Towards Query-Aware Core-Type Selection on Heterogeneous Multi-core Processors: An Operator-Level Energy Study

XUEQING FENG, The University of Tokyo, Japan
YUTO HAYAMIZU, The University of Tokyo, Japan
KAZUO GODA, The University of Tokyo, Japan

Modern processors increasingly adopt heterogeneous core designs that combine cores with different performance and efficiency characteristics, such as Intel-style high-performance cores (P-cores) and energy-efficient cores (E-cores). While prior work has explored heterogeneity-aware query execution for performance, energy-aware core-type selection in DBMSs on modern Intel-style P-core/E-core processors has received limited attention. This paper studies energy-aware core-type selection for queries with a dominant operator on a heterogeneous multi-core processor platform. As an initial evaluation, we focus on hash aggregation, hash join, and sort, and vary workload characteristics such as cardinality ratio, key skew, and input distribution. We measure wall-plug system power using a high-precision external power meter. Our results show that the energy-efficient core type is not determined by operator type alone, but is strongly affected by workload parameters and data distribution. Based on these observations, we design a query-aware core-selection policy that selects P-cores or E-cores using query-level characteristics. Compared with Linux default scheduling, the policy reduces total above-idle system energy by **18.14%** across the evaluated workload set. These results show that database systems can actively exploit heterogeneous cores by incorporating query semantics into core-type selection.

CCS Concepts: • **Information systems** → **Data management systems**; • **Computer systems organization** → *Heterogeneous (hybrid) systems*; • **Hardware** → **Power and energy**.

Additional Key Words and Phrases: Heterogeneous Multi-core Processors, Energy-Aware Query Processing, Query Scheduling, Hybrid Architectures, Database Systems

1 Introduction

Energy consumption has become a central concern in modern computing systems, and analytical database queries are an important contributor because they continuously exercise CPUs, memory hierarchies, and storage subsystems over large volumes of data [1, 2, 7, 24]. Prior work has shown that database systems can reduce energy by considering energy-performance trade-offs in query optimization and execution-time decisions [8, 14, 25]. However, these techniques mainly focus on *what* plan or execution strategy to use.

At the same time, processor vendors increasingly adopt heterogeneous core designs to improve energy efficiency. Recent Intel processors, for example, integrate high-performance cores (P-cores) and energy-efficient cores (E-cores) on the same chip, and similar principles appear in Arm-based and other architectures [20, 21]. On such heterogeneous multi-core processors, energy-aware database execution must also decide *where* a query should run. This decision is non-trivial: lower-power E-cores may increase runtime, while faster P-cores may finish soon enough to reduce total energy. This creates a new query-level core-selection problem for database systems.

Recent studies have begun to explore heterogeneous-core execution in DBMSs[16, 18], but they primarily focus on performance and do not address query-level energy-aware core-type selection on P-core/E-core processors[4, 15]. Operating-system schedulers can distinguish core types, but they rely on general-purpose scheduling signals rather than database-level information such as operator type, cardinality, skew, and input distribution. This suggests a database-side approach, where much of this information is available from query plans and database statistics.

As a first step toward this problem, we study query-level core-type selection for simple analytical queries with a single dominant operator. This controlled workload design preserves end-to-end execution inside DuckDB while isolating common analytical execution patterns. Our study covers hash aggregation, hash join, and sort, with workload characteristics such as cardinality ratio, key skew, and input distribution varied systematically. We focus on above-idle system energy, defined as system energy above the idle baseline during query execution, because it isolates the query-induced energy directly affected by core-type selection.

Across these workloads, no single core type consistently minimizes above-idle system energy. Core preference can change within the same operator type, making fixed P-core/E-core policies, operator-only rules, and default OS scheduling insufficient. Although our paper uses queries with a dominant operator as a controlled starting point, they provide an initial step toward database-managed execution on heterogeneous multi-core processors. The results show that database-level query information can expose energy-saving opportunities beyond general-purpose OS scheduling, suggesting a promising direction for energy-aware query processing. We therefore prototype a query-aware policy that selects an energy-efficient core type using query-level characteristics, without runtime instrumentation.

The contributions of this paper are summarized as follows:

- We characterize P-core/E-core time-energy behavior for basic analytical queries with a dominant operator on a heterogeneous multi-core processor, showing that energy-efficient core-type selection is workload-sensitive and cannot be determined by fixed policies or operator type alone.
- We prototype and evaluate query-aware core selection using a runtime-instrumentation-free energy prediction policy. The policy reduces total above-idle system energy by **18.14%** compared with Linux default scheduling.
- We highlight database-managed heterogeneous execution as a promising direction for energy-aware query processing on modern heterogeneous multi-core processors.

Authors' Contact Information: Xueqing Feng, The University of Tokyo, Tokyo, Japan, xueqing@tkl.iis.u-tokyo.ac.jp; Yuto Hayamizu, The University of Tokyo, Tokyo, Japan, haya@tkl.iis.u-tokyo.ac.jp; Kazuo Goda, The University of Tokyo, Tokyo, Japan, kgoda@tkl.iis.u-tokyo.ac.jp.

2 Background and Related Work

Energy-aware query processing. Prior work has studied database energy efficiency through energy-performance trade-offs, query optimization, and execution-time decisions [8, 13, 14, 25], with recent extensions to proactive energy management, join reordering, and cloud analytics [3, 9, 19]. These studies show that query-level decisions can reduce energy, but they do not address P-core/E-core selection on heterogeneous multicore processors.

Operator-aware query processing. Analytical operators interact differently with modern hardware, including scans and access-path selection [5, 10], hash joins [6], hash aggregation [11, 17], and sorting [12]. These studies motivate operator-aware analysis, but focus mainly on performance rather than energy-aware core-type selection.

Heterogeneous-core scheduling and database workloads. Heterogeneous multicore processors introduce scheduling choices across cores with different performance and efficiency characteristics. Prior work has studied Intel hybrid-processor energy behavior [22], OS support such as Intel Thread Director [21], and P-core/E-core database workload behavior [23]. Our work focuses on query-level energy-aware core-type selection for analytical SQL using workload characteristics such as cardinality, skew, and input distribution.

Top-Down Microarchitectural Analysis. We use Top-Down Microarchitectural Analysis (TMA) to interpret execution-time differences [26]. TMA attributes CPU pipeline slots to useful work or major bottlenecks: *Retiring* denotes completed work, while *Bad Speculation*, *Frontend Bound*, *Memory Bound*, and *Core Bound* denote losses from incorrect speculation, frontend supply limits, memory-hierarchy stalls, and non-memory backend limits, respectively. In our figures, Backend Bound is decomposed into Memory Bound and Core Bound.

3 Problem Statement and Approach

We study query-level core type selection for analytical database execution on heterogeneous multi-core processors. Given a query before execution, the system chooses whether to run it on one P-core or one E-core. We define the decision problem as follows:

- **Input:** pre-execution query features, including operator type, cardinality ratio, and distribution-related features such as key skew or sort-key generation pattern. These features can be obtained or approximated from the query template, database statistics, or workload metadata.
- **Output:** the selected core type, either P-core or E-core.
- **Objective:** minimize above-idle system energy for the query.

Our hypothesis is that operator type alone is insufficient for this decision. The underlying trade-off is that P-cores generally shorten execution time, whereas E-cores usually lower instantaneous system power. The lower above-idle system energy therefore depends on the runtime–power trade-off, which can change within the same operator as cardinality, skew, or input distribution changes. Although hash aggregation, hash join, and sort exhibit different hardware behavior, their energy-efficient core type cannot be captured by a fixed operator-to-core mapping. Therefore, energy-aware core-type selection should use query-level workload characteristics rather than fixed P-core/E-core choices or operator-only rules.

We evaluate this hypothesis in two stages:

- **First**, Section 4 systematically characterizes P-core/E-core time and above-idle energy behavior using queries with a dominant hash aggregation, hash join, or sort operator.
- **Second**, Section 5 uses these observations to implement and evaluate a query-aware policy as a proof of concept without runtime instrumentation.

This evaluation tests whether database-level query information provides useful signals for energy-aware core-type selection.

4 Operator-Level Energy Characterization

4.1 Experimental Setup

We run experiments on an Intel Core i5-14400F heterogeneous multicore processor with Hyper-Threading disabled, 16 GiB DDR5 memory, Ubuntu 24.04.4 LTS, Linux 6.8.0-101-generic, and DuckDB 1.1.3. All data resides in main memory. To isolate core-type effects, we pin each query to the target core type and use one DuckDB execution thread to compare single-core P-core and E-core execution. We measure system power using a high-precision Yokogawa WT1800 power meter with a basic power accuracy of ±0.1%. Each configuration is executed three times, and we report mean execution time and above-idle system energy, $E_{\text{above-idle}} = E_{\text{sys}} - P_{\text{idle}}T$, where E_{sys} is integrated system energy, P_{idle} is idle power, and T is execution time. We also collect TMA metrics with Linux `perf stat` over the same query interval and report them in the per-operator figures.

We use complete DuckDB queries dominated by one target operator rather than isolated operators. All inputs are generated deterministically from a synthetic `row_id`. Aggregation and sort use $N = 10^8$ input tuples; hash join uses $N = 10^8$ probe tuples and a build relation whose size is controlled by the build-side cardinality ratio. For cardinality ratio r, the key-domain size is $K = \text{round}(rN)$.

For hash aggregation, we vary group cardinality ratio and key distribution over *uniform*, *heavy50*, and *heavy90*. For hash join, we vary build-side cardinality ratio and probe-key distribution over *uniform*, *skew50*, and *skew90*. For sort, we vary sort-key cardinality ratio and key-generation pattern over *modulo-repeated* and *pseudo-random*. The heavy/skewed distributions assign 50% or 90% of tuples to one hot key, while modulo-repeated and pseudo-random use deterministic modulo and multiplicative mappings, respectively.

4.2 Queries Dominated by Hash Aggregation

Figure 1 reports execution time and above-idle system energy for queries dominated by Hash Aggregation. We compare three key distributions: *uniform*, *heavy50*, and *heavy90*. Across all distributions, P-cores execute faster than E-cores, with the gap increasing at higher cardinalities as more active groups are maintained in the hash table.

Energy behavior is more workload-dependent. Although P-cores are consistently faster, the above-idle system energy gap between P-cores and E-cores is often small under *heavy50* and, for some cardinalities, under *uniform*. At high cardinalities, however, P-cores become more favorable because their runtime advantage offsets

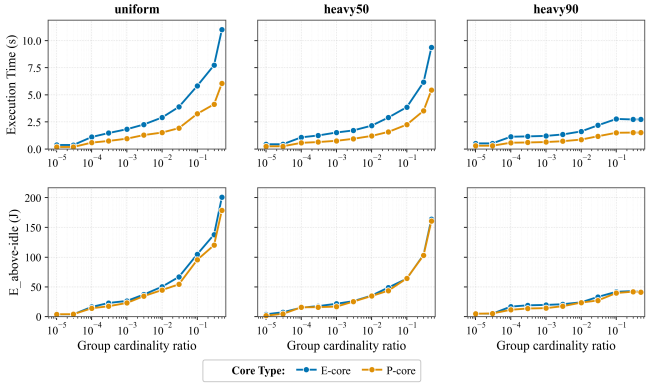

Fig. 1. Hash aggregation performance and above-idle energy under three key distributions. P-cores consistently reduce execution time, while the energy gap varies with group cardinality and key distributions.

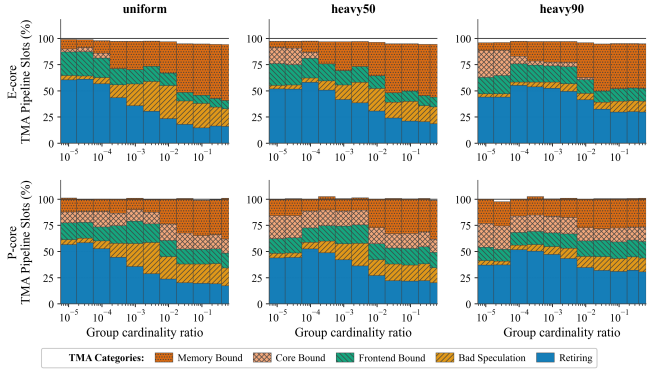

Fig. 2. TMA breakdown for hash aggregation. Higher group cardinality increases memory-bound behavior, especially for uniform keys, indicating growing hash-table pressure on the memory hierarchy.

their higher power. These results show that both group cardinality and key skew affect the P-core/E-core energy trade-off.

Figure 2 reports the Top-Down Microarchitecture Analysis results for the aggregation workload. At higher cardinalities, especially under the *uniform* distribution, a larger fraction of pipeline slots is attributed to memory-related stalls and a smaller fraction to retiring work. These measurements provide microarchitectural context for the increase in execution time at high group cardinalities.

Overall, hash aggregation shows that operator name alone is insufficient for energy-aware core-type selection. Even within one operator type, group cardinality and key skew substantially change the P-core/E-core energy trade-off.

4.3 Queries Dominated by Hash Join

Figure 3 reports execution time and above-idle system energy for queries dominated by hash join. The workload value denotes the build-side cardinality ratio, and we compare three probe-key distributions: *uniform*, *skew50*, and *skew90*. P-cores are faster across all settings, but the energy-efficient core type depends strongly on the probe-key distribution.

Under *uniform* probes, P-cores are consistently more energy-efficient. Uniform probes spread accesses over many build-side keys. As the build side grows, this access pattern creates more memory-hierarchy pressure, which appears more prominently on E-cores in the TMA results. Under *skew50* and *skew90*, E-cores consume less above-idle system energy despite longer execution time. This is likely because repeated probes to hot keys improve locality in the build-side hash table, reducing the E-core memory-stall penalty and allowing its lower power to dominate the remaining runtime overhead. Thus, uniform probes favor P-cores, while skewed probes favor E-cores in our experiments.

Figure 4 provides microarchitectural context for the runtime behavior. Under *uniform* probes, increasing the build-side cardinality substantially increases the memory-bound fraction on E-cores, while the P-core profile changes only modestly. This suggests that uniform probes expose E-core execution more strongly to memory-hierarchy stalls as the build-side key domain grows, increasing the E-core runtime penalty. Under *skew50* and *skew90*, repeated probes to hot keys improve locality in the build-side hash table, which is reflected by lower memory-bound fractions and higher retiring fractions on E-cores. The reduced E-core stall pressure lowers the runtime penalty relative to P-cores; combined with lower E-core power, this leads to lower above-idle system energy under skewed probe distributions.

Overall, hash-join core selection is primarily shaped by probe-key distribution, while build-side cardinality mainly changes the size of the P-core/E-core performance–energy gap.

4.4 Queries Dominated by Sort

Figure 5 reports execution time and above-idle system energy for queries dominated by Sort. The cardinality ratio denotes the sort-key cardinality ratio, and we compare two key-generation patterns: *modulo-repeated* and *pseudo-random*. Across all tested configurations, P-cores execute sort queries faster than E-cores. However, the energy benefit of P-core execution depends on the input pattern: P-cores are clearly favorable for *pseudo-random* inputs, while the two core types are often close in above-idle system energy for *modulo-repeated* inputs.

The input pattern also affects above-idle system energy. The *pseudo-random* pattern generally consumes more above-idle system energy than *modulo-repeated*, particularly at medium-to-high cardinality ratios. Although both patterns use the same sort-key domain, the less regular input order of *pseudo-random* leads to longer execution time and higher above-idle system energy.

Figure 6 reports the Top-Down Microarchitecture Analysis results for the sort workload. As the sort-key cardinality ratio increases, E-core execution shows a lower retiring fraction and a larger fraction of memory-bound pipeline slots, while P-core execution remains more stable across the tested configurations. This suggests that E-core execution becomes more sensitive to memory-hierarchy stalls as the sort-key domain grows, helping account for the larger runtime advantage of P-cores for sort queries. Overall, energy behavior for sort queries is input-dependent. P-cores are more energy-efficient

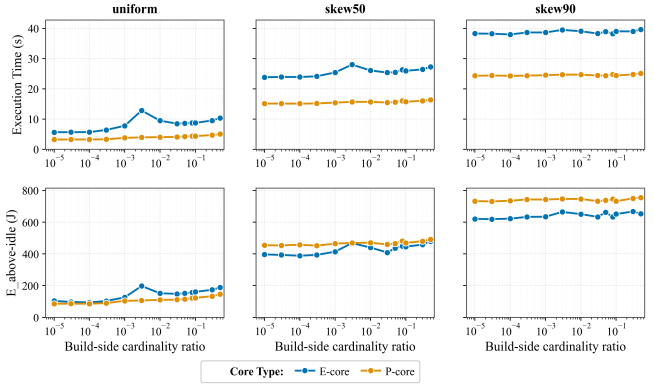

Fig. 3. Hash join performance and above-idle energy under three probe-key distributions. P-cores consistently reduce execution time, while energy preference shifts from P-cores under uniform probes to E-cores under skewed probes.

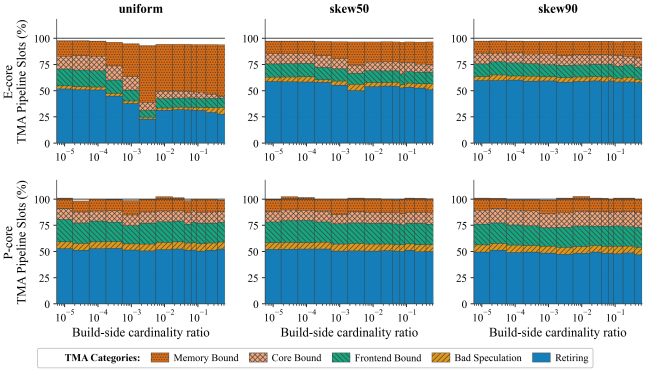

Fig. 4. TMA breakdown for hash join. Uniform joins become more memory-bound as build-side cardinality increases, especially on E-cores, while skewed joins show more stable pipeline behavior.

for *pseudo-random* inputs, while *modulo-repeated* inputs show a smaller P-core/E-core above-idle system energy gap.

4.5 Implications for Core-Type Selection

The operator-level results show that energy-efficient core-type selection is not an operator-only decision. Although P-cores consistently reduce execution time, their higher power does not always lead to lower above-idle system energy. Instead, the preferred core type changes with query-level workload characteristics, including aggregation cardinality and skew, join build-side cardinality and probe-key distribution, and sort-key cardinality and input pattern.

This workload dependence explains why a database system can provide useful information beyond what is visible to a general-purpose OS scheduler. Query features such as operator type, estimated cardinality, key-distribution statistics, and input-order properties can be obtained or approximated from query plans, optimizer estimates, and workload metadata before execution. The hypothesis stated in Section 3—that operator type alone does not determine the energy-efficient core type—is supported by the observations in Sections 4.2–4.4. We therefore implement and evaluate a query-feature-based core-selection policy in Section 5.

5 Query-Aware Core-Type Selection Policy

5.1 Policy Implementation and Features

We prototype the policy with a learned energy model that uses only pre-execution query features and requires no runtime instrumentation. Each training sample represents a candidate execution of one query configuration on a specific core type. In this study, the two candidates are single-core P-core execution and single-core E-core execution. The model predicts the above-idle system energy of each candidate, and the policy selects the core type with lower predicted energy.

The input features include operator type, candidate core type, workload-specific cardinality ratio, and distribution category, such as aggregation-key skew, probe-key skew, or sort-key generation pattern. Although the implementation retains input size and core count as generic fields, both are fixed in this study and therefore do not drive the reported core-selection results. These features are available before execution from the query template, optimizer statistics, or workload metadata.

Categorical features are encoded with one-hot encoding. We implement the energy predictor as a random forest regression model with 500 trees and a minimum leaf size of 2 using `scikit-learn`. The model predicts above-idle system energy per tuple. This design does not require runtime hardware counters or online profiling; learning is used only as a compact implementation of the core-selection policy.

5.2 Evaluation Workload Set and Metrics

Each sample in the evaluation set is a query configuration defined by its operator type, cardinality ratio, and distribution setting, rather than a random sample of rows from a fixed table. All workloads use deterministic synthetic data with base input size $N = 10^8$ tuples. The full set contains 800 configurations: 300 hash aggregation configurations from 100 group cardinality ratios and three key distributions, 300 hash join configurations from 100 build-side cardinality ratios and three probe-key distributions, and 200 sort configurations from 100 sort-key cardinality ratios and two key-generation patterns.

For each configuration, we measure single-core P-core and E-core execution in DuckDB with one execution thread, repeating each query three times and averaging execution time and above-idle system energy. We use Linux default scheduling as the non-database-aware baseline and report held-out energy saving as Saving $= (E_{\text{Linux}} - E_{\text{Model}})/E_{\text{Linux}} \times 100\%$, where E_{Linux} and E_{Model} are total above-idle system energy under the baseline and model-selected policy.

5.3 Core-Type Selection Results

Figure 7 compares the learned core-selection policy with Linux default scheduling. Across all 800 query configurations, the policy reduces total above-idle system energy by **18.14%**. The largest benefit comes from hash join, with a **22.23%** total saving. Hash

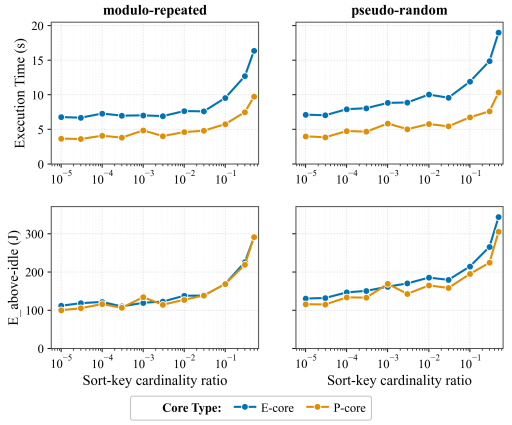

Fig. 5. Sort performance and above-idle energy under two key-generation patterns. P-cores are faster, while pseudo-random inputs show a clearer P-core energy advantage than modulo-repeated inputs.

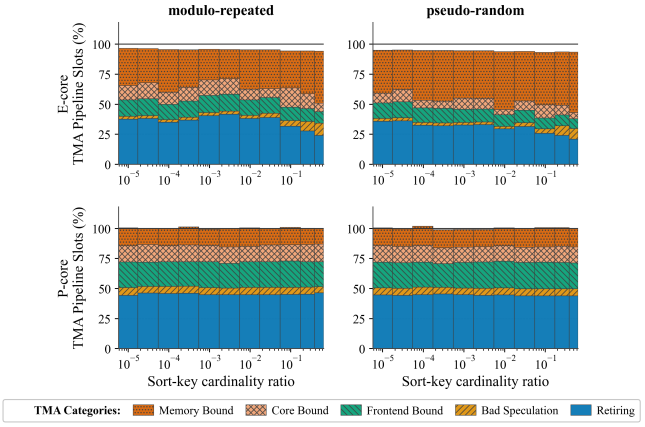

Fig. 6. TMA breakdown for sort. Higher sort-key cardinality increases E-core memory-bound behavior, especially for pseudo-random inputs.

aggregation and sort also improve, reducing total energy by **6.87%** and **9.78%**, respectively.

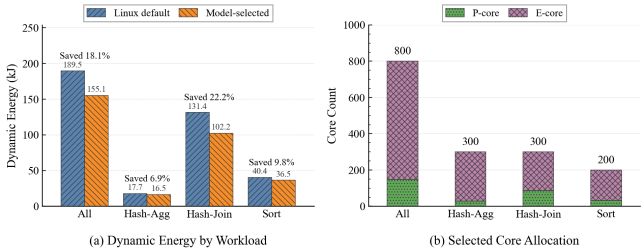

Fig. 7. Model-guided core-type selection compared with Linux default scheduling. (a) The policy reduces total above-idle system energy by 18.14% across 800 configurations, with the largest gain on hash join. (b) The policy selects mostly E-cores but still assigns 148 configurations to P-cores, showing that it is workload-aware rather than an all-E-core rule.

Figure 7(b) shows that the policy selects E-cores for most configurations overall, 652 out of 800, but it is not equivalent to an all-E-core rule. It still assigns 148 configurations to P-cores when the predicted runtime advantage is large enough to offset higher power. This mixed allocation appears across all operators: hash aggregation is mostly assigned to E-cores, with 28 P-core and 272 E-core selections; hash join has the largest number of P-core selections, with 87 P-core and 213 E-core selections; and sort also favors E-cores, with 33 P-core and 167 E-core selections. These results show that the policy uses workload characteristics to distinguish cases where lower-power E-core execution is sufficient from cases where faster P-core execution is more energy-efficient.

6 Conclusion

This paper studied energy-aware core-type selection for analytical database queries on a heterogeneous multi-core processor platform. Using queries whose dominant operators are hash aggregation, hash join, or sort, we showed that the energy-efficient core type is not determined by operator type alone. Instead, it depends on query-level characteristics such as cardinality ratio, key skew, and input distribution. These results show that database systems can provide useful core-selection signals that are not directly available to general-purpose operating-system schedulers.

We further evaluated an implementation of query-aware core-type selection that uses only pre-execution query features and requires no runtime instrumentation. Across the evaluated workloads, the policy reduced total above-idle system energy by **18.14%** compared with Linux default scheduling. Although our evaluation is limited to single-threaded queries with a dominant operator on a heterogeneous multi-core processor platform, the results provide initial evidence that database-managed heterogeneous execution is a promising direction for energy-aware query processing.

This work explores an initial step toward database-managed execution on heterogeneous multi-core processors. Future work will study multi-operator plans, parallel execution, and cross-platform calibration on heterogeneous processors.

Acknowledgments

This work has been partially supported by the Research and Development Project of the Enhanced Infrastructures for Post-5G Information and Communication Systems (JPNP20017) of NEDO. In addition, this work was in part supported by JST SPRING, Japan Grant Number JPMJSP2108.

References

[1] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Udit Gupta, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. 2023. Carbon explorer: A holistic framework for designing carbon aware datacenters. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 118–132.

[2] Michail Bachras and Hans-Arno Jacobsen. 2025. Environmental Footprints of Query Processing: A Vision for Sustainable Database Architectures. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4064–4072.

[3] Ladjel Bellatreche, Fouad Djellali, Wojciech Macyna, and Carlos Ordonez. 2023. Energy-aware query processing: A case study on join reordering. In *2023 IEEE International Conference on Big Data (BigData)*. IEEE, 3743–3752.

[4] Sebastian Breß, Henning Funke, and Jens Teubner. 2016. Robust query processing in co-processor-accelerated databases. In *Proceedings of the 2016 international conference on management of data*. 1891–1906.

[5] David Broneske, Sebastian Breß, and Gunter Saake. 2013. Database scan variants on modern CPUs: A performance study. In *International Workshop on In-Memory Data Management and Analytics*. Springer, 97–111.

[6] Shimin Chen, Anastassia Ailamaki, Phillip B Gibbons, and Todd C Mowry. 2007. Improving hash join performance through prefetching. *ACM Transactions on Database Systems (TODS)* 32, 3 (2007), 17–es.

[7] Alex de Vries-Gao. 2026. The carbon and water footprints of data centers and what this could mean for artificial intelligence. *Patterns* 7, 1 (2026).

[8] Binglei Guo, Jiong Yu, Dexian Yang, Hongyong Leng, and Bin Liao. 2022. Energy-efficient database systems: A systematic survey. *Comput. Surveys* 55, 6 (2022), 1–53.

[9] Yuto Hayamizu, Masaru Kitsuregawa, and Kazuo Goda. 2024. Proactive Energy Management in Database Systems. *ACM SIGENERGY Energy Informatics Review* 4, 5 (2024), 154–159.

[10] Michael S Kester, Manos Athanassoulis, and Stratos Idreos. 2017. Access path selection in main-memory optimized data systems: Should I scan or should I probe?. In *Proceedings of the 2017 ACM international conference on management of data*. 715–730.

[11] Laurens Kuiper, Peter Boncz, and Hannes Mühleisen. 2024. Robust external hash aggregation in the solid state age. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 3753–3766.

[12] Laurens Kuiper and Hannes Mühleisen. 2023. These rows are made for sorting and that's just what we'll do. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2050–2062.

[13] Willis Lang, Stavros Harizopoulos, Jignesh M Patel, Mehul A Shah, and Dimitris Tsirogiannis. 2012. Towards energy-efficient database cluster design. *arXiv preprint arXiv:1208.1933* (2012).

[14] Willis Lang, Ramakrishnan Kandhan, and Jignesh M Patel. 2011. Rethinking query processing for energy efficiency: Slowing down to win the race. *IEEE Data Eng. Bull.* 34, 1 (2011), 12–23.

[15] Dumitrel Loghin, Bogdan Marius Tudor, Hao Zhang, Beng Chin Ooi, and Yong Meng Teo. 2015. A performance study of big data on small nodes. *Proceedings of the VLDB Endowment* 8, 7 (2015), 762–773.

[16] Tobias Mühlbauer, Wolf Rödiger, Robert Seilbeck, Alfons Kemper, and Thomas Neumann. 2014. Heterogeneity-Conscious Parallel Query Execution: Getting a better mileage while driving faster!. In *Proceedings of the Tenth International Workshop on Data Management on New Hardware*. 1–10.

[17] Ingo Müller, Peter Sanders, Arnaud Lacurie, Wolfgang Lehner, and Franz Färber. 2015. Cache-efficient aggregation: Hashing is sorting. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 1123–1136.

[18] Shiqiang Nie, Yingming Liu, Jie Niu, and Weiguo Wu. 2024. CAL: Core-Aware Lock for the big.LITTLE Multicore Architecture. *Applied Sciences* 14, 15 (2024), 6449.

[19] Carlos Ordonez, Wojciech Macyna, and Ladjel Bellatreche. 2024. Energy-aware analytics in the cloud. In *Proceedings of the International Workshop on Big Data in Emergent Distributed Environments*. 1–6.

[20] Nikhil Rukmabhatla, Rajshree Chabukswar, Sneha Gohad, and Michael Chynoweth. 2021. Intel performance hybrid architecture & software optimizations. *Intel, Tech. Rep.* (2021).

[21] Juan Carlos Saez and Manuel Prieto-Matias. 2022. Evaluation of the intel thread director technology on an alder lake processor. In *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems*. 61–67.

[22] Robert Schöne, Markus Velten, Daniel Hackenberg, and Thomas Ilsche. 2024. Energy efficiency features of the intel alder lake architecture. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*. 95–106.

[23] Rathijit Sen. 2024. Performance or Efficiency? A Tale of Two Cores for DB Workloads. In *Proceedings of the 20th International Workshop on Data Management on New Hardware*. 1–5.

[24] Arman Shehabi, Alex Newkirk, Sarah J Smith, Alex Hubbard, Nuoa Lei, Md Abu Bakar Siddik, Billie Holecek, Jonathan Koomey, Eric Masanet, and Dale Sartor. 2024. 2024 united states data center energy usage report. (2024).

[25] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. 2012. PET: reducing database energy cost via query optimization. *Proceedings of the VLDB Endowment* 5, 12 (2012), 1954–1957.

[26] Ahmad Yasin. 2014. A top-down method for performance analysis and counters architecture. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 35–44.