Heterogeneity-Conscious Parallel Query Execution: Getting a better mileage while driving faster!

Tobias Mühlbauer Wolf Rödiger Robert Seilbeck
Alfons Kemper Thomas Neumann

Technische Universität München
{muehlbau, roediger, seilbeck, kemper, neumann}@in.tum.de

ABSTRACT

Physical and thermal restrictions hinder commensurate performance gains from the ever increasing transistor density. While multi-core scaling helped alleviate dimmed or dark silicon for some time, future processors will need to become more heterogeneous. To this end, single instruction set architecture (ISA) heterogeneous processors are a particularly interesting solution that combines multiple cores with the same ISA but asymmetric performance and power characteristics. These processors, however, are no free lunch for database systems. Mapping jobs to the core that fits best is notoriously hard for the operating system or a compiler. To achieve optimal performance and energy efficiency, heterogeneity needs to be exposed to the database system.

In this paper, we provide a thorough study of parallelized core database operators and TPC-H query processing on a heterogeneous single-ISA multi-core architecture. Using these insights we design a heterogeneity-conscious job-to-core mapping approach for our high-performance main memory database system HyPer and show that it is indeed possible to *get a better mileage while driving faster* compared to static and operating-system-controlled mappings. Our approach improves the energy delay product of a TPC-H power run by 31% and up to over 60% for specific TPC-H queries.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems

Keywords

dark silicon; heterogeneous; multi-core; energy efficiency

1. INTRODUCTION

Following Moore's law, transistor density on a given chip area continues to double with each process generation. Coupled with Dennard scaling, i.e., the proportional scaling of threshold and supply voltages to keep power density constant, the growing number of transistors led to commensurate performance increases in the past. In recent years, however, supply voltage scaling has slowed down [10]. The failure of Dennard scaling and a constant processor power budget, which is constrained by thermal and physical restrictions, now pose the dilemma that either transistors need to be underclocked or not all transistors can be used simultaneously, leading to *dimmed* or *dark silicon* [7, 8]. Although multi-core scaling helps to alleviate dark silicon, it is just a workaround as the fraction of transistors that can be powered continues to decrease with each process generation [2]. Future processors will thus need to become more *heterogeneous*, i.e., be composed of cores with asymmetric performance and power characteristics to use transistors effectively [2, 7, 8, 29].

Examples of commercially available heterogeneous processors include the IBM Cell processor, Intel CPUs with integrated graphics processors, AMD accelerated processing units, and the Nvidia Tegra series. With *big.LITTLE* [23], ARM proposes another, particularly interesting heterogeneous design that combines a cluster of high performance out of order cores (*big*) with a cluster of energy efficient in-order cores (*LITTLE*). Despite being asymmetric in performance and power characteristics, both types of cores implement the same instruction set architecture (ISA). Single-ISA multi-core architectures are desirable for a number of reasons: (i) LITTLE cores reduce energy consumption during phases of low load, (ii) multiple LITTLE cores provide high parallel performance while big cores ensure high serial performance, mitigating the effects of Amdahl's law, (iii) the single instruction set allows to maintain a single implementation, and (iv) heterogeneous general-purpose cores avoid over-specialization that can occur with ASICs and FPGAs. While ARM big.LITTLE processors are currently only available for the embedded and mobile market, AMD has announced 64 bit big.LITTLE server processors for 2014.

Single-ISA heterogeneous processors, however, are no free lunch for database systems. Each query processing job needs to be mapped to a core that is best suited for the job. just like non-uniform memory access (NUMA) needs to be taken into account during query processing [17, 16], we argue that processor heterogeneity needs to be exposed to the database system [3] in order to achieve an optimal job-to-core mapping. Such a mapping is both important and challenging: heuristics based on load, CPI, and miss rates do not achieve optimum performance and energy efficiency [29, 3]. Whereas the operating system and compilers rely on such heuristics, database systems have a priori knowledge about the workload, enabling them to make better mapping decisions.

DaMoN'14, June 23, 2014, Snowbird, UT, USA.

ACM 978-1-4503-2971-2/14/06 ...$15.00.
http://dx.doi.org/10.1145/2619228.2619230.

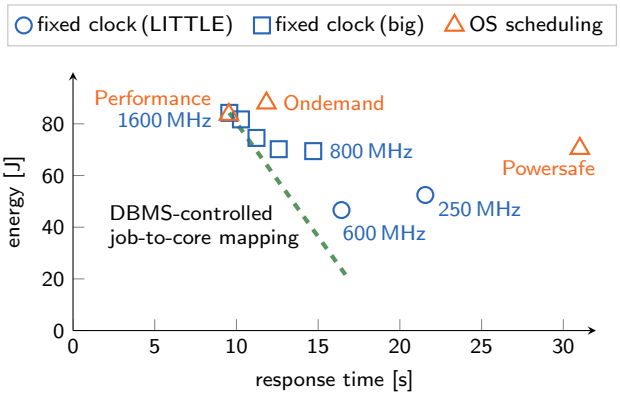


Figure 1: Response time and energy consumption of TPC-H scale factor 2 power runs with static and operating-system-controlled (OS) job-to-core mappings. The potential of DBMS-controlled job-to-mapping is to reduce energy consumption and improve performance compared to fixed clock rates and OS scheduling. The dashed line indicates a constant energy delay product (EDP) relative to the big cluster at 1600 MHz, i.e., trading an equal percentage of performance for energy savings. DBMS-controlled mapping targets the area below the constant EDP curve.

In this work we examine the potential of a heterogeneity-conscious DBMS-controlled job-to-core mapping approach for parallel query execution engines. In particular, we make the following contributions: (i) We provide a thorough study on the effects of running parallelized core database operators and TPC-H query processing on a big.LITTLE architecture. (ii) Using the insights gained from our study, we design and integrate a heterogeneity-conscious job-to-core mapping approach in our high-performance main memory database system HyPer [12] and show that it is indeed possible to *get a better mileage while driving faster* compared to static and operating-system-controlled (OS) mappings. (iii) We evaluate our approach with the TPC-H benchmark and show that we improve response time by 14% and reduce energy consumption by 19% compared to OS-controlled mapping. This corresponds to a 31% improvement of the energy delay product. For specific TPC-H queries, we show that improvements of over 60% are possible. (iv) Finally, we explore the design space for future heterogeneous multi-core processors in light of dark silicon and highlight the implications for parallel query execution engines.

While fast query response times have always been of importance in database research, improving energy efficiency by adapting database software has only recently gained importance. Related work in this area focuses on achieving energy proportionality in database clusters [26, 14, 15], analyzing energy efficiency of database operators on homogeneous multi-core servers [28, 9], adapting the query optimizer for energy efficiency [30], and using specialized hardware such as FPGAs and ASICs to improve performance and reduce energy consumption [13, 21]. In contrast to previous work, we show how to improve energy efficiency and make query processing faster in the context of single-ISA heterogeneous multi-core processors. To the best of our knowledge we are the first to explore this potential for database systems.

The main question we tackle is: *How can we make a parallel query processing engine use a single-ISA heterogeneous multi-core processor such that we reduce energy consumption while maintaining or even improving query processing performance?*

Fig. 1 illustrates our goal. It shows response time and energy consumption for TPC-H scale factor 2 power runs with static and OS-controlled job-to-core mappings on our big.LITTLE evaluation system. This initial data confirms the finding of Tsirogiannis et al. [28] who stated that for a database server *"the most energy-efficient configuration is typically the highest performing one"*. To correlate energy efficiency and performance, we use the energy delay product (EDP), which is defined as $energy \times delay$ and is measured in Joules times seconds. It is typically used to study trade-offs between energy and performance. In the context of query processing, energy is the energy consumed and delay the response time to process a query. The dashed line in Fig. 1 indicates a constant EDP relative to the highest performing configuration (big cluster at 1600 MHz). This means that along this line we trade an equal percentage of performance for energy savings. Ideally, DBMS-controlled mapping is either on or even below this line. Our benchmark reveals that with current static and OS-controlled mappings even true energy proportionality, an important aspect in today's cluster design [4], cannot be achieved.

In the following, we show that, some parallelized core database operators achieve a better EDP on the LITTLE than the big cluster, if evaluated in isolation. This opens the opportunity for our DBMS-controlled mapping approach.

2. HETEROGENEITY-AWARE PARALLEL QUERY EXECUTION

For our experiments, we use our high-performance main memory database system HyPer [12], which we ported to the ARM architecture [20]. HyPer implements a parallel query execution engine based on just-in-time compilation [22] and a *morsel*-driven parallelization engine [16]. Compared to classical Volcano-style (tuple-at-a-time) and vectorized (e.g., Vectorwise) query execution engines, our data-centric code generation relies on execution pipelines in which operators that do not require intermediate materialization are interleaved and compiled together. Such operators include join probes and aggregations, while join builds mark pipeline breakers. With the morsel-driven query execution framework, scheduling of pipeline jobs becomes a fine-grained runtime task. Morsel-driven processing essentially takes fragments of input data coming from either a pipeline breaker or a base relation, so-called morsels[1], and dispatches pipeline jobs on these morsels to worker threads. The degree of parallelism is thereby elastic and can even change during query execution. Worker threads are created at database startup and are pinned to cores. Only one thread per hardware context is created to avoid oversubscription. During query processing, no thread needs to be created to avoid thread creation and cleanup costs. Once started, a pipeline job on a morsel should not migrate to another core as our approach tries to keep data in registers and low-level caches for as long as possible. Switching cores would evict these registers and caches, leading to severe performance degradations.

[1]Our experiments show that morsels of 100,000 tuples enable an almost perfect degree of parallelism and load balancing.

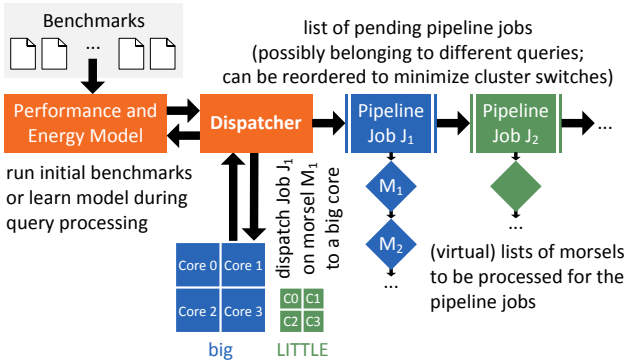


Figure 2: Heterogeneity-aware dispatching: query pipeline jobs on morsels (i.e., input data fragments) are dispatched to appropriate cores using a performance and energy model (PEM)

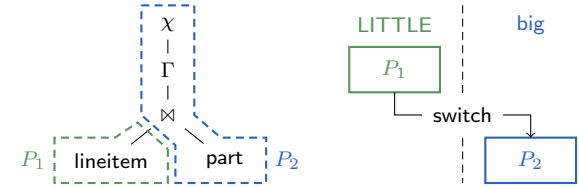
In HyPer, all database operators are parallelized such that the same pipeline job can efficiently run on multiple morsels in parallel. Mapping of pipeline jobs to worker threads (and thus cores) is performed by a dispatcher (see Fig. 2). The mapping decision of the dispatcher can be made at two points in time: during query optimization or at runtime. We argue to make this decision at runtime for two important reasons. First, mapping at optimization time has to rely on an estimate of intermediate result set cardinalities, while at runtime actual cardinalities are known. Second, runtime-based mapping can further take response time, energy, and other quality of service constraints into account.

To make the job-to-core mapping of the dispatcher conscious of heterogeneity, we extended our system with a *Performance and Energy Model (PEM)*. The dispatcher queries the PEM for each operator of a pipeline job to determine which cluster is the better choice for the job in terms of energy consumption and performance, encapsulated in the EDP. The PEM consists of multiple segmented multivariate linear regression models that estimate the energy consumption and performance of the parallelized database operators given a target cluster (LITTLE or big), a specified number of cores, and operator-specific parameters. The set of these parameters is selected carefully for each operator, such that the amount of data points that need to be collected to develop the model for a given hardware platform stays small. The data to calibrate the model is either collected through initial benchmarking or are gathered and adaptively updated during query processing. Our PEM-based approach is not limited to ARM big.LITTLE systems but is generally applicable to any kind of single-ISA heterogeneous multi-core architecture. This also includes architectures with more than two clusters and non-symmetrical numbers of cores. Besides HyPer, we are convinced that our PEM-based approach can also be integrated in many existing database systems. The general idea of our approach is independent of query compilation and can be adapted for query engines based on Volcano- and vectorized execution. Instead of entire operator pipelines, individual operators are mapped to cores.

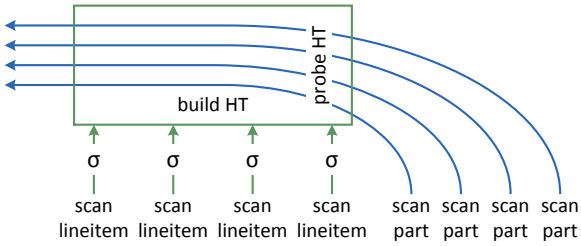
Fig. 3 demonstrates morsel-driven query processing and our heterogeneity-conscious mapping of pipeline jobs to cores using TPC-H query 14 as an example. Fig. 3(a) shows the SQL definition of query 14. HyPer parses the SQL statement and creates an algebraic tree, which is then optimized

```
select 100.00 *
       sum(case when p_type like 'PROMO%'
           then l_extendedprice * (1 - l_discount)
           else 0 end) /
       sum(l_extendedprice * (1 - l_discount))
       as promo_revenue
from lineitem, part
where l_partkey = p_partkey and
      l_shipdate >= date '1995-09-01' and
      l_shipdate < date '1995-10-01'
```

(a) SQL



(b) algebraic evaluation plan with pipeline boundaries (left) and mapping of pipeline jobs to clusters (right)



(c) four-way parallel processing of $\sigma(lineitem) \bowtie part$

Figure 3: Example: processing TPC-H Q14 using morsel-driven query execution and heterogeneity-conscious pipeline job-to-core mapping

by a cost-based optimizer. The optimizer estimates that the cardinality of `lineitem` after filtering on `l_shipdate` is smaller than `part`. Thus, as shown in Fig. 3(b), the hash table for the join of the two relations is built on the side of `lineitem`. Query 14 is divided into two pipelines. Pipeline P_1 scans `lineitem` and selects tuples that apply to the restriction on `l_shipdate`. For these tuples the hash table for the equi-join ($\bowtie$) with `part` is built. Building the hash table is a pipeline breaker. The second pipeline P_2 scans part and probes the hash table that resulted from P_1. Finally, the aggregation (Γ) and mapping (χ) evaluate the case expression and calculate the arithmetic expression for the result on the fly. HyPer generates LLVM code for the two pipeline jobs and compiles it to efficient native machine code. For each of the two pipeline jobs, the dispatcher determines the cluster that fits best. It then dispatches jobs on input morsels of the pipeline to worker threads of the determined cluster. Fig. 3(c) shows the four-way parallel processing of the equi-join between the filtered tuples of `lineitem` and `part`.

As shown in Fig. 3(b), pipeline P_1 is mapped to the LITTLE cluster and pipeline P_2 is mapped to the big cluster. Our analysis in Sect. 2.3 shows that building a large hash table is faster and more energy efficient on the LITTLE cluster due to cache and TLB misses as well as atomic compare and swap instructions. P_2 on the other hand contains a string operation and probes a hash table, which the big cluster is better suited for. The dispatcher thus switches to the big cluster when executing P_2 jobs.

	LITTLE (A7)	big (A15)
Cores	4	4
Clock rate	250–600 MHz	800–1600 MHz
Peak issue rate	2 ops/clock	3 ops/clock
Pipeline length	8–10 stages	15–24 stages
Pipeline scheduling	in-order	out of order
Branch predictor	two-level	two-level
Cache line	32 byte (VIPT)	64 byte (PIPT)
L1 I-/D-Cache	32 kB/32 kB	32 kB/32 kB
	2-way/4-way	2-way/2-way
L2 D-Cache	512 kB (shared)	2 MB (shared)
	8-way	16-way
TLB	two-level	two-level
	10 I/10 D	32 I/32 D
	256 (2-way)	512 (4-way)
Die area	3.8 mm^2	19 mm^2

Table 1: Specifications of the LITTLE cluster with A7 cores and the big cluster with A15 cores

2.1 System Under Test

Our system under test has a Samsung Exynos 5 Octa 5410 processor based on the ARM big.LITTLE architecture. It features a LITTLE cluster with four Cortex A7 cores and a big cluster with four Cortex A15 cores. Both clusters allow dynamic voltage and frequency scaling (DVFS) with clock rates up to 600 MHz (LITTLE) and 1.6 GHz (big). A cache coherent interconnect (CCI) ensures cache coherency and connects the clusters with 2 GB of dual-channel LPDDR3 memory (12.8 GB/s transfer rate). Both, LITTLE and big cores, implement the ARMv7-A instruction set architecture (ISA). Despite that, the cores' features differ: LITTLE cores are in-order cores with shallow pipelines and a small last-level cache. big cores are out-of-order cores with a deep pipeline and a comparatively large last-level cache. These differences lead to a staggering difference in size: a big core occupies 5× as much space on the die than a LITTLE core. Table 1 contains the full set of specifications.

Both clusters further exhibit asymmetric performance and power characteristics for different workloads. While the big cluster shows its strengths at compute-intensive workloads with predictable branching and predictable memory accesses, the LITTLE cluster has a much better EDP in memory-intensive workloads and workloads where branches are hard to predict, many atomic operations are used, or data accesses show no temporal or spatial locality. For these workloads, the out of order pipelines of big cores are frequently stalled, which has a negative impact on energy efficiency [10]. Further, the larger caches of big cores are more energy hungry than the smaller cores of LITTLE cores. Our analysis in Sect. 2.3 shows that for certain tasks the LITTLE cluster not only uses less energy but also offers better performance. In light of dark silicon many LITTLE in-order cores seem to be more appealing than a single big out of order core for OLAP-style query processing. We show that four LITTLE cores, which occupy approximately the same die area as one big core, outperform the big core in almost all benchmarks.

The operating system of our system is based on Linux kernel version 3.11, which assigns jobs to cores using the *cluster migration* approach. In this approach only one of the two clusters is active at a time, which makes it a natural extension to DVFS (through a unified set of P-states).

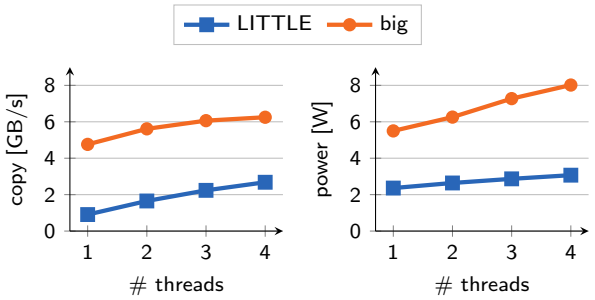


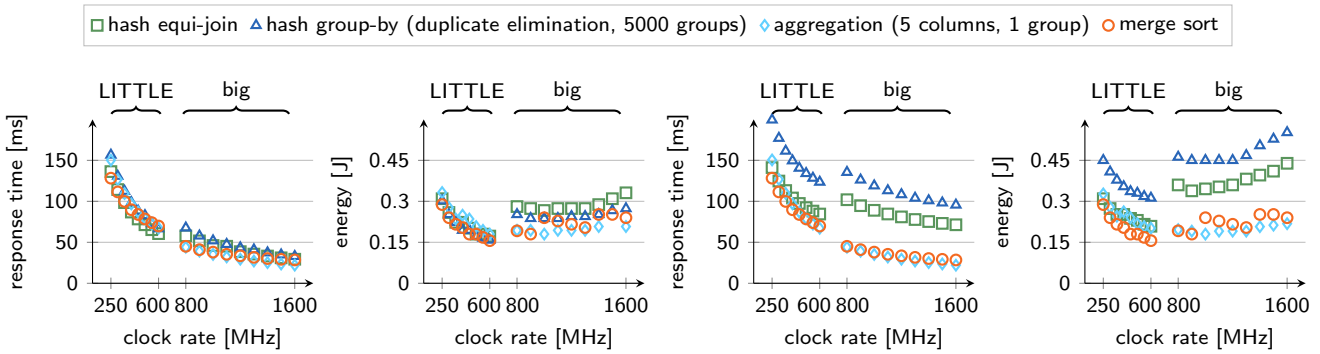
Figure 4: Throughput and power consumption of the stream copy benchmark with a varying number of threads on the LITTLE and big cluster

The operating system transitions between the two clusters based on a governor mode that adjusts the P-state. The default governor mode is *ondemand*, which sets the cluster and its clock rate depending on current system load. Cluster switching completes in under 2,000 instructions on our hardware. We use the cpufreq library to switch clusters and clock rates from inside the database system. Upcoming big.LITTLE processors and newer kernel versions will implement two more operating system scheduling modes: *in-kernel switching* (IKS) and *global task scheduling* (GTS). IKS pairs a LITTLE with a big core and switches on a per-core basis. Beyond that, GTS enables true heterogeneous multi-processing, where all cores can be used at the same time. Unfortunately our hardware does not allow the simultaneous usage of all eight cores. However, we expect the main results presented in this work to be equally true for upcoming IKS and GTS modes on newer hardware. Even if operating-system-based job-to-core mapping strategies become more sophisticated, these strategies are likely based on performance counters and are unaware of memory-level parallelism and how misses and other indicators translate into overall performance. We thus argue strongly for a DBMS-controlled mapping approach.

Our system under test is connected to a power supply with a power meter such that we can measure the actual energy drawn by the whole system from the wall socket. The power meter has a sampling rate of 10 Hz and a tolerance of 2%. It exposes its collected data to the system via a USB interface. Being idle, the system draws 2 W. Under load, a single LITTLE core draws 240 mW, a big core 2 W.

2.2 Initial Benchmarks

In order to get an estimate for the peak sustainable memory bandwidth of our system under test, we first run the STREAM benchmark [18]. Fig. 4 shows the throughput and power consumption of the STREAM copy benchmark with a varying number of threads on the LITTLE and big cluster. For reasons of brevity we do not show the scale, add, and triad results as these only differ by a constant factor. The copy benchmark essentially copies an array in memory. We performed the benchmark with an array size of 512 MB. Reported numbers are an average over multiple runs. With four threads, the LITTLE cluster achieved a peak bandwidth of 3 GB/s, the big cluster of just over 6 GB/s. The higher copy performance comes at a price. With the big cluster, the system draws close to 8 W while with the LITTLE cluster it only draws around 3 W.



(a) working set in LLC

(b) size of working set exceeding LLC

Figure 6: Response time and energy consumption of multi-threaded hash equi-join, hash group-by, aggregation, and merge sort operators on the LITTLE and big cluster with varying clock rates and working set sizes that (a) fit in the last level cache (LLC) of the cluster and (b) exceed the LLC of the cluster

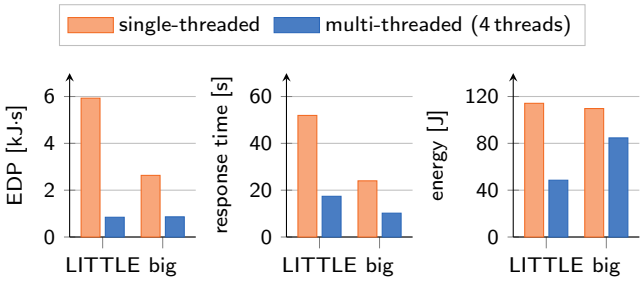


Figure 5: Single- and multi-threaded execution of the 22 TPC-H queries (scale factor 2) on the clusters

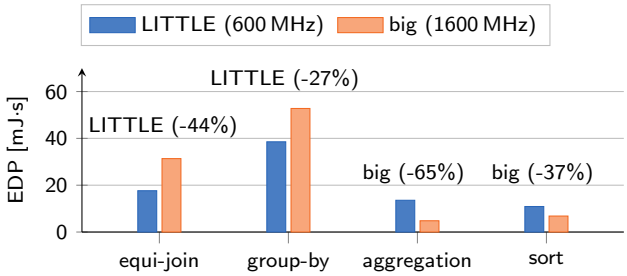


Figure 7: EDP for multi-threaded equi-join, group-by, aggregation, and sort operators on the LITTLE and big cluster at the highest clock rate and with working set sizes exceeding the LLC (cf., Fig. 6)

In a second benchmark we compare single- and multi-threaded execution of the 22 TPC-H queries on the LITTLE and big cluster at their highest respective clock rate. Fig. 5 shows the results. With single-threaded execution, the big core clearly outperforms the LITTLE core. It finishes processing the queries so much quicker that it's energy consumption is even below that of the LITTLE core. With multi-threaded execution, the big cluster is still faster but the difference in performance is much smaller. Conversely, the LITTLE cluster now consumes less energy. As a consequence, the EDP of the LITTLE and big cluster is almost equal for multi-threaded query processing.

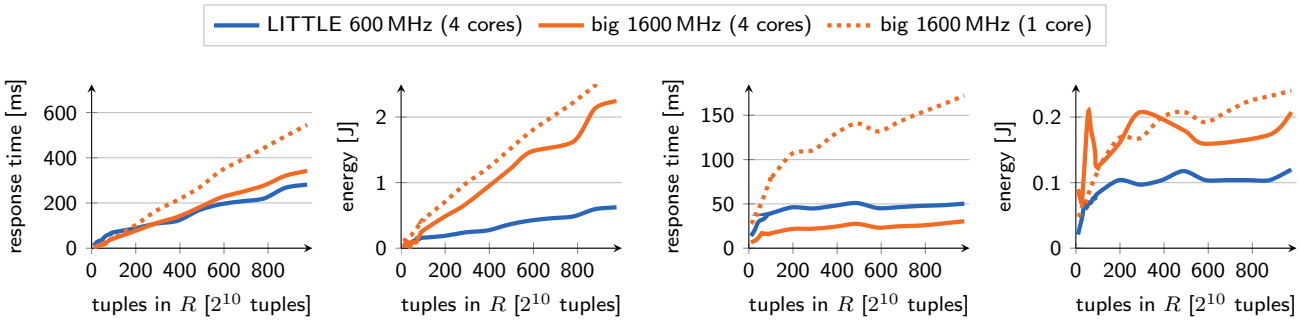
2.3 Analysis of Database Operators

To develop a Performance and Energy Model (PEM) for our big.LITTLE platform, we first analyze how parallelized database operators behave on the two clusters. We choose equi-join, group-by/aggregation, and sort as benchmark operators. The reason behind this choice is that by far most cycles of a TPC-H run are spent in these operators. We expect this to be equally true for most analytical workloads.

In an initial experiment (see Fig. 6 and Fig. 7), we benchmark the parallelized operators on the two clusters with four cores each and vary the clock rate of the cores. The LITTLE cluster is benchmarked with clock rates from 250 to 600 MHz and the big cluster with clock rates from 800 to 1600 MHz. Two cases are considered: (i) the working set of the operator fits in the last-level cache (LLC) and (ii) the working set exceeds the LLC of the clusters.

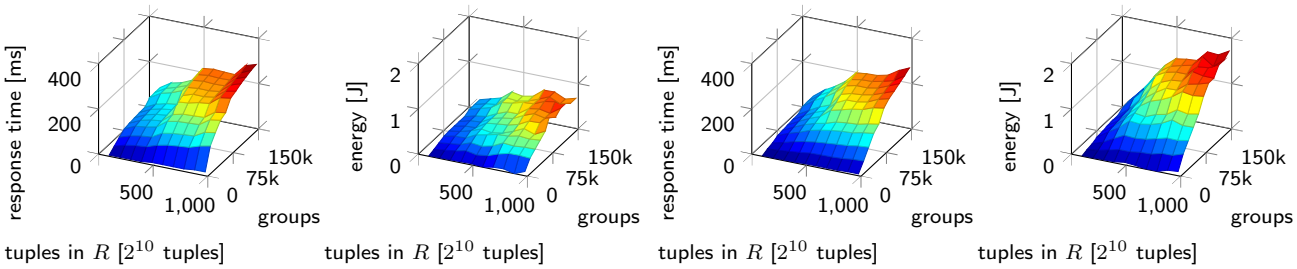
Equi-join. Joins rank among the most expensive core database operators and appear in almost all TPC-H queries. Main memory database systems implement the join operator usually as either a hash-, radix-, or a sort-merge-join. In special cases a nested loop or index-based join method is used. The choice of implementation in general depends on the system implementation as well as the physical database design [5]. For equi-joins, HyPer uses a hash join implementation that allows the hash table to be built in parallel in two phases. In the first phase, build input tuples are materialized in thread-local storage. Then, a perfectly sized global hash table is created. In the second phase, each worker thread scans its storage and sets pointers in the global hash table using atomic compare-and-swap instructions[2]. For our initial benchmark we run a hash equi-join on two relations with random numeric values. If the working set exceeds the LLC, the LITTLE cluster shows a much better energy delay product (EDP) than the big cluster. In Sect. 2.3.1 we further explore the join operator and benchmark the build and probe phases individually. The build phase usually shows a better EDP on the LITTLE cluster because the atomic compare-and-swap instructions stall the pipeline to guarantee serializability. This hinders out of order execution and diminishes the performance advantages of the big cores.

[2]For details, we refer to [16]. Note that our evaluation system is a 32 bit system. Thus the tagging approach described in [16] is not used, which leads to more pointers being chased.



(a) build (b) probe

Figure 8: Response time and energy consumption of multi-threaded build and probe phases of the hash equi-join $R \bowtie S$ on the LITTLE and big cluster (build cardinality $|R| \leq 1000 \cdot 2^{10}$ tuples, probe cardinality $|S| = 1000 \cdot 2^{10}$ tuples, 8 byte keys)



(a) LITTLE (b) big

Figure 9: Response time and energy consumption of a multi-threaded hash grouping with a varying number of input tuples in R and a varying number of result groups (distinct keys) on the LITTLE and big cluster

Group-by/aggregation. Groupings/aggregations occur in all TPC-H queries. In HyPer parallel aggregation is implemented using a two-phase aggregation approach similar to IBM DB2 BLU's aggregation [25]. First, worker threads pre-aggregate heavy hitters using a thread-local, fixed-size hash table. When the table is full, it is flushed to overflow partitions. In the second phase these partitions are then repeatedly exchanged, scanned, and aggregated until all partitions are finished. For our benchmark we separate two cases: pure grouping for duplicate elimination and pure aggregation with only a single group. As our results show, when the working set exceeds the LLC, both cases show different performance and power characteristics. While pure aggregation profits from the big cores' higher compute power, pure grouping has a better EDP on the LITTLE cluster. In our grouping benchmark, 66% of keys were duplicates. In Sect. 2.3.2 we show that the performance and energy efficiency of group-by/aggregation operators depends much on the number of groups (distinct keys). For few groups, partitions (i.e., groups) fit into caches, which benefits the big cores. Many groups on the other hand lead to many cache and TLB misses. In this case, pipelines on big cores are frequently stalled and LITTLE cores achieve a better EDP.

Sort. HyPer uses sorting to implement *order by* and *top-k* clauses, which are both frequently used in TPC-H queries. Internally, sorting is implemented as a two-phase merge sort. Worker threads first perform a local in-place sort followed by a synchronization-free parallel merge phase. For the benchmark we sort tuples according to one integer attribute. The total payload of a tuple is 1 kB. Our results indicate that sorting always achieves a better EDP on the big cores, no matter if the working set fits or exceeds the LLC.

The initial benchmark leads us to the conclusion that working set size is an important indicator for where operators should be placed. While big cores always show a better EDP when the working set fits into the LLC, LITTLE cores show a better EDP for the equi-join and group-by operators when the working set exceeds the LLC. The benchmark also shows that running the cores at the highest clock rate (600 MHz and 1600 MHz, respectively) almost always yields the best EDP for the cluster. In the following we thus run cores at their highest clock rate. The PEM can nevertheless be extended to take frequency scaling into account, which can save substantial amounts of energy [15].

2.3.1 *Equi-join*

To better understand the equi-join operator, we split it into its build and probe phase and repeat our benchmark for varying input sizes. For the PEM, it is necessary to get separate estimates for the build and probe phases as both phases are parts of different pipelines. Fig. 8 shows our results for both join phases of the parallelized hash equi-join operator. Building the hash table is the dominant time factor and shows a much better EDP on the LITTLE cluster for working set sizes exceeding the LLC. Probing on the other hand has a better EDP on the big cluster. In light of dark silicon, however, the LITTLE cluster is better compared to a single big core, which approximately occupies the same die area. In this case, the LITTLE cluster is again the winner.

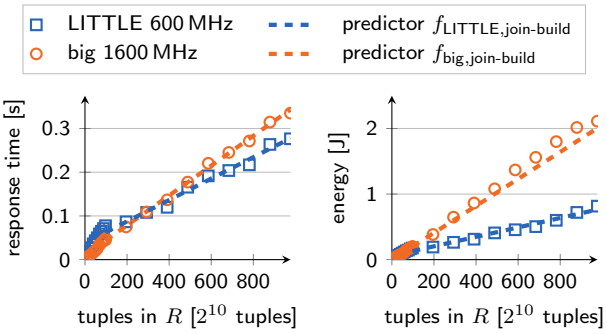


Figure 10: Segmented linear regression model for the build phase of the hash equi-join operator

operator	variables for regression
equi-join (build)	build cardinality
equi-join (probe)	size of hash table, probe cardinality
group-by	input cardinality, groups (estimate)
aggregation	input cardinality, groups (estimate), number of aggregates
sort	input cardinality, number of attributes, attribute types
all operators	string operations (yes/no)

Table 2: Parameters for operator regression models

2.3.2 *Group-by/aggregation*

Group-by/aggregation is an example for an operator for which the EDP is not only dependent on the input size, but also on the number of groups it generates. We repeat our benchmarks for the group-by operator (without aggregation) and vary the input size and the number of groups. Fig. 9 shows that the number of groups has a great influence on operator runtime and energy consumption.

2.3.3 *Further operators*

Due to lack of space we refrain from showing benchmark results for expression evaluations and string operations. In general, the big cluster has a better EDP for complex expressions and string operations. For example, TPC-H Q2, Q9, and Q13 have costly like predicates. Pipelines that evaluate these predicates should be executed on the big cluster.

2.4 Performance and Energy Model

The data points collected during our benchmarks build the foundation for the Performance and Energy Model (PEM) for our big.LITTLE platform. For other hardware platforms these benchmarks need to be run initially or can be collected during query processing. The goal of the PEM is to provide a model that estimates the response time and energy consumption of database operators given a target cluster (LITTLE or big), the number of threads, and operator-specific parameters. Table 2 lists the parameters that we consider for the operators when developing the PEM. For each database operator we create multiple multivariate segmented linear regression models using the least squares method: one response time and one performance model for each combination of target cluster and number of threads. In general, for clusters $c_1, \ldots, c_n$ with $|c_i|, 1 \leq i \leq n$ cores each and $|o|$ operators, $\sum_{i \in \{1,\ldots,n\}} |c_i| \cdot |o| \cdot 2$ segmented linear regression models are created. We use the R C++ library to automatically compute the models given our benchmark results. Computing the models takes less than a few seconds. Similarly the models can be quickly refined if new data points were collected during query processing.

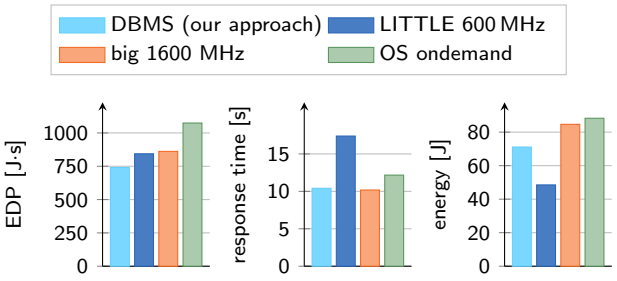


Figure 11: TPC-H (scale factor 2) evaluation

All database operators that we study show a linear correlation given the independent parameters we have chosen for the respective operator. We regard the two cases where the working set of the operator fits into the LLC and the case where the working set exceeds the LLC[3]. The models are thus divided into two segments. The first segment is a model for working sets that fit into the LLC and the second segment is a model for working sets that exceed the LLC.

Fig. 10 shows an example of our segmented linear regression model for the build phase of the hash equi-join operator. For each of the two segments, an equal number of data points is collected. The predictor functions for the LITTLE and big clusters estimate the data points with only a negligible residual error.

2.5 Heterogeneity-Conscious Dispatching

The dispatcher uses the Energy and Performance Model (PEM) to estimate the response time and energy consumption of an execution pipeline if ran on a specific cluster of the heterogeneous processor. For a pipeline p with operators $o_1, \ldots, o_n$, response time and energy consumption, and thus also the energy delay product (EDP), for the LITTLE and big cluster are estimated by querying the PEM for each of the operators and each of the clusters. In general, the response time r_p for pipeline p on a cluster c is estimated as $\sum_{i \in \{1,\ldots,n\}} r_{c,o_i}(ctx)$, where r_{c,o_i} is the response time predictor function for operator o_i on cluster c and ctx is a context that contains the operator-specific parameters. Energy consumption is estimated analogously. For a pipeline, the dispatcher estimates response time and energy consumption for all clusters and dispatches the pipeline jobs to the cluster that exhibits the best weighted EDP. The weights are user-definable and allow to either put a stronger emphasis on energy efficiency or performance. For our evaluation we used a 60/40 ratio where we set the weight for performance to 0.6 and the weight for energy efficiency to 0.40.

2.6 Evaluation

We implemented our heterogeneity-conscious dispatching approach in our HyPer system and evaluate its performance and energy efficiency using the TPC-H benchmark by comparing our approach (DBMS) against the operating system's

[3]The only database operator not showing a linear correlation is the cross product. Cross products, however, occur only rarely in real use cases and are not considered in our PEM.

	cycles in M	time [s]	energy [J]	EDP [J·s]
LITTLE 600 MHz	80,623	0.18	0.5	0.09
big 1600 MHz	104,392	0.17	1.31	0.22
OS ondemand	102,659	0.22	1.4	0.31
DBMS (our app.)	68,435	0.15	0.77	0.12

Table 3: TPC-H Q14 (scale factor 2) evaluation

ondemand cpufreq governor (OS ondemand) and running TPC-H on the LITTLE and big cluster at a fixed clock rate of 600 MHz and 1600 MHz, respectively. We do not show results for the other OS governors "performance" and "powersafe" as these correspond to the big cluster at highest clock rate and LITTLE cluster at lowest clock rate configuration, respectively. Scale factor 2 is the largest TPC-H data set that fits into the main memory of our system under test. Fig. 11 shows the results of our benchmark. Reported response time and energy numbers are the sum of all 22 TPC-H queries and are an average of multiple runs. Fig. 12 shows the detailed results for all 22 queries. When comparing against the default operating system setting, our DBMS approach decreases response time by 14% and saves 19% of energy, thus getting a better mileage while being faster. This corresponds to a 31% improvement of the EDP. Our results show that when running the clusters at their highest clock rate, the LITTLE and big cluster can barely be separated. Compared to fixed clock rates the EDP is improved by 12% compared to the LITTLE cluster and 14% compared to big cluster. These improvements are much better than what could be achieved with frequency scaling and also lie below the constant EDP curve relative to the highest performing configuration (cf. Fig 1).

Not all queries profit equally from our DBMS approach. Fig. 12 shows the detailed evaluation results for the TPC-H scale factor 2 benchmark. The queries that profit most are Q5, Q14, and Q18. The EDP for all these queries is improved by more than 40%. For Q5, Q14, and Q19, response times are even faster than what was originally benchmarked with the big cluster at the highest clock rate. These queries have pipelines that better fit the LITTLE than the big cluster. The OS likely draws the wrong conclusions when such a query is executed. As it only sees a load spike it thus gradually increases clock rate by increasing the P-state. Ultimately it will switch to the big cluster and reach the highest clock rate. Our DBMS-controlled approach on the other hand enforces the LITTLE cluster for the aforementioned pipelines. For Q1 where it seems that DBMS-controlled mapping can do little to improve the EDP as the query contains no joins and only a single pipeline, DBMS-controlled mapping still improves the EDP significantly compared to OS-controlled mapping. This is because OS ondemand has to react to the sudden load spike and gradually increases the clock rate of the cores. Our approach on the other hand knows that Q1 is best executed on the big cluster at the highest clock rate and immediately switches to that configuration. After the query is finished, the clock rate can be decreased again. The same applies to queries Q2, Q3, Q6, Q8, Q11, Q13, Q15, Q16, Q17, Q20, Q21, Q22. All these queries are dispatched exclusively to the big cluster. To a certain extent this is due to our EDP weights. Changing the weights in favor of energy efficiency results in more pipelines being mapped to the LITTLE cluster.

Table 3 shows detailed performance counters for Q14 (cf. Fig. 3). Query 14 is one of the queries that profits most from our DMBS approach. Compared to the operating system's ondemand governor, our approach reduces response time by 35% and decreases energy consumption by 45%.

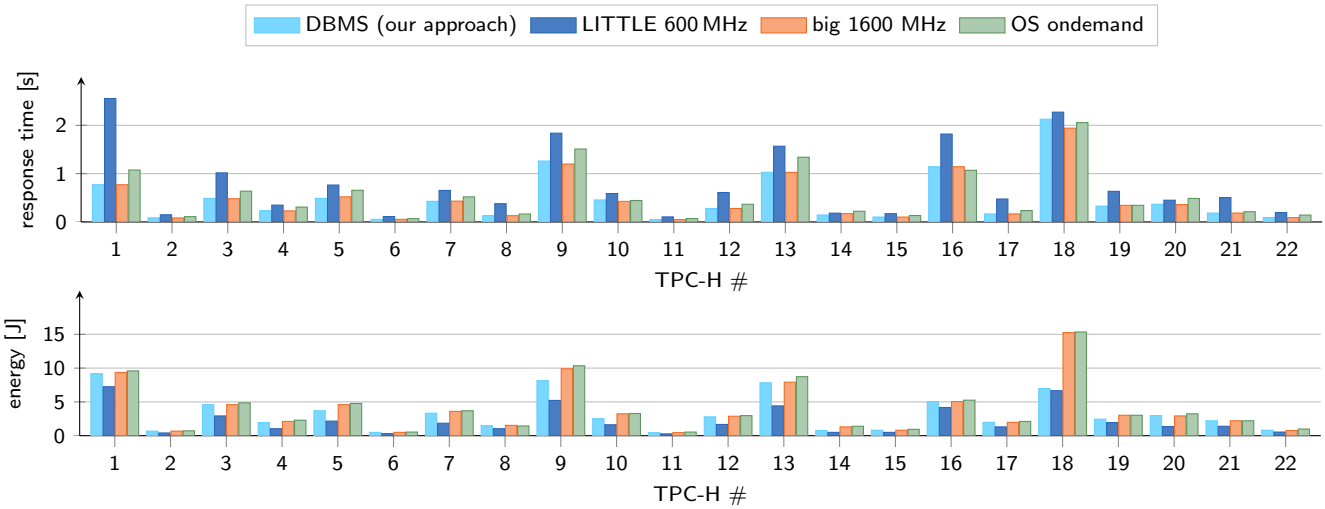
3. RELATED WORK

In light of dimmed or dark silicon, several authors call for heterogeneous system architectures [2, 7, 8]. In this respect, the usage of general purpose GPUs for query (co-)processing already receives a lot of attention. Pirk et al. [24] recently described a generic strategy for efficient CPU/GPU cooperation for query processing by using the GPU to calculate approximate result sets which are then refined on the CPU. Karnagel et al. [11] showed how to accelerate stream joins by outsourcing parts of the algorithm to the GPU. Other authors show how FPGAs [21] can be used to improve performance and reduce energy consumption in database systems. Further, on-chip accelerators have been investigated for database hash table lookups [13].

We focus on single-ISA heterogeneous multi-core architectures and how these can be used to improve performance and energy efficiency of a database system. Such architectures include ARM's big.LITTLE [23] and Intel's QuickIA [6], which combines a Xeon server CPU with an energy efficient Atom CPU in a single system. Previous work on single-ISA heterogeneous architectures has shown that database systems need to be adapted in order to optimally use heterogeneous processors [3] and that job-to-core mapping in such a setting is important and challenging [29].

Numerous efforts analyzed the energy efficiency of individual database operators and TPC-H queries on single homogeneous servers [28, 9, 30]. Tsirogiannis et al. [28] stated that for a database server *"the most energy-efficient configuration is typically the highest performing one"*. On heterogeneous platforms, this statement still holds for database systems that are not adapted to the underlying hardware. Our DBMS-controlled mapping approach and its evaluation, however, have shown that this statement no longer holds for query engines that dynamically map execution jobs to the core that fits best. In contrast to these proposals we study how heterogeneous multi-core architectures can be used to not only improve energy efficiency but also improve performance of query processing.

Besides single node servers, significant energy efficiency improvements have been proposed for database clusters [26, 15, 14, 27]. Schall et al. [26] suggest using a cluster of wimpy nodes that dynamically powers nodes depending on query load. Lang et al. [15] propose turning hardware components off or slowing them down to save energy. Szalay et al. [27] consider using many blades composed of low-power CPUs together with solid state disks to increase I/O throughput while keeping power consumption constant. Lang et al. [14] further study the heterogeneous shared nothing cluster design space for energy efficient database clusters. By combining wimpy and brawny nodes in a heterogeneous cluster setup, better than proportional energy efficiency and performance benefits were achieved. At the macro level, the authors use a similar approach to determine which workload should be executed on which nodes. In contrast to the this study, this work studies the effects and possibilities of heterogeneity inside a single node with heterogeneous processors and shared memory.



TPC-H #	LITTLE 600 MHz time [s]	LITTLE 600 MHz energy [J]	big 1600 MHz time [s]	big 1600 MHz energy [J]	OS ondemand time [s]	OS ondemand energy [J]	DBMS (our approach) time [s]	DBMS (our approach) energy [J]	compared to OS ondemand time [%]	compared to OS ondemand energy [%]	compared to OS ondemand EDP [%]
Q1	2.55	7.25	0.77	9.36	1.08	9.59	0.77	9.18	-28.23	-4.23	-31.75
Q2	0.13	0.41	0.08	0.68	0.11	0.72	0.09	0.68	-22.01	-6.25	-22.73
Q3	1.02	2.93	0.48	4.59	0.64	4.86	0.49	4.59	-23.56	-5.56	-27.69
Q4	0.35	1.04	0.23	2.12	0.31	2.3	0.24	1.94	-23.65	-15.69	-34.70
Q5	0.76	2.16	0.52	4.59	0.66	4.77	0.49	3.69	-25.81	-22.64	-42.57
Q6	0.11	0.32	0.05	0.5	0.07	0.54	0.06	0.5	-20.08	-8.33	-20.63
Q7	0.65	1.85	0.43	3.6	0.52	3.69	0.43	3.33	-17.65	-9.76	-25.38
Q8	0.38	1.04	0.13	1.53	0.16	1.44	0.13	1.49	-19.81	3.13	-15.93
Q9	1.84	5.22	1.2	9.9	1.51	10.35	1.26	8.15	-16.44	-21.30	-34.29
Q10	0.59	1.62	0.43	3.24	0.44	3.29	0.46	2.52	2.82	-23.29	-19.92
Q11	0.11	0.27	0.05	0.45	0.07	0.54	0.05	0.45	-30.10	-16.67	-40.48
Q12	0.61	1.67	0.28	2.88	0.37	2.97	0.28	2.79	-23.91	-6.06	-28.91
Q13	1.57	4.41	1.03	7.92	1.34	8.73	1.03	7.83	-23.24	-10.31	-31.06
Q14	0.18	0.5	0.17	1.31	0.22	1.4	0.15	0.77	-34.99	-45.16	-62.50
Q15	0.17	0.5	0.1	0.81	0.13	0.95	0.1	0.81	-22.72	-14.29	-34.41
Q16	1.82	4.18	1.14	5.04	1.07	5.26	1.14	5.04	6.87	-4.11	2.09
Q17	0.48	1.31	0.17	1.98	0.24	2.12	0.17	1.98	-29.47	-6.38	-33.84
Q18	2.27	6.66	1.94	15.26	2.06	15.35	2.13	6.98	3.38	-54.55	-52.98
Q19	0.63	1.94	0.34	3.24	0.34	3.02	0.33	2.45	-4.70	-19.05	-21.26
Q20	0.45	1.35	0.36	2.93	0.49	3.24	0.37	2.97	-24.66	-8.33	-30.78
Q21	0.51	1.4	0.18	2.2	0.21	2.21	0.19	2.2	-12.93	0.00	-9.93
Q22	0.2	0.54	0.09	0.77	0.14	0.99	0.09	0.81	-34.86	-18.18	-47.40
Sum	17.39	48.51	10.18	84.65	12.17	88.29	10.41	71.12	-14.46	-19.45	-30.89
Geo. mean	0.52	1.45	0.28	2.52	0.36	2.67	0.29	2.25	-19.44	-15.73	-31.8

Figure 12: Detailed TPC-H scale factor 2 evaluation results

4. HETEROGENEOUS PROCESSORS FOR FUTURE DATABASE SYSTEMS

Database machines and appliances are celebrating a revival. Oracle is now shipping its Exadata database appliance and IBM is offering the DB2 Analytics Accelerator (formerly Netezza), which performs data filtering on FPGAs. More interestingly, the Oracle labs project RAPID [1] is investigating how many low-power cores and fixed function accelerators can be integrated in a heterogeneous database appliance that is optimized for both, high performance and energy efficiency. Our study of heterogeneous single-ISA multi-core processors has shown that a cluster of such low-power in-order cores can offer tremendous performance per die area. Our results however also suggest, that such processors are no free lunch for current database systems. Database systems need to be adapted. Future database machine development will thus require a tightly coupled hardware/database co-design process.

5. CONCLUDING REMARKS

Besides GPUs, ASICs and FPGAs, single instruction set architecture (ISA) heterogeneous multi-core processors are another way of utilizing otherwise dimmed or dark silicon. The thorough study of parallelized core database operators and TPC-H query processing on a heterogeneous single-ISA multi-core architecture in this work has shown that these processors are no free lunch for database systems. In order to achieve optimal performance and energy efficiency, we have shown that heterogeneity needs to be exposed to

the database system, which, because of its knowledge of the workload, can make better mapping decisions than the operating system (OS) or a compiler. We have integrated a heterogeneity-conscious job-to-core mapping approach in our high-performance main memory database system HyPer that indeed enables HyPer to get a better mileage while driving faster compared to fixed and OS-controlled job-to-core mappings; improving the energy delay product of a TPC-H power run by 31% and up to over 60% for specific queries. We would like to stress that this is a significant improvement as our approach is integrated in a complete query processing system and the whole TPC-H benchmark is evaluated rather than single operators or micro-benchmarks.

In future work we plan to extend our performance and energy model to include all relational database operators. Using the work of Lang et al. [14] as a foundation, we want to explore energy efficient cluster designs for distributed transaction and query processing [19] where not only the cluster can be composed of heterogeneous nodes but nodes themselves can again be composed of heterogeneous processors. Further, upcoming hardware that allows the simultaneous usage of heterogeneous single-ISA cores will open the opportunity for co-processing of queries on heterogeneous cores.

6. ACKNOWLEDGEMENTS

Tobias Mühlbauer is a recipient of the Google Europe Fellowship in Structured Data Analysis, and this research is supported in part by this Google Fellowship. Wolf Rödiger is a recipient of the Oracle External Research Fellowship. This work has been sponsored by the German Federal Ministry of Education and Research (BMBF) grant RTBI 01IS12057.

7. REFERENCES

[1] N. Agarwal. Oracle labs : Leading the way in hardware software co-design.

[2] G. Alonso. Hardware killed the software star. In *ICDE*, 2013.

[3] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The Impact of Performance Asymmetry in Emerging Multicore Architectures. *SIGARCH*, 33(2), 2005.

[4] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12), 2007.

[5] P. A. Boncz, T. Neumann, and O. Erling. TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. In *TPCTC*, 2013.

[6] N. Chitlur, G. Srinivasa, S. Hahn, P. Gupta, D. Reddy, et al. QuickIA: Exploring Heterogeneous Architectures on Real Prototypes. In *HPCA*, 2012.

[7] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *ISCA*, 2011.

[8] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward Dark Silicon in Servers. *Micro*, 31(4), 2011.

[9] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. Energy Efficiency: The New Holy Grail of Data Management Systems Research. In *CIDR*, 2009.

[10] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2011.

[11] T. Karnagel, D. Habich, B. Schlegel, and W. Lehner. The HELLS-join: A Heterogeneous Stream Join for Extremely Large Windows. In *DaMoN*, 2013.

[12] A. Kemper and T. Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *ICDE*, 2011.

[13] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, et al. Meet the Walkers: Accelerating Index Traversals for In-memory Databases. In *MICRO*, 2013.

[14] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis. Towards Energy-Efficient Database Cluster Design. *PVLDB*, 5(11), 2012.

[15] W. Lang, R. Kandhan, and J. M. Patel. Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race. *DEBU*, 34(1), 2011.

[16] V. Leis, P. Boncz, A. Kemper, and T. Neumann. Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age. In *SIGMOD*, 2014.

[17] Y. Li, I. Pandis, R. Müller, V. Raman, and G. M. Lohman. NUMA-aware algorithms: the case of data shuffling. In *CIDR*, 2013.

[18] J. D. McCalpin. Memory Bandwidth and Machine Balance in Current High Performance Computers. *TCCA*, 1995.

[19] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann. ScyPer: Elastic OLAP throughput on transactional data. In *DanaC*, 2013.

[20] T. Mühlbauer, W. Rödiger, R. Seilbeck, A. Reiser, A. Kemper, and T. Neumann. One DBMS for all: the Brawny Few and the Wimpy Crowd. In *SIGMOD*, 2014.

[21] R. Müller and J. Teubner. FPGA: What's in It for a Database? In *SIGMOD*, 2009.

[22] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9), 2011.

[23] A. Peter Greenhalgh. big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7, 2011.

[24] H. Pirk, S. Manegold, and M. Kersten. Waste Not... Efficient Co-Processing of Relational Data. In *ICDE*, 2014.

[25] V. Raman, G. Attaluri, R. Barber, N. Chainani, D. Kalmuk, et al. DB2 with BLU Acceleration: So Much More Than Just a Column Store. *PVLDB*, 6(11), 2013.

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

[27] A. S. Szalay, G. C. Bell, H. H. Huang, A. Terzis, and A. White. Low-power Amdahl-balanced Blades for Data Intensive Computing. *SIGOPS*, 44(1), 2010.

[28] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the Energy Efficiency of a Database Server. In *SIGMOD*, 2010.

[29] K. Van Craeynest and L. Eeckhout. Understanding Fundamental Design Choices in single-ISA Heterogeneous Multicore Architectures. *TACO*, 9(4), 2013.

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