidr20.pdf

[55] Shuhao Zhang, Yancan Mao, Jiong He, Philipp M Grulich, Steffen Zeuch, Bingsheng He, Richard TB Ma, and Volker Markl. 2021. Parallelizing Intra-Window Join on Multicores: An Experimental Study. (2021).