Dynamic Fine-Grained Scheduling for Energy-Efficient Main-Memory Queries

Iraklis Psaroudakis[⋆‡] Thomas Kissinger[†] Danica Porobic[⋆] Thomas Ilsche[†]
Erietta Liarou[⋆] Pınar Tözün[⋆] Anastasia Ailamaki[⋆] Wolfgang Lehner[†]

[⋆]EPFL
{first.last}@epfl.ch

[†]TU Dresden
{first.last}@tu-dresden.de

[‡]SAP AG
{first.last}@sap.com

ABSTRACT

Power and cooling costs are some of the highest costs in data centers today, which make improvement in energy efficiency crucial. Energy efficiency is also a major design point for chips that power whole ranges of computing devices. One important goal in this area is energy proportionality, arguing that the system's power consumption should be proportional to its performance. Currently, a major trend among server processors, which stems from the design of chips for mobile devices, is the inclusion of advanced power management techniques, such as dynamic voltage-frequency scaling, clock gating, and turbo modes.

A lot of recent work on energy efficiency of database management systems is focused on coarse-grained power management at the granularity of multiple machines and whole queries. These techniques, however, cannot efficiently adapt to the frequently fluctuating behavior of contemporary workloads. In this paper, we argue that databases should employ a fine-grained approach by dynamically scheduling tasks using precise hardware models. These models can be produced by calibrating operators under different combinations of scheduling policies, parallelism, and memory access strategies. The models can be employed at run-time for dynamic scheduling and power management in order to improve the overall energy efficiency. We experimentally show that energy efficiency can be improved by up to 4x for fundamental memory-intensive database operations, such as scans.

1. INTRODUCTION

Dynamic power costs comprise a major part of overall costs in modern large scale data centers [7]. As costs of static power infrastructure decrease, the fraction of dynamic power costs is expected to increase in the near future, and companies continuously strive to optimize them. As data management applications are fundamental data center workloads, energy efficiency of the DBMS in general is crucial [20]. One of the major goals in this area is to achieve energy proportionality, that argues that power consumption of the system should be proportional to its current load.

At first glance, using low-power processor components or computing nodes, which can be turned on and off depending on the workload, is an appealing option for energy proportionality. Recent studies, however, show that commercial DBMS achieve more performance per unit of power when running on more powerful servers [24]. These findings are in line with the observations of data center operators and studies of energy efficiency of data center workloads [10].

Faced with limitations in the manufacturing processes and the inability to efficiently cool all transistors on a chip [1], processor vendors are turning towards dynamic management of processor resources, especially power. Common techniques include turbo mode, a hierarchy of sleep states, and dynamic voltage and frequency scaling.

In this paper, we experimentally assess the impact of these power features on the performance and energy consumption of common memory-intensive database operations such as scans and aggregations. We build energy efficiency curves (correlating performance with energy) under various power configurations, scheduling policies, levels of parallelism, and memory access patterns. We argue that DBMS can follow a similar approach to calibrate the energy efficiency of database operations dynamically at run-time to improve energy efficiency and achieve energy proportionality.

Contributions. Our main contributions are:

- We identify the features of modern processors that can be used to dynamically adjust power at run-time in order to improve energy efficiency of database workloads.
- We experimentally show that energy efficiency can be significantly improved (by up to 4x) by tuning processor power features, thread scheduling, parallelism, and memory access patterns.
- We outline a promising research direction toward power-aware DBMS employing calibration, dynamic resource monitoring, and scheduling to improve energy efficiency.

Paper organization. Section 2 presents power control features of modern processors and discusses related work in the data center community. We survey related work in the database community in Section 3. Section 4 argues for run-time power-aware scheduling, and describes our methodology for

DaMoN'14, June 22-27 2014, Snowbird, UT, USA
Copyright is held by the owner/author(s). Publication rights licensed to ACM. ACM 978-1-4503-2971-2/14/06...$15.00.
http://dx.doi.org/10.1145/2619228.2619229.

producing calibration curves for analytical operations. We show our experimental evaluation in Section 5, which includes calibration curves with a sensitivity analysis. Finally, Section 6 concludes the paper and outlines future work.

2. ENERGY EFFICIENCY BACKGROUND

Energy efficiency can be considered from (at least) two perspectives: small-scale and large-scale. In the large-scale, i.e., in the context of a data center or a compute cluster, energy efficiency is very often achieved by elasticity at a node level. For example, Schall and Härder [22] demonstrate energy proportionality by efficiently partitioning and migrating database fragments. Also, Chen et al. [2] have identified inefficiencies in running data analytics on Hadoop and proposed various policies to turn off servers. However, power cycling decreases endurance of power supplies significantly and entails costs for starting up the system in terms of latency and energy consumption. Thus, a much better approach is to find more work or to lower the frequency. Additional work can be found with more precise scheduling of non-interfering workloads [3].

In the small-scale, processor vendors are turning toward dynamic power management of processors, because they increasingly face power and scaling limitations. Dynamic Voltage and Frequency Scaling (DVFS) is the most popular way to improve energy efficiency, by using the ACPI P-states to adjust processor frequencies. Lowering frequencies allows lowering the voltage, which results in significantly decreased power consumption. Although frequencies can be defined at the operating system (OS) level with respect to individual cores, processor vendors currently only allow adjustments at a coarser granularity. For example, Intel processors based on Nehalem and Sandy Bridge architectures keep all cores on the same processor at the same P-state.

A special form of DVFS that has become popular with recent processors is turbo mode (marketed as Turbo Boost by Intel and Turbo Core by AMD). Turbo mode is a special case of computational sprinting [21]. It boosts frequency and voltage (frequency sprinting) of a few cores, with a frequency potentially beyond the nominal one, depending on the power and thermal budget and the number of idle cores. This scheme is used primarily to boost the performance of single threaded workloads and is typically automatically controlled by the processor. It can, however, be disabled from the user level applications through OS interfaces, or by lowering the processor frequency below the maximum.

Another energy efficiency feature of modern processors are the idle states. The ACPI standard describes C-states that turn off some processor features to lower the power consumption. Although current processors implement a more fine-grained hierarchy of C-states compared to the standard, their use is mostly reserved to the OS scheduler.

3. RELATED WORK

Recent studies in the area of energy efficiency of DBMS can be divided into three broad categories: (a) analysis of energy proportionality of clusters, (b) measuring energy efficiency of operators and analytical queries, and (c) making query optimizers aware of a query's energy requirements.

Choosing wimpy or beefy nodes for data center workloads has been a hot topic in systems communities. In the database community, a number of recent studies with distributed and analytical workloads have examined different trade-offs between various types and numbers of nodes in the cluster [12, 13, 14, 22]. The principal motivation is that while achieving energy proportionality of a single server is hard, it would be more realistic to achieve it when work is distributed among multiple nodes. The common conclusion is that current distributed database designs perform better on beefy nodes since operators and schedulers are oblivious to energy requirements. A load balancing subsystem that is aware of the energy cost for processing different parts of a query on different nodes and the cost of potentially transferring data among nodes can make better decisions and come closer to achieving energy proportionality [22].

On a single server, a number of studies have analyzed energy efficiency of different system configurations and both individual operators and TPC-H queries [9, 24, 26]. Their goal is to find a way to minimize energy costs per unit of work and the most common approach is to save energy by lowering processor voltage and frequency at the expense of query response time [13]. An extensive study [24] of energy efficiency of database software suggests that for a single node server system, the most energy-efficient configuration is the highest performing one. Authors of this study have performed a range of experiments with varying machine configurations and observed that for the same CPU utilization, power consumption for different database operators can vary as much as 60%. In this paper, we go a step further by analyzing potential power savings with dynamic frequency scaling and fine-grained power-aware scheduling at run-time at a granularity of threads and operators in the query plan.

The most natural place to add energy efficiency awareness to a DBMS is the query optimizer. One recent project augments the operator performance models in PostgreSQL with information about the energy cost of processing tuples [26, 27]. By changing the query optimization criteria, the energy cost of TPC-H queries has been lowered by 19% without significantly increasing the query response times.

Another approach for general systems is Dynamic Concurrency Throttling (DCT) which aims to reduce the number of active threads during parallel regions, that are constrained by shared resources, to reduce energy consumption [16]. Such general purpose approaches for energy-efficient scheduling typically focus on a single set of parameters that is adjusted beforehand or at run-time, using a limited set of performance metrics. They use data mining tools to generate a model and select the most appropriate configuration at run-time. The main problem is that they rely on the predictability of observed metrics over time to compensate for the tracing overhead and system reconfiguration. For this reason, they cannot be used efficiently on DBMS applications, which might need to handle unpredictable workloads and manifest multiple behaviors over a short period of time.

A more radical approach to energy efficiency is the specialization at the hardware level by creating ASIC or FPGA chips for accelerating common operations [6, 11, 19, 25]. With specialized hardware we can achieve an order of magnitude better efficiency, but at the price of high design costs and inflexibility. As the era of Dark Silicon [8] looms, we will not be able to power the whole chip. Having specialized circuits to use on demand can be very appealing. From the software side, we propose a similar notion: specializing the scheduling scope of the DBMS for a fine-grained dynamic scheduling approach to improve energy efficiency.

4. POWER-AWARE SCHEDULING

Both general purpose and database specific proposals for improving energy efficiency of DBMS fail to achieve a lot of potential energy savings. General purpose optimization tools treat applications as black boxes and optimize them by measuring specific performance counters externally. DBMS, however, pose a significant challenge to general purpose approaches. They are heavily multi-threaded applications whose memory access behavior changes over time for different database operations and can range from pure independent streaming to high contention. Thus, an approach, that utilizes internal DBMS knowledge to quickly assess resources, including caches, memory controllers, and socket inter-connects, and enhances energy efficiency using the power features of modern processors, seems more appropriate.

Database specific approaches, on the other hand, suggest optimization of large parts of the workload using coarse-grained changes to the hardware environment. Recent proposals argue for adding power consumption models to the traditional query optimizer and treating power either as an additional optimization goal or an optimization constraint (see Section 3). While this is a step in the right direction, additional energy savings can be achieved by careful thread scheduling and memory allocation policies that also tune processor configuration.

In this paper, we evaluate the potential of dynamic fine-grained power-aware scheduling of analytical workloads at run-time using precise hardware models. Our general strategy relies on query plans generated by a power-aware query optimizer and on a set of energy efficiency models based on power/performance metrics. These models can be produced by the DBMS beforehand by calibrating operators using a combination of parameters including different levels of parallelism, thread placement policies, and memory access strategies to improve energy efficiency. To assess the potential improvements, we produce energy efficiency calibration curves for fundamental building blocks for analytical queries: concurrent partitioned scans and a parallel aggregation. In the rest of this section, we detail our methodology.

Experimental methodology. We conduct two sets of experiments, one for concurrent partitioned scans and one for a parallel aggregation. Both use common tools for measuring energy, but have different experimental setups to explore different aspects of energy efficiency for memory-intensive operations. Below, we detail the common tools and then describe the experimental setup for each experiment.

Hardware counters. For both experiments, we use Intel's RAPL (Running Average Power Limit) for measuring energy consumed during the experiments. We use the following two domains of RAPL counters:

- *Package (PKG).* This metric includes the energy consumption of the entire package, including the core and the uncore parts, but excluding the attached DRAM.
- *DRAM.* This metric includes the energy consumption of the DRAM attached to a package.

The total energy consumed at the level of a CPU socket is calculated as the sum of the package and DRAM domains. In addition, we measure, during the experiments, the used memory bandwidth (traffic) of the sockets' DRAM [4].

Concurrent partitioned scans. Section 5.1 evaluates the energy efficiency of scans with different scheduling policies and processor frequencies. In our application (written in C++), each thread continuously scans 128 MB of 64-bit integers, allocated on the local memory node. Each run lasts 5 seconds and we measure the number of scans performed. We seek to maximize the following metric:

$$\text{performance per power} = \frac{\text{throughput (scans/sec)}}{\text{power (W)}}$$

The machine used in this experiment is composed of two 8-core Intel Xeon E5-2690 (Sandy Bridge-EP) 2.9 GHz processors, with 20 MB last-level cache, and hyper-threading (HT) enabled. The system has 64 GB of DDR3 RAM. Processors have different frequencies (or P-states) available, ranging from 1.2 to 2.9 GHz, which we vary. In our experiments, *Auto* refers to the configuration where the OS controls the frequency, plus the dynamic Turbo Boost (which can reach frequencies up to 3.8 GHz). The P-states also affect the voltage supplied to the processor's cores and their energy consumption when working.

In addition to using RAPL counters that have known limitations [5], we employ dedicated energy instrumentation. Specifically, we measure the total power consumption on the AC side of the Power Supply Unit (PSU) using a calibrated high accuracy power analyzer (ZES Zimmer LMG450). This device reports 20 samples per second but uses a much higher internal sampling rate in order to report accurate energy / average power. To avoid perturbation on the system under test, we process the measurements on a separate system. By comparing the high precision measurements to RAPL counters, we assess their accuracy and determine if there is additional significant energy consumption to consider.

Parallel aggregation. Section 5.2 demonstrates the energy efficiency of an aggregation under different parallelism, scheduling, and memory allocation configurations. We evaluate the performance of a variant of STREAM [17], a synthetic benchmark that measures sustainable memory bandwidth and computation rate for simple vector kernels.

Our variant builds two arrays b and c, and performs the aggregation: $a = \sum (b(i) + c(i))$. Elements are of *double* data type. In our experiments, we use 4 GB arrays. Our benchmark performs the operation 10 times (reducing standard deviation below 5%) and we measure the total response time of all iterations (excluding the allocation of the arrays).

Contrary to the previous set of experiments, in this experiment our goal is to decrease both the response time and energy consumption. We seek to minimize an energy delay product metric [15] that is commonly used in latency sensitive experiments:

$$\text{EDP} = \text{response time (sec)} * \text{energy (J)}$$

Furthermore, STREAM supports the OpenMP API [18] for parallelism, and we express the operation as:

```
#pragma omp parallel for reduction(+:a)
for (i=0; i< STREAM_ARRAY_SIZE; i++)
   a = a + b[i] + c[i];
```

The range is partitioned and distributed among the given number of threads. Thus, we can easily assess the energy efficiency for different levels of parallelism (i.e., number of threads) and scheduling policies. Moreover, we measure the effect of non-uniform memory access (NUMA) latencies on energy efficiency by varying memory allocation policies.

In this experiment we use the machine composed of two 8-core Intel Xeon E5-2640 v2 (Ivy Bridge-EP) 2.0 GHz processors, with a 20 MB last-level cache. The system has 256 GB of DDR3 RAM. Contrary to the previous experiment, we disable hyper-threading to minimize interference of threads running on the same core, which introduces variability.

5. EXPERIMENTAL RESULTS

This section summarizes the results of the experiments for concurrent partitioned scans and the parallel aggregation. We present calibration curves and discuss their implications.

5.1 Concurrent partitioned scans

In this experiment, we vary the number of concurrent threads, the scheduling policy, and the frequencies of the processors. We measure total throughput, energy consumption, and report the energy efficiency of each configuration.

5.1.1 Socket-fill scheduling

In this policy, we bind threads in a way to first fill socket 1 and then socket 2. We begin by binding threads to every core of socket 1, then to the HT sibling cores of socket 1, then to every core of socket 2, and finally to the HT sibling cores of socket 2. Figure 1 shows the results.

We can draw a number of observations from this experiment, some of which are shared with other scheduling policies. First, we note that the lines corresponding to measurements with RAPL counters and external equipment follow the same trends. The difference between the two remains constant for each frequency, irrespective of the number of threads used. Since we do not exercise non-processor parts of the system heavily (e.g., disks or network), the difference corresponds to the constant power consumed by the mainboard and other system components. RAPL counters provide a good approximation for the power consumed by sockets. For clarity, we do not include further measurements with external equipment. Thus, energy efficiency refers to sockets and DRAM instead of the entire system.

Second, we notice that for each frequency, the memory bandwidth gets saturated after a number of threads on each socket. We depict the average memory traffic of the sockets' DRAM in Figure 2. The saturation point is different for each frequency, since the maximum memory bandwidth of the memory controllers depends on the frequency [23]. To better illustrate the performance bottleneck caused by the saturation, we depict the throughput and power measurements for the case of 2.0GHz in Figure 3. Throughput flattens around 7 threads due to the memory bandwidth being saturated. Additional threads do not increase throughput, but, depending on the frequency, may hurt energy efficiency since they draw additional power. We notice also that the hyper-threads do not consume additional power.

A third implication is that, as shown in Figure 3, throughput is ultimately doubled, but the total power is not, due to the constant power of the second socket that we consider in the performance per power formula while we fill the

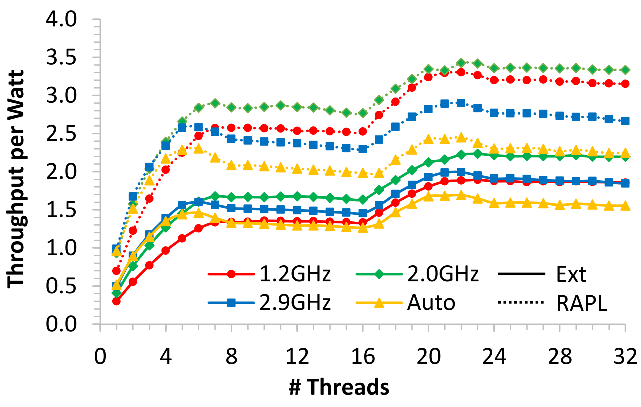

Figure 1: Measuring the energy efficiency, with RAPL counters and external equipment, for the socket-fill scheduling policy, while varying the number of threads and the processor frequency.

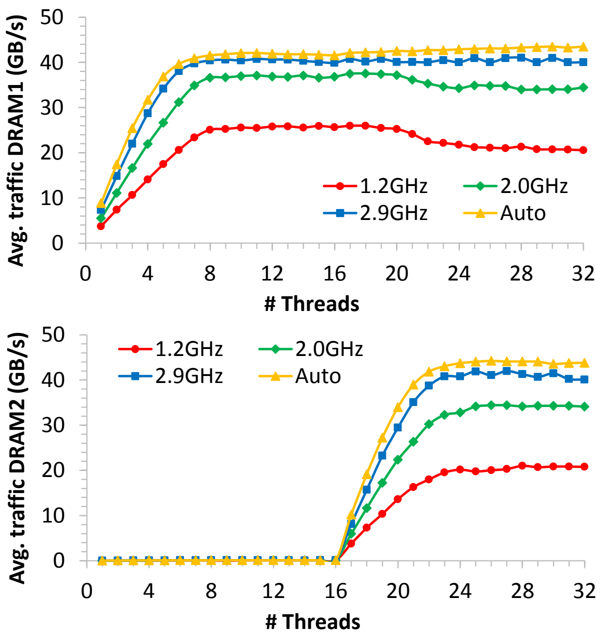

Figure 2: Average memory traffic of Figure 1.

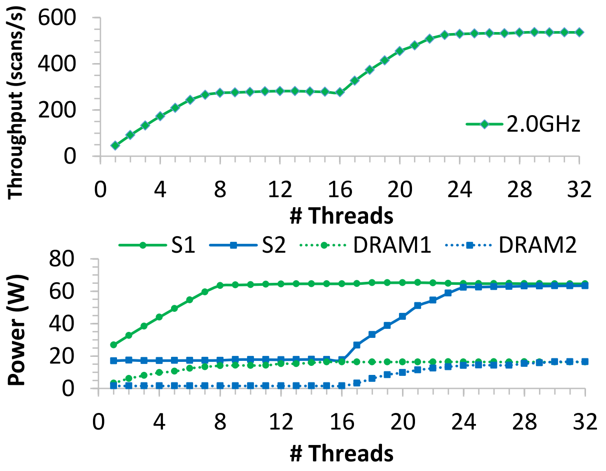

Figure 3: Throughput and RAPL counters breakdown of Figure 1 for the case of 2.0GHz frequency.

first socket. The last implication is that a lower frequency can achieve the best energy efficiency. The lowest frequency, 1.2GHz, cannot keep up with memory requests, but an intermediate frequency, 2.0 GHz, can keep up with the requests, while not powering up cores completely.

Overall, when we compare the most energy efficient configuration, i.e., using 23 threads at 2.0Ghz, we get 3.6x improvement (4.3x when comparing total machine power consumption) compared to the default OS-managed single-threaded configuration. It is also 1.4x more efficient than the most efficient OS-managed configuration (using 22 threads). Thus, the database can calibrate memory-bound operations on the specific hardware to calculate the best frequency and parallelism settings to use.

5.1.2 *Socket-fill HT scheduling*

This scheduling policy is similar to the socket-fill policy, with the exception that, before going to the next core, we bind the next thread to a HT sibling. In Figure 4, we show the results of this scheduling policy. The main implication is that hyper-threads can efficiently multiplex their memory requests with those of their sibling threads for memory-bound operations, increasing the throughput with negligible additional power (additional to their sibling threads).

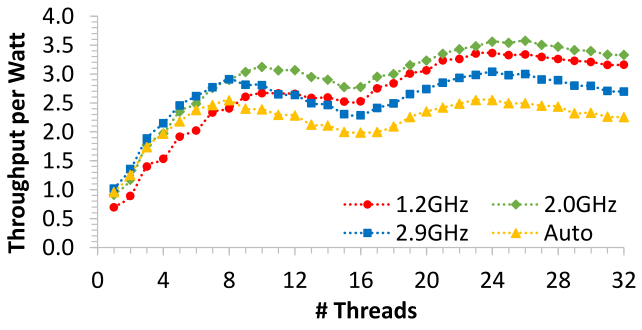

Figure 4: The socket-fill HT scheduling policy.

5.1.3 *Socket-wise scheduling*

The socket-wise scheduling policy corresponds to binding threads in a round-robin manner across the two sockets, first using all cores of both sockets and then using their HT siblings. The results are shown in Figure 5.

This policy gradually aggregates the memory bandwidth of both sockets, avoiding the early socket-specific memory bandwidth bottleneck of the previous policy. In this configuration, we saturate the memory bandwidth of both sockets as early as 12 threads (6 on each socket) at 2.0GHz. Additional threads do not increase throughput any more, but consume more power and decrease energy efficiency.

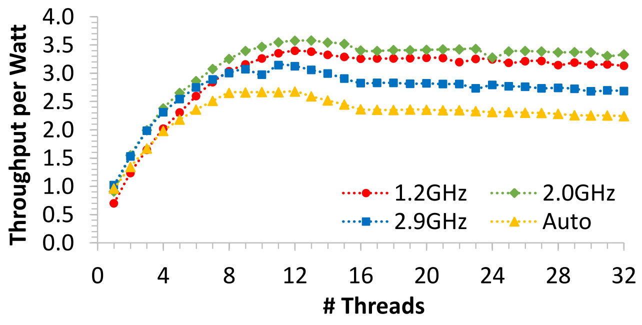

Figure 5: The socket-wise scheduling policy.

5.1.4 *Socket-wise HT scheduling*

This scheduling policy is similar to the previous one, with the exception that we bind the next thread to a HT sibling before going to the next core of the other socket. Results are shown in Figure 6.

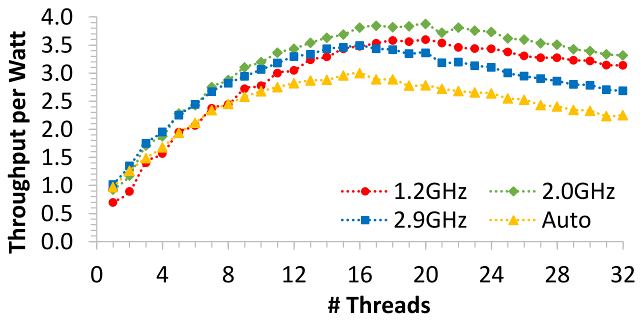

Figure 6: The socket-wise HT scheduling policy.

This scheduling policy is the most energy efficient at around 20 threads for 2.0GHz. After 20 threads, additional threads do not increase throughput but consume more power and decrease energy efficiency. This scheduling policy combines the best characteristics of the previous scheduling policies for memory-bound operations: it saturates the memory bandwidth gradually while using hyperthreads to increase throughput without significant additional power. The most energy-efficient configuration improves the best OS-managed single-threaded and multi-threaded configurations by 4x and 1.3x respectively, confirming that choosing the best scheduling policy does not diminish the potential positive impact of choosing an appropriate frequency.

Summary. Our experiments with parallel scans demonstrate that a higher frequency does not improve energy efficiency because of memory bandwidth saturation while a very low frequency is not energy efficient because it does not sufficiently saturate the memory bandwidth. Furthermore, a scheduling policy that distributes threads on both sockets and utilizes hyperthreads can improve energy efficiency by maximizing the use of available memory bandwidth without unnecessary power consumption.

5.2 Parallel aggregation

In this experiment, we vary three parameters: the number of threads used for the parallel version of our variant of the STREAM benchmark, the scheduling policy, and the memory allocation policy. Since we disable hyper-threading to minimize the interference of threads running on the same core, we use the socket-fill and socket-wise scheduling policies of Section 5.1 without HT. With respect to memory allocations, we assume that the data is already loaded in memory before the aggregation query arrives. It is, therefore, expensive for the aggregation to move the data. Under this assumption, we either allocate memory on the first socket (similar implications can be drawn if we allocate on the second socket), or interleave it across both sockets. We note that further memory allocation policies can be applicable for intermediate results, e.g., partitioning (the aggregation

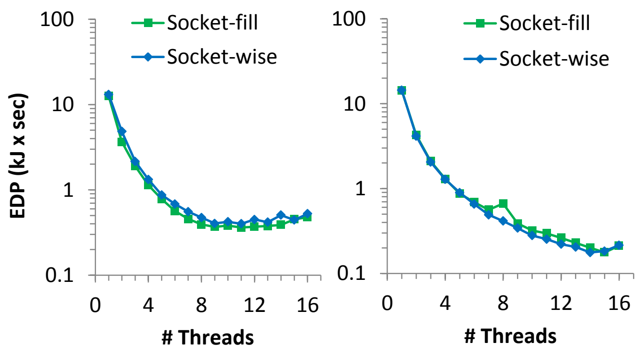

Figure 7: Energy delay product for both scheduling strategies, when the memory is allocated on the first socket (left) and when it is interleaved (right).

can then parallelize with locally-allocated partitions, with implications similar to Section 5.1), as the DBMS needs to place the intermediate results in memory.

The energy efficiency of all combinations of the parameters is shown in Figure 7. We also include detailed measurements (response time, average memory traffic, and energy consumption) for the best scheduling policies for each memory policy in Figure 8 (socket-fill scheduling when the memory is allocated on the first socket and socket-wise scheduling when the memory is interleaved).

We first discuss the case when the memory is allocated on the first socket. A first implication is that parallelism improves the energy efficiency of aggregations up to a point. The best combination of parameters that minimizes the EDP is achieved at around 9 threads, for the socket-fill scheduling policy. At this point, the memory bandwidth of the memory controller of the first socket is saturated (see Figure 8). Employing more threads just consumes more energy without improving response time, since we are limited by the memory bandwidth. Thus EDP increases with more threads.

The second case of memory allocation, i.e., interleaving, ultimately achieves better energy efficiency than the case of only allocating memory on the first socket. This is despite around half of the total memory accesses being remote, suffering a higher latency in NUMA architectures, and involving communication between the two sockets. At the same time, we are not constrained by the maximum memory bandwidth of the first socket, but can utilize the memory bandwidth of both sockets, as shown in the average memory traffic of Figure 8. Interleaving results in balanced use of resources of both sockets as shown by the energy and memory traffic graphs. We also note that socket-wise scheduling is better than socket-fill, because it balances threads across sockets. Specifically, the socket-wise policy avoids saturating the resources of a single socket, while the socket-fill strategy increases EDP around 8 threads when it fills the first socket.

Summary. The parallel aggregation experiments show that the most energy-efficient configuration involves a memory allocation policy that can utilize the memory bandwidth of all sockets and a scheduling policy that balances the threads across the sockets. In our experiments, interleaving memory and using the socket-wise scheduling can decrease the worst EDP (single-threaded execution) by an order of magnitude, which shows a significant improvement for energy efficiency.

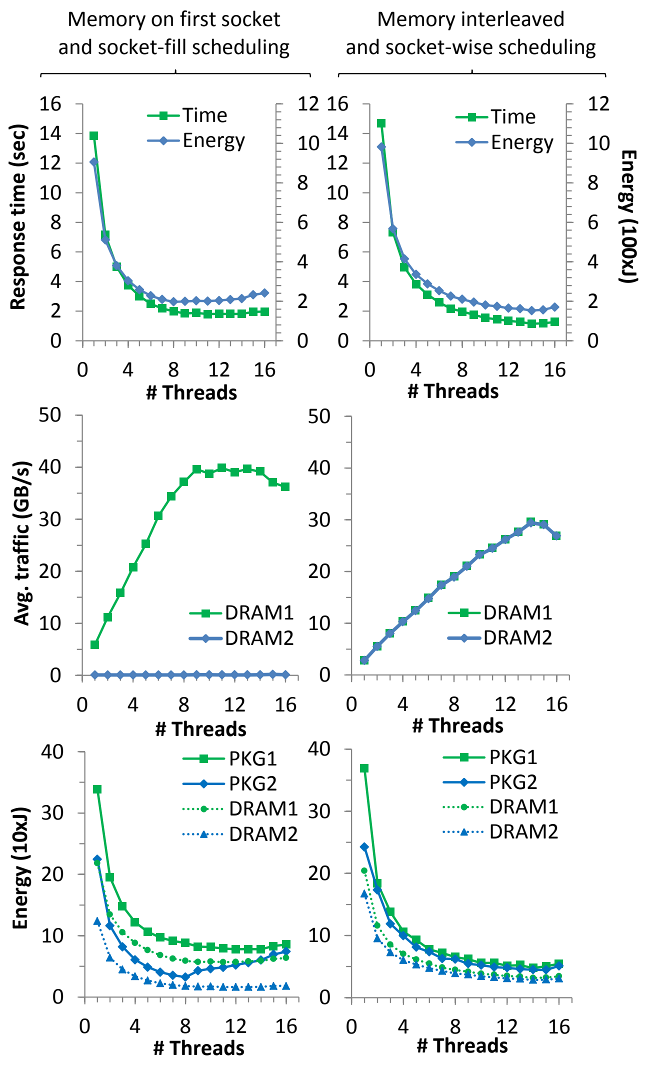

Figure 8: Detailed measurements for the best scheduling policy of each memory allocation policy.

6. CONCLUSIONS AND FUTURE WORK

Energy efficiency is currently a major issue for hardware vendors, data center operations, and application designers. One of the major goals in this area is the energy proportionality, which argues that power consumption of the system should be proportional to its current level of performance. In the area of DBMS, a lot of recent work for energy efficiency is focused on coarse-grained power management on the level of multiple machines and whole queries.

In this paper, we argue that DBMS can achieve better energy efficiency through fine-grained scheduling at run-time using precise hardware models. To avoid the overhead of on-the-fly data mining, DBMS should calibrate operators beforehand using different parameters for scheduling policies, parallelism, and memory access strategies. These models can be used at run-time for dynamic scheduling of multiple queries and power management. Our experimental results suggest that this direction is highly promising: we show that calibration curves of basic analytical operations can improve energy efficiency by up to 4x.

Future work. We intend to use our calibration curves for a power-aware query optimizer, dynamically adjusting power management features, and scheduling multiple queries at run-time. Our approach includes ideas such as: (a) *sleep on block*, whenever there is a single-threaded pipeline blocker operator (e.g., sort), we can get other cores to sleep and let the worker core go into turbo mode, (b) *frequency adjustment*, if we have a group of memory-intensive threads on the same socket, we can lower the frequency of the socket, and (c) *socket power cycling*, if a whole socket, including the content of its caches, is not needed for a period of time, we can save power by putting it in deep sleep, which flushes the caches. Ultimately, we intend to combine our calibration curves with dynamic thread scheduling, data placement, and power adjustment techniques to achieve energy proportionality. We will also extend our analysis to include the power consumption of other system components (e.g., mainboard, fans, etc.) to reveal the energy efficiency of the entire system in addition to sockets and DRAM. The fine-grained approach to energy efficiency will be even more important for future processors and memories that will face more constrained power budgets.

Acknowledgements

Supported in part by the German Research Foundation (DFG) in the Collaborative Research Center 912 “Highly Adaptive Energy-Efficient Computing” and by the Swiss National Science Foundation (Grant No. 200021-146407/1).

7. REFERENCES

[1] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *MICRO*, pages 319–330, 2004.

[2] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *Eurosys*, pages 43–56, 2012.

[3] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and QoS-aware Cluster Management. In *ASPLOS*, pages 127–144, 2014.

[4] R. Dementiev, T. Willhalm, O. Bruggeman, P. Fay, P. Ungerer, A. Ott, P. Lu, J. Harris, P. Kerly, and P. Konsor. Intel performance counter monitor 2.0, 2012. http://www.intel.com/software/pcm.

[5] D. Hackenberg, T. Ilsche, R. Schone, D. Molka, M. Schmidt, and W. E. Nagel. Power measurement techniques on standard compute nodes: A quantitative comparison. In *ISPASS*, pages 194–204, 2013.

[6] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz. Understanding sources of inefficiency in general-purpose chips. In *ISCA*, 2010.

[7] J. R. Hamilton. Internet-Scale Datacenter Economics: Where the Costs And Opportunities Lie. In *HPTS*, 2011.

[8] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, 2011.

[9] S. Harizopoulos, J. Meza, M. A. Shah, and P. Ranganathan. Energy efficiency: The new holy grail of data management systems research. In *CIDR*, 2009.

[10] U. Hölzle. Brawny cores still beat wimpy cores, most of the time. *IEEE Micro*, 30(4), 2010.

[11] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan. Meet the Walkers: Accelerating Index Traversals for In-memory Databases. In *MICRO*, pages 468–479, 2013.

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

[13] W. Lang, R. Kandhan, and J. M. Patel. Rethinking query processing for energy efficiency: Slowing down to win the race. *IEEE DEBull*, 34(1):12–23, 2011.

[14] W. Lang, J. M. Patel, and S. Shankar. Wimpy node clusters: what about non-wimpy workloads? In *DaMoN*, pages 47–55, 2010.

[15] J. H. Laros III, K. Pedretti, S. M. Kelly, W. Shu, K. Ferreira, J. Vandyke, and C. Vaughan. Energy delay product. In *Energy-Efficient High Performance Computing*, pages 51–55. 2013.

[16] D. Li, B. R. de Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron. Strategies for energy-efficient resource management of hybrid programming models. *IEEE TPDS*, 24(1):144–157, 2013.

[17] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE CS TCCA Newsletter*, pages 19–25, Dec. 1995.

[18] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, May 2008. http://www.openmp.org/mp-documents/spec30.pdf.

[19] Oracle Labs. Project RAPID. Available at https://labs.oracle.com.

[20] M. Poess and R. O. Nambiar. Energy cost, the key challenge of today's data centers: A power consumption analysis of TPC-C results. *PVLDB*, 1(2):1229–1240, 2008.

[21] A. Raghavan, L. Emurian, L. Shao, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. Martin. Computational sprinting on a hardware/software testbed. In *ASPLOS*, pages 155–166, 2013.

[22] D. Schall and T. Härder. Energy-proportional query execution using a cluster of wimpy nodes. In *DaMoN*, pages 47–55, 2013.

[23] R. Schöne, D. Hackenberg, and D. Molka. Memory performance at reduced CPU clock speeds: an analysis of current x86_64 processors. In *HotPower*, 2012.

[24] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *SIGMOD*, 2010.

[25] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross. Q100: The Architecture and Design of a Database Processing Unit. In *ASPLOS*, pages 255–268, 2014.

[26] Z. Xu, Y.-C. Tu, and X. Wang. Exploring power-performance tradeoffs in database systems. In *ICDE*, pages 485–496, 2010.

[27] Z. Xu, Y.-C. Tu, and X. Wang. Pet: reducing database energy cost via query optimization. *PVLDB*, 5(12):1954–1957, 2012.