B. Mitschang et al. (Hrsg.): BTW 2017 – Workshopband,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 335

Energy Efficiency in Main-Memory Databases

Stefan Noll[1]

Abstract: As the operating costs of today's data centres continue to increase and processor manufacturers are forced to meet thermal design power constraints, the energy efficiency of a main-memory database management system becomes more and more important. In this paper, we experimentally study the impact of reducing the clock frequency of the processor on the energy efficiency of common database algorithms such as scans, simple aggregations, simple hash joins and state-of-the-art join algorithms. We stress the fundamental trade-off between peak performance and energy efficiency, as opposed to the established race-to-idle strategy. Ultimately, we learn that database workloads behave considerably different than other workload types and that reducing the computing power e.g. by limiting the clock frequency can significantly improve energy efficiency.

Keywords: main-memory database systems, energy efficiency, DFVS.

1 Introduction

While a lot of current research in the domain of *main-memory database systems* (MMDBs) focuses on improving performance, only a small part of the research community focuses on improving energy efficiency. However, the energy consumption of a main-memory database system makes for an interesting research objective for several reasons. First, reducing the energy consumption also means reducing the operating costs, because the system needs less power and cooling. As electricity costs increase due to the growing demand for computing power, it is expected that the trend towards increased operating and cooling costs will intensify in the near future [PN08].
In addition, energy efficiency is a major aspect for chip design. In fact, semiconductor engineers have to meet fixed *thermal design power* (TDP) constraints. This means for example that not all cores of a processor can be fully powered at the same time without damaging parts of the integrated circuit. As a result, underutilised areas of the chip must be kept powered off (dark) or have to be operated at a low voltage and clock frequency, which is referred to as the Dark Silicon problem [Es11].
The aim of this work is to experimentally study as well as to understand the relationship between energy consumption and software to improve energy efficiency. We show that most algorithms used in a MMDB differ from algorithms used in other software. In fact, most database algorithms are memory-bound instead of compute-bound. As a result, unused computing power as well as electrical power gets wasted during the execution of database algorithms. Therefore, we propose to reduce the unused computing power of a processor for example by lowering the clock frequency and voltage which decreases the power consumption of a processor.

[1] TU Dortmund University, Databases and Information Systems Group, Otto-Hahn-Straße 14, 44227 Dortmund, stefan.noll@cs.tu-dortmund.de

We conduct a series of experiments with a compute-bound microbenchmark, with different memory-bound microbenchmarks simulating database algorithms as well as with two state-of-the-art join algorithms and analyse the impact of a limited clock frequency on the energy efficiency of the different workloads. In particular, we stress the fundamental trade-off between peak performance and energy efficiency, as opposed to the established race-to-idle strategy, which is usually used to maximise energy efficiency. Ultimately, we show that reducing unused computing power significantly improves the energy efficiency of memory-bound database algorithms.
In summary, we make the following contributions:

- We develop microbenchmarks simulating a compute-bound algorithm and different memory-bound algorithms.
- We experimentally study the impact of a limited clock frequency on the energy efficiency of the microbenchmarks and two state-of-the-art join algorithms.
- We use the results of the experiments to propose concepts for energy-aware software intended to improve the energy efficiency of a MMDB.

2 Basics

In this section we focus on the characteristics of database workloads. In addition, we introduce the power management features, which we use in the design of the experiments as well as for the reasoning in the course of the paper.

2.1 Characteristics of Database Workloads

Most database operations of a MMDB such as scans, aggregations or joins heavily depend on the memory bandwidth of the computer system. Thus, database algorithms are usually more memory-intensive than compute-intensive. In fact, most database algorithms are typically memory-bound or more specifically bandwidth-bound or latency-bound, because they perform lots of (random) memory accesses but only a few computations.
If an algorithm is memory-bound, the CPU stalls. It has to wait for data from the main memory until it can proceed with the execution of the algorithm. As a consequence, a lot of clock cycles are wasted. However, wasting computing power also means wasting electric power. Consequently, we argue that a high amount of computing power is not needed if an algorithm is memory-bound. To improve the energy efficiency of a database algorithm we propose to carefully reduce unused computing power.

2.2 Power Management Features

The *Advanced Configuration and Power Interface* (ACPI) defines several power management features of a modern computer system. This includes power and performance states

a computer system can enter to manage power consumption. In fact, the standard specifies two groups of states for a processor: the *processor power states* (C-States) and the *processor performance states* (P-States).
C-States describe the capability of an idle processor to save power by powering down unused parts of the processor. When individual processor cores or entire packages are idle, the operating system can put the hardware in a low-power state. Therefore, C-States are used to implement the **race-to-idle** strategy. Race-to-idle means executing a task as fast as possible and subsequently putting the processor in a sleep mode to save power.
P-States define the capability of an active processor to save power by lowering the clock frequency and voltage. A more common term used to describe this concept is *dynamic frequency and voltage scaling* (DFVS). In fact, a P-State can be interpreted as a pair consisting of a clock frequency and voltage. Higher (numerical) P-States result in slower processing speeds but also in a lower power consumption. The lowest P-State refers to the maximum clock frequency using turbo mode (e.g. "Turbo Boost" by Intel).

3 Experimental Design

In this section we present the experimental design. First, we introduce the database algorithms used in the experiments: one compute-bound microbenchmark, four memory-bound microbenchmarks and two state-of-the-art join algorithms. Afterwards, we introduce the hardware platform used to execute the experiments. Following this, we describe the measurement metrics and the experiments.

3.1 Benchmarks

The microbenchmark `compute` is designed to simulate a compute-bound algorithm. It induces a heavy load situation on all processor cores but avoids any accesses to the main memory. The benchmark creates several worker threads that spin on calculating the square root of a random number and on performing a multiplication.
The other four microbenchmarks are designed to simulate memory-bound, read-intensive database algorithms. They process integers with a size of 64 bits. The benchmarks `scan` and `local_scan` feature a sequential memory access pattern and resemble a scan or a simple aggregation. They create several worker threads that read a different section of a shared data array containing randomly generated integers (500 MB per thread). Each thread adds up all the integers of its section and prints the sum. In addition, we let each thread execute its work ten times to prolong the execution time in order to reduce variations and confounding factors.
The benchmarks `dira` and `local_dira` feature a *data-independent random* memory *access* (DIRA) [Ba14] pattern and resemble a simple hash join. The memory access pattern is called data-independent, because the processor can perform a random access without knowing the result of the previous random access. Each thread reads a different section of a shared data array containing randomly generated integers (500 MB per thread). These integers are used as indices to randomly access a second shared data array (8 GB).

We implement the microbenchmarks in `C++` using the program library `libnuma` to allocate the input data on specific NUMA nodes. The data is either allocated on one NUMA node only (`scan` and `dira`) resulting in both local and remote memory accesses or equally distributed across both NUMA nodes (`local_scan` and `local_dira`) resulting in local memory accesses only. We execute the microbenchmark using 32 threads.
In addition, we implement two benchmarks featuring two state-of-the-art join algorithms by using the program code of Balkesen et al. [Ba13]. We use the implementation of the optimised parallel radix hash join and the implementation of the NUMA-aware sort-merge join with either the multi-way or the multi-pass merging strategy. We slightly modify the source code to include the measurement metrics described in Section 3.3. We execute the algorithms using 16 threads and join two relations with a size of 4 GB each.

3.2 Hardware Platform

The hardware platform is configured as a dual-socket machine. It has two Intel Xeon E5-2690 processors featuring 8 processor cores (per socket), which are clocked at a base frequency of 2.9 GHz. Using turbo mode the processor can theoretically achieve a maximum clock frequency of up to 3.8 GHz. Furthermore, the system has 32 GB of DDR3 main memory per socket. The maximum memory bandwidth amounts to 51.2 GB/s for one socket or to 102.4 GB/s for both sockets.

3.3 Measurement Metrics

First, we determine the **execution time** of a benchmark to evaluate the performance. The execution time refers to the time each benchmark needs to process all of the input data. It does not include the time it takes to generate the data.
Second, we measure the total **energy consumption** of both processors by using the energy estimations provided by the *Running Average Power Limit* (RAPL) interface. The energy estimations refer to the energy consumption of the entire processor die including the core and uncore parts of the processor but excluding the attached DRAM.
Third, we compute the **average power consumption** of both processors by dividing the energy consumption by the execution time. In addition, we use an external power meter[3], which we plug between the power supply socket and the power supply unit to measure the power consumption of the entire computer system.
Fourth, we measure the **memory read bandwidth** by accessing hardware performance counters [WDF16] which provide information about the amount of bytes that are read from the memory controller of both sockets. Note that this includes additional memory traffic caused by cache misses if a new cache line has to be fetched but not all the data of the cache line is used by the program.
We do not use a specific metric to evaluate the *energy efficiency*. Instead, we argue that we improve the energy efficiency of a benchmark if the percentage decrease of the energy consumption is more significant than the percentage decrease of the performance. The point

[3] ELV Energy Master Basic 2 Energiekosten-Messgerät

of comparison is the measurement data from the execution at the highest clock frequency of 3.8 GHz, which is the default setting of the operating system.

3.4 Experiments

We conduct experiments to study the impact of a reduced computing power using the example of lowering the clock frequency of the processor. In fact, we limit the maximum clock frequency the Intel P-State driver of the Linux kernel `intel_pstate` is able to select. The driver can still select a clock frequency which lies below the limit but not above the limit. We execute every benchmark at different clock frequency limits from 1.2 GHz to 3.8 GHz while measuring the metrics described in Sect. 3.3.

4 Evaluation

Before we start evaluating the energy efficiency of the benchmarks, we first take a look at the accuracy of the energy estimations provided by the RAPL interface. Using the example of the microbenchmark `compute`, we compare the power estimations of both processors with the measurement data obtained from an external power meter. Figure 1a illustrates the experimental results. We observe that the two curves differ in approximately 65 W. Only in the frequency range of the turbo mode the two curves differ in up to 85 W. Thus, we argue that the energy estimations are reliable enough to evaluate the energy efficiency of the benchmarks. Furthermore, we verify that reducing the power consumption of both processors significantly improves the energy balance of the entire computer system. In addition, we notice that the curves break at 3.4 GHz. That is because the processor is not able to achieve a higher clock frequency using all cores due to thermal constraints.
Next, we evaluate the impact of a limited clock frequency on the energy efficiency of the benchmarks. The experimental results of the microbenchmark `compute` depicted in Fig. 1b reveal that the performance of the benchmark heavily depends on the clock frequency. Limiting the clock frequency to e.g. 1.9 GHz decreases the performance by more than 42%. At the same time the energy consumption only degrades by 26%. Thus, we observe that limiting the clock frequency does not improve the energy efficiency. Instead, it seems best to clock the processor at the highest frequency, finish the tasks as fast as possible and then put the processor in a sleep mode to save power (race-to-idle). Furthermore, we verify that the benchmark is only compute-intensive because the measured memory read bandwith is approximately 0 GB/s (cf. Table 1).
Figure 2a and Figure 2b illustrate the experimental results of the microbenchmarks `scan` and `dira`. We notice that limiting the clock frequency to e.g. 1.9 GHz only results in performance degradations of up to 4%. However, the energy consumption is reduced by 47%. Thus, limiting the clock frequency improves the energy efficiency of both benchmarks.
Moreover, we notice that the microbenchmarks `scan` and `dira` achieve a memory read bandwidth of up to 30 GB/s. The value lies below the hardware limit of 51 GB/s per socket. In fact, we determine that both benchmarks are limited by the bandwidth of the QPI links which connect both sockets. That is because all threads executed on the second

Benchmark Name	Execution Time [s]	Energy Consumption [J]	Memory Read Bandwidth [GB/s]
`compute`	16.1 / 5.9	956 / 1202	0.0 / 0.0
`scan`	6.6 / 5.4	455 / 973	24.3 / 30.2
`local_scan`	3.5 / 1.7	268 / 452	45.0 / 90.4
`dira`	5.5 / 4.5	387 / 837	25.5 / 31.2
`local_dira`	3.3 / 1.9	248 / 465	42.0 / 71.5
Parallel Radix Hash Join	4.3 / 2.5	295 / 448	15.4 / 24.8
Sort-Merge Join (multi-way)	8.5 / 3.7	611 / 842	9.9 / 21.9
Sort-Merge Join (multi-pass)	10.0 / 5.6	744 / 1233	18.5 / 32.3

Tab. 1: Execution time, energy consumption and memory read bandwidth of every benchmark at the lowest and highest clock frequency limit: 1.2 GHz / 3.8 GHz.

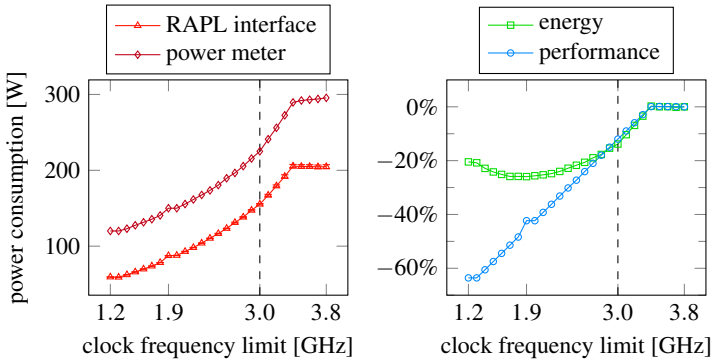

Fig. 1: Results of the benchmark `compute`: comparison of the energy estimations provided by the RAPL interface with the data obtained from an external power meter (a) and percentage decrease of the energy consumption and the runtime performance at different clock frequency limits (b). The dashed line marks the start of the turbo mode.

socket perform remote memory accesses to the input data which is located on the first socket. As a result, the execution slows down considerably.

Figure 3a and Figure 3b illustrate the results of the microbenchmarks `local_scan` and `local_dira`. We observe that limiting the clock frequency to e.g. 1.9 GHz saves more than 40% of the energy. At the same time we lose 20% of the performance. Thus, limiting the clock frequency improves the energy efficiency of both benchmarks. Furthermore, we learn that the microbenchmark `local_scan` reaches a memory read bandwidth of up to 90 GB/s (cf. Table 1), which is very close to the hardware limit of 102.4 GB/s for both sockets. In fact, the benchmark is bandwidth-bound. The microbenchmark `local_dira` on the other hand only reaches a memory read bandwidth of up to 72 GB/s (cf. Table 1). We conclude that the benchmark is latency-bound, i.e. by the time it takes to transfer data from the main memory to the processor.

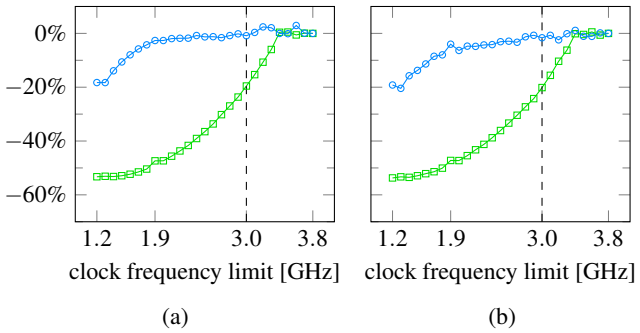

Fig. 2: Percentage decrease of the energy consumption and the runtime performance of the benchmarks `scan` (a) and `dira` (b) at different clock frequency limits.

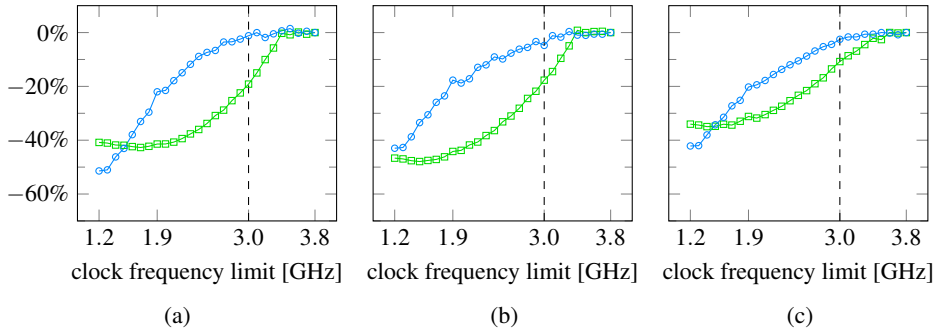

Fig. 3: Percentage decrease of the energy consumption and the runtime performance of the benchmarks `local_scan` (a), `local_dira` (b) and the parallel radix hash join (c) at different clock frequency limits.

Figure 3c depicts the measurement results of the parallel radix hash join. We learn that limiting the clock frequency to e.g. 1.9 GHz degrades the performance by 20%. At the same time the energy consumption is reduced by 31%. Thus, limiting the clock frequency improves the energy efficiency of the parallel radix hash join, too.

Figure 4a and Figure 4b illustrate the results of the sort-merge join algorithm using the multi-way and the multi-pass merging strategy. We observe that in case of the multi-way merging, limiting the clock frequency to e.g. 1.9 GHz deteriorates the performance by 34% while the energy consumption is reduced by 28%. In case of the multi-pass merging, limiting the clock frequency to 1.9 GHz degrades the performance by 21%. At the same time we save 36% of the energy. Thus, limiting the clock frequency significantly improves the energy efficiency of the sort-merge join algorithm if the multi-pass merging is used.

In order to explain the results we have to visualise the difference between the two algorithms. Figure 4c illustrates the execution times of every phase of the sort-merge join algorithm using either multi-way or multi-pass merging. We learn that the partitioning phase, the sorting phase and the joining phase all take approximately the same amount of

time. However, we observe that the merging phase lasts significantly longer if the algorithm uses the multi-pass merging technique.
Considering that the in-cache sorting is compute-intensive while merging is memory-intensive, it is plausible that the energy efficiency of the sort-merge join algorithm using multi-way merging does not improve. The algorithm is mainly compute-intensive. Thus, it heavily depends on the clock frequency of the processor. The sort-merge join using the multi-pass merging on the other hand is mainly memory-intensive. Therefore, we can reduce unused computing power of the processor to improve the energy efficiency.

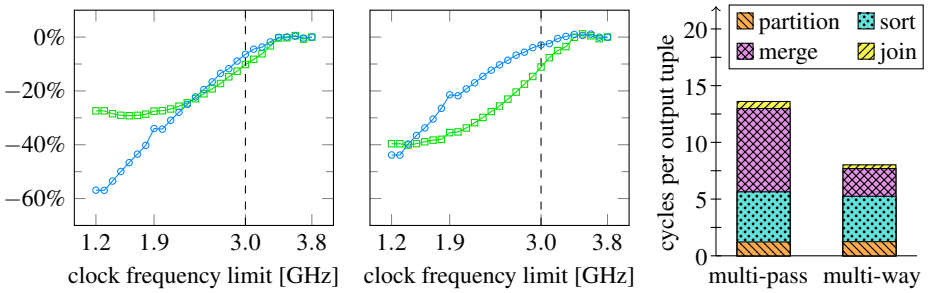

Fig. 4: Percentage decrease of the energy consumption —□— and the runtime performance —○— of the sort-merge join using either the multi-way merging strategy (a) or the multi-pass merging strategy (b) while limiting the clock frequency. Breakdown of the execution times (c) of the different phases of the sort-merge join algorithm using either multi-way merging or multi-pass merging [Ba13].

Discussion. Our evaluation reveals that the energy efficiency of the microbenchmark `compute` does not improve if we limit the clock frequency. We conclude that the most energy-efficient execution strategy for a compute-bound algorithm is the race-to-idle strategy. In contrast, we learn that limiting the clock frequency improves the energy efficiency of the memory-bound microbenchmarks.
If the input data is allocated on only one NUMA node, reducing the clock frequency hardly degrades the performance. If the input data is equally distributed across all NUMA nodes, limiting the clock frequency has a bigger impact on the performance. However, the energy savings prevail. Moreover, we observe that the impact of a limited clock frequency does not differ between a sequential memory access pattern (bandwidth-bound) and a random memory access pattern (latency-bound). In addition, we learn that reducing the clock frequency improves the energy efficiency of the parallel radix hash join, too.
In case of the sort-merge join we observe that the impact of a limited clock frequency depends on the merging strategy. If we use the multi-way merging, the algorithm is mostly compute-intensive. Reducing the clock frequency does not improve the energy efficiency. If we use the multi-pass merging, the algorithm is mostly memory-intensive. Limiting the clock frequency improves the energy efficiency.
Therefore, we conclude that we can improve the energy efficiency of a database algorithm by reducing the computing power if the algorithm is more memory-intensive than

compute-intensive. We showed that we can for example limit the clock frequency to improve the energy efficiency of database algorithms: If we execute a *compute-bound* algorithm, we should *increase* the *clock frequency* to the maximum (race-to-idle). If we execute a *memory-bound* algorithm, we should *decrease* the *clock frequency*.

Towards Energy-Efficient Software. The results of the evaluation reveal that we can reduce unused computing power to improve the energy efficiency of database algorithms. This allows us to develop ideas for energy-efficient software used in a MMDB.
First, the results can influence the design of database algorithms. If an algorithm consists of a compute-bound phase and a memory-bound phase, we propose to balance both phases in order to avoid energy waste. In case of e.g. the sort-merge join algorithm, we could overlap the compute-bound sorting phase with the memory-bound merging phase.
Second, we propose to extend the optimiser of a MMDB to avoid energy waste. When the optimiser decides upon the physical database operations of an execution plan, it should balance compute-bound operations and memory-bound operations. Thus, the optimiser needs specific cost models. These can be built based on the ratio between compute intensity and memory intensity of a database operation.
Third, we propose to extend the execution engine of a MMDB by a live monitoring component. The engine could use hardware performance counters to periodically calculate the ratio between compute intensity and memory intensity of database operations at runtime. This can be done on multiple levels: for the entire system, individual sockets or individual cores. The collected statistics can be used to dynamically change settings such as the clock frequency, thread-to-core mappings or the data placement.

5 Related Work

In this section we briefly present related work from the research community specialising on the topic of energy efficiency in DBMS. Harizopoulos et al. [Ha09] argue that hardware optimisations are only part of the solution towards improving energy efficiency. They predict that software will be the key to improve energy efficiency and propose to focus on system-wide configuration parameters and query optimisations. In fact, they propose to consolidate resource utilisation and power in order to facilitate powering down unused hardware and to redesign data-structures and algorithms. Thus, their observations align with our own results.
Tu et al. [Tu14] propose a storage manager which puts unused disk drives in a low power mode and employs energy-aware data placements. In addition, they present a prototype DBMS, which uses a feedback loop using DFVS based on the current performance and a performance threshold. They claim to achieve significant energy savings.
The work of Psaroudakis et al. [Ps14] further supports our results. They propose a fine granular approach for improving energy efficiency by dynamically scheduling tasks as well as using DVFS. They conclude that using simultaneous multithreading, a moderate clock frequency and a thread placement that fills both sockets can significantly improve the energy efficiency of memory-intensive workloads. In addition, they propose to use a memory allocation policy that uses the memory bandwidth of all available sockets.

6 Conclusion

The characteristics of typical database workloads matter. In fact, most database algorithms are memory-bound instead of compute-bound. As a result, computing power as well as electrical power gets wasted if the processor has to wait for data from the main memory. We experimentally explored the impact of reducing the computing power on the energy efficiency of several benchmarks representing common database algorithms such as scans, simple aggregations and joins. Our evaluation reveals that limiting the clock frequency deteriorates the energy efficiency of the compute-bound microbenchmark, while it improves the energy efficiency of the memory-bound benchmarks including state-of-the-art joins. We show that database workloads behave considerably differently than other workload types and that reducing the computing power e.g. by limiting the clock frequency can significantly improve energy efficiency.

Acknowledgments

This work was supported by the DFG, Collaborative Research Center SFB 876, A2.

References

[Ba13] Balkesen, C.; Alonso, G.; Teubner, J.; Özsu, M. T.: Multi-Core, Main-Memory Joins: Sort vs. Hash Revisited, Proc. VLDB Endow. 7/1, pp. 85–96, 2013.

[Ba14] Barber, R.; Lohman, G.; Pandis, I.; Raman, V.; Sidle, R.; Attaluri, G.; Chainani, N.; Lightstone, S.; Sharpe, D.: Memory-Efficient Hash Joins, Proc. VLDB Endow. 8/4, pp. 353–364, 2014.

[Es11] Esmaeilzadeh, H.; Blem, E.; St. Amant, R.; Sankaralingam, K.; Burger, D.: Dark Silicon and the End of Multicore Scaling, SIGARCH Comput. Archit. News 39/3, pp. 365–376, 2011.

[Ha09] Harizopoulos, S.; Shah, M. A.; Meza, J.; Ranganathan, P.: Energy Efficiency: The New Holy Grail of Data Management Systems Research. In (): Online Proc. CIDR, 2009.

[PN08] Poess, M.; Nambiar, R. O.: Energy Cost, the Key Challenge of Today's Data Centers: A Power Consumption Analysis of TPC-C Results, Proc. VLDB Endow. 1/2, pp. 1229–1240, 2008.

[Ps14] Psaroudakis, I.; Kissinger, T.; Porobic, D.; Ilsche, T.; Liarou, E.; Tözün, P.; Ailamaki, A.; Lehner, W.: Dynamic Fine-grained Scheduling for Energy-Efficient Main-memory Queries. In (): Proc. DaMoN, ACM, 1:1–1:7, 2014.

[Tu14] Tu, Y.; Wang, X.; Zeng, B.; Xu, Z.: A System for Energy-Efficient Data Management, SIGMOD Rec. 43/1, pp. 21–26, May 2014.

[WDF16] Willhalm, T.; Dementiev, R.; Fay, P.: Intel Performance Counter Monitor - A better way to measure CPU utilization, 2016, URL: http://www.intel.com/software/pcm, visited on: 08/01/2016.