Powerful GPUs or Fast Interconnects: Analyzing Relational Workloads on Modern GPUs

Marko Kabić
Systems Group, D-INFK,
ETH Zurich, Switzerland
marko.kabic@inf.ethz.ch

Bowen Wu
Systems Group, D-INFK,
ETH Zurich, Switzerland
bowen.wu@inf.ethz.ch

Jonas Dann
Systems Group, D-INFK,
ETH Zurich, Switzerland
jonas.dann@inf.ethz.ch

Gustavo Alonso
Systems Group, D-INFK,
ETH Zurich, Switzerland
alonso@inf.ethz.ch

ABSTRACT

In this study we explore the impact of different combinations of GPU models (RTX3090, A100, H100, GraceHoppers - GH200) and interconnects (PCIe 3.0, PCIe 4.0, PCIe 5.0, and NVLink 4.0) on various relational data analytics workloads (TPC-H, H2O-G, ClickBench). We present MaxBench, a comprehensive framework designed for benchmarking, profiling, and modeling these workloads on GPUs. Beyond delivering detailed performance metrics, MaxBench estimates query execution performance using a novel cost model. With this model, we move beyond traditional metrics such as arithmetic intensity and GFlop/s and suggest using instead the notions of *characteristic query complexity* and *characteristic GPU efficiency*, as more suitable metrics for data analytics workloads. We conduct an extensive experimental analysis with MaxBench across different combinations of GPU models and interconnects on various data analytics workloads. The insights from this analysis reveal the trade-offs between GPU computing capacity and interconnect bandwidth on query processing. Using this cost model, we also examine future trends by investigating how enhancements in interconnect bandwidth or GPU efficiency would affect performance in the future.

PVLDB Reference Format:
Marko Kabić, Bowen Wu, Jonas Dann, and Gustavo Alonso. Powerful GPUs or Fast Interconnects: Analyzing Relational Workloads on Modern GPUs. PVLDB, 18(11): 4350 - 4363, 2025.
doi:10.14778/3749646.3749698

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://gitlab.inf.ethz.ch/PUB-SYSTEMS/eth-dataprocessing/Maximus.

1 INTRODUCTION

Rapid data growth and a variety of workloads have created a demand for more efficient processing solutions in modern data centers. To meet this demand, there has been a clear shift toward heterogeneous hardware architectures that incorporate specialized accelerators [1, 38]. Among these, Graphics Processing Units (GPUs) stand out due to their exceptional parallel processing capabilities and high memory bandwidth, making them particularly well-suited for data-intensive tasks. Their suitability is further reinforced by the rising computational demands of AI and machine learning workloads, which have driven the widespread adoption of GPUs in data center environments. This, in turn, has led to an increasing interest in the development of GPU-accelerated data analytics engines [6, 8, 11, 14, 17–19, 21, 24, 25, 28, 32, 36, 44, 52, 56].

Although early generations of GPUs were limited by small memory capacities and slow data transfer over interconnects, recent advancements have significantly expanded GPU memory and improved interconnect speeds, making them increasingly viable for large-scale data processing. With the development of fast interconnects, such as NVLink, featuring high bandwidth and more powerful GPUs, the interconnect may no longer be the main bottleneck [30].

Despite these advancements, several questions still remain: *(i)* Is there still a necessity for powerful GPUs, or do fast interconnects suffice? *(ii)* How do different GPU models and interconnect technologies affect the performance of data analytics? *(iii)* What impact does increasing interconnect bandwidth and GPU efficiency have on query execution?

Answering these questions is challenging for two reasons. First, it requires comprehensive modeling of both interconnect performance and GPU capabilities, requiring a robust benchmarking and profiling framework. Second, doing so involves characterizing query complexity on GPUs and evaluating their efficiency. Traditionally, arithmetic intensity (operations per byte) has been used to measure query complexity with the roofline model. However, for data analytics workloads, which tend to be memory-bound, such a metric often yields very low values on GPUs. Additionally, GPU efficiency is typically measured in GFlops — a metric suitable for compute-intensive tasks but inadequate for data analytics.

We address these challenges, by making the following contributions:

(1) We present MaxBench – a framework for benchmarking, profiling, and modeling analytics workloads on heterogeneous hardware. MaxBench provides detailed performance metrics, including the breakdown of CPU↔GPU data transfers and operator-level breakdown of the query execution time.
(2) Using MaxBench, we provide a comprehensive analysis running the TPC-H, H2O-G, and ClickBench benchmarks on different GPU models (RTX3090, A100, H100, GH200), as well as different interconnects (PCIe 3.0, PCIe 4.0, PCIe 5.0, NVLink 4.0).
(3) Based on the comprehensive analysis, we introduce the notions of characteristic query complexity, characteristic GPU efficiency and characteristic interconnect efficiency.
(4) Using this characterization, we introduce a novel cost model that allows us to estimate the computation and communication costs of evaluated benchmarks on any combination of the evaluated GPUs and interconnects.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749698

(5) Using this cost model, we derive theoretical limits on the ratios of CPU-GPU data transfer costs and the GPU query execution costs, in the limit case when the data size tends to infinity.
(6) Through extensive benchmarks, we provide numerous insights, including identifying a crossing point when the workloads shift from being interconnect-bound to being GPU-bound for different workloads and various data-movement models.
(7) We show how optimizations, such as selective data transfer based on the filtering predicate can be incorporated into the cost model. In the experimental section, we evaluate the impact of this optimization on different hardware configuration.
(8) With the introduced analytical models, we estimate the performance impact of future GPU devices and interconnects, when their efficiencies are scaled beyond their current capabilities.

2 RELATED WORK

There is extensive related work on GPU query processing: relational operators on GPUs [27, 41, 43, 45, 49, 52], multi-GPU query processing [37, 42, 48], or CPU-GPU co-processing of queries [23, 29] covered in detail by surveys [10, 12, 40, 55]. We have structured the discussion around two main categories of related work: *(i)* experimental analyses and modeling, and *(ii)* analyzing the interconnect.

Experimental Analyses and Modeling. Cao et al. [12] analyze the performance and resource utilization of different GPU database systems for a specific GPU. Yuan et al. [55] propose a GPU-based query engine to understand the effect of query characteristics, software optimization, and hardware configuration. Their evaluation is constrained to machines with limited interconnects (PCIe 3.0 at most) and currently outdated GPUs, whereas our work includes a greater variety of interconnects and modern data center-grade GPUs. On building the Tensor Query Processor (TQP) with tensor abstractions, He et al. [22] also evaluate its performance with TPC-H queries, however, without looking into data transfers over the interconnect. Shanbhag et al. [44] advocate against the co-processor model, focusing on the effect of memory bandwidth on performance. Lastly, Arefyeva et al.[4] discuss memory management strategies for relational data processing on GPUs.

On the modeling side, many papers have modeled the performance of singular relational operators on the GPU (e.g., [44, 52]) and data transfer costs (e.g., [55]). Existing models introspect the operator implementations but are tied to those implementations. In the HPC domain, CPU-GPU transfers in supercomputers with PCIe 2.0 and PCIe 3.0 interconnects have been studied [50].

Tackling the Interconnect Bottleneck. Numerous ideas have been proposed to alleviate the limited bandwidth between the CPU and the GPU. We refer interested readers to [40] for a thorough review. Most recently, Yogatama et al. [53] develop a fine-grained semantic-aware caching policy for CPU-GPU co-processing. Yuan et al. [54] overcome the limited interconnect bandwidth by aggregating the PCIe bandwidth of multiple GPUs to serve a single GPU. Beyond traditional CPU-GPU interconnects, Lutz et al. [30, 31] developed hash join algorithms for NVLink. Lastly, Afroozeh et al.[2] explore lightweight encoding techniques to take pressure off the interconnect and memory subsystem of the GPU with minimal performance impact. Our work, MaxBench, is orthogonal to this line of work because MaxBench can be used as a platform to evaluate optimizations proposed in previous work and to demonstrate their effects on end-to-end query runtime. Some of these optimizations can be implemented in MaxBench, e.g. as policies (Section 3).

3 MAXBENCH ARCHITECTURE

MaxBench is a framework for benchmarking, profiling, and modeling performance of data analytics on heterogeneous architectures, built on top of the Maximus query engine [26], using a push-based, vectorized execution model [9, 57]. It employs the Apache Arrow data format [46] for efficiently ingesting the data to the CPU memory, the RAPIDS cuDF dataframe library [3] for GPU execution, and integrates the Caliper Profiling Toolbox [7] for detailed code instrumentation. It is architecturally organized into three layers (Figure 1). The *Input Layer*, facilitates the selection of the benchmark suite and the dataset to be executed. The *Profiling Configuration Layer* allows users to customize the execution parameters, including the execution flow and profiling granularity. The *Analysis & Modeling Layer* is dedicated to performance metrics.

3.1 Input Layer

Supported Benchmarks. In the input layer, users can specify the benchmark suite to run. The framework currently supports the full TPC-H suite [47] – modeling the traditional data warehouse analytics workloads, the Clickbench suite [15] – modeling large-scale web analytics workloads, and the H2O-G (group-by) suite [20] – modeling tasks frequently encountered in data science workloads.

Datasets. The datasets component enables the data generation for the specified benchmarking suite. During data generation, the size of the data can be set using the scaling factor. Moreover, certain benchmarks, such as H2O-G (group-by), offer additional parameters that allow users to control attributes like the percentage of NULLs or the data distribution by specifying the number of groups.

Data Formats. The system currently supports reading CSV and Parquet files by using the Arrow's and cuDF's readers.

3.2 Profiling Configuration Layer

Data Location. The users can choose whether the input data initially resides on a CPU or on a GPU. For the CPU option, it is further possible to choose whether it should be stored in a pinned (page-locked) or non-pinned region. This can significantly affect the interconnect bandwidth on PCIe interconnects. However, the pinned memory is a scarce resource and should be used with caution, as it prevents the OS from swapping pages. Similarly, it can be specified whether the query result should stay on the GPU or be transferred to the CPU after the execution. Traditionally, query inputs and results are expected to be stored in the CPU memory. However, if subsequent processing is expected, such as within AI/ML pipelines, retaining the data on the GPU may be beneficial.

Copy Policy. The data copying policy determines whether the CPU↔GPU data transfers occur on a *per-dataset* or *per-query* basis. In a per-dataset approach, the entire dataset is transferred to the GPU before executing queries. Conversely, in a per-query approach, only the specific input data for each query is copied to the GPU prior to its execution. Each option has its advantages: per-dataset copying can be more efficient if the entire dataset fits within the GPU's memory capacity, which can benefit subsequent queries,

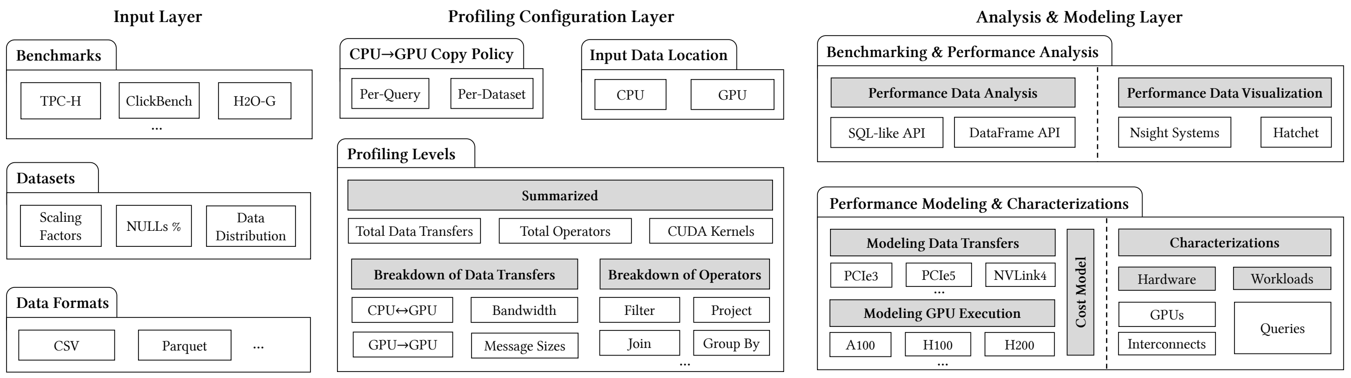

Figure 1: The architecture of MaxBench.

amortizing the upfront data transfer costs. Meanwhile, per-query copying is advantageous when the input data required for individual queries is significantly smaller than the full dataset.

Profiling Levels. The system offers multiple profiling levels to accommodate different profiling granularities. At the *summarized* level, it tracks only the overall time spent on CPU↔GPU data transfers and GPU query execution, along with capturing the duration spent in top CUDA kernels. For more granular insights into data transfers, users can enable the *breakdown of data transfers* profiling level. This option provides detailed metrics on the time spent in CPU↔GPU data transfers as well as information on bandwidth, message count and total sizes. Enabling the *breakdown of operators* records the time spent in each specific operator, such as filter, project, join, group-by, and others.

3.3 Analysis & Modeling Layer

Benchmarking & Performance Analysis. The profiler output can be further analyzed using SQL- or DataFrame-like APIs. The integration with third-party tools enables visualization through NVIDIA's official Nsight Systems profiler [16] and the Python-based Hatchet library [5], which facilitates performance analysis.

Performance Modeling & Characterizations. In addition to analyzing the profiling data, MaxBench employs this data to model the time spent on data transfers and operator execution on GPUs. The time spent in data transfers can be modeled for different types of interconnects. Similarly, the time spent in operators can be modeled for different types of GPUs. It also provides the characterization of hardware (GPUs and interconnects) and workloads (queries) through the notions of *characteristic efficiency* and *characteristic complexity* introduced in the next section.

4 COST MODELING & CHARACTERIZATION

MaxBench assumes the query execution cost consists of two components: the CPU↔GPU data transfers cost, impacted by the interconnect and the operator costs, depending on the GPU used.

4.1 Data Transfers Cost (Communication)

The cost of CPU↔GPU data transfers depends on the type of interconnect used. This interconnect is typically constrained by its peak bandwidth, denoted as $B_{\max}$, which represents the maximum rate at which data can be transferred. However, the actual achieved bandwidth depends on the size of the data being transferred. Smaller message sizes may not fully utilize the available bandwidth. For a given data size s, we denote the achieved bandwidth as $B(s)$. We model this relationship as follows:

$$B(s) = B_{max} \cdot \frac{s}{s + s_0} \tag{1}$$

The parameter s_0 represents the data size at which half of the peak bandwidth, $B_{\max}/2$, is reached. Intuitively, smaller values of s_0 indicate that saturation occurs for smaller message sizes, while larger s_0 values suggest that larger data sizes are necessary to reach the bandwidth saturation point.

Once the bandwidth is known, we can estimate the time spent in data transfers when executing query q, denoted by $T_{q,\mathtt{comm}}(s)$ as:

$$T_{q,\mathtt{comm}}(s) = \frac{s}{B(s)} \tag{2}$$

Observe that the latency is ignored in this formula. In our experiments, the latency was in the range $0.8 - 1.2\mu s$ across all tested interconnects, negligible for the purpose of data analytics.

4.2 Operators Cost (Computation)

The operators cost represents the time spent executing all operators in a query plan, such as join, group-by, order-by, filter, project, limit, and others. This cost does not involve the data transfers cost, but rather the execution of the operators, once the data is already residing on a GPU. For this reason, this cost depends solely on the GPU type and is independent of the interconnect being used.

In our analysis, presented in the results section, we observe that the cost associated with executing operators increases linearly with respect to the size of the dataset. Thus, we use a linear model to estimate the execution time $T_{q,\mathtt{op}}$ of operators on a GPU device d, based on both the data size s and the query q:

$$T_{q,\mathtt{op}}(s) = a_{q,d} \cdot s + b_{q,d} \tag{3}$$

Observe that the slope $a_{q,d}$ and the intercept $b_{q,d}$ depend on both the query q and the GPU d. Now, we define the operator's cost model as the ordered pair of both the slope and intercept matrix.

Definition 4.1. For a query set Q and the GPU device set D, the operators cost model is defined as an ordered pair of matrices (A, B), where $A = [a_{q,d}]_{q\in Q, d\in D}$ is the slope matrix and $B =$

$[b_{q,d}]_{q \in Q, d \in D}$ is the intercept matrix. All entries in A must be strictly positive, and all entries in B must be non-negative.

4.3 Incorporating Optimizations

The proposed cost model can be extended to include optimizations that fall into three categories: (1) reducing data transfer costs (e.g., compression), (2) minimizing computation costs (e.g., kernel fusion), and (3) reducing both. For instance, performing initial filtering on the CPU and offloading the remaining query plan to the GPU reduces communication costs by transferring less data and lowers GPU computation costs by skipping the filtering step. This section demonstrates how such optimizations, using selective data transfers as an example, can be integrated into the cost model. Other optimizations can be incorporated similarly. In practice, selective data transfers based on predicates are available in some of the state-of-the-art engines [33, 39, 54].

Communication Costs. Let us generalize the communication cost from Section (4.1), by separating the cost of transferring the query input and the query output, as: $T_{q,\mathtt{comm}}(s) = T^{\mathtt{in}}_{q,\mathtt{comm}}(s) + T^{\mathtt{out}}_{q,\mathtt{comm}}(s)$. Let $T_{\mathtt{filter}}(s)$ denote the cost of filtering data of size s on the CPU, where $\sigma(s) \in (0, 1]$ is the filter selectivity and $\sigma(s) \cdot s$ is the size of the filtered data. Now we extend the communication cost from Eq. (2), and define perceived bandwidth $B_{perc}(s)$, as follows:

$$T^{\mathtt{in}}_{q,\mathtt{comm}}(s) = T_{\mathtt{filter}}(s) + \frac{\sigma(s) \cdot s}{B(\sigma(s) \cdot s)}, \qquad B_{perc} = \frac{s}{T^{\mathtt{in}}_{q,\mathtt{comm}}(s)} \tag{4}$$

If the query result is copied back to the CPU, we can still use Eq. (2) and (1) to model these transfer costs, since selective filtering only affects the input data transfers.

Computation Cost. Let $q' = q \setminus \{\sigma\}$ denote the query plan without the initial filter operators that are used for selective data transfers. For q', we can still use the cost model from Section 4.2.

4.4 Hardware & Query Characterization

Interconnects. Equation (1) suggests the interconnects are characterized by the peak bandwidth B_{max} and the s_0 parameter, which controls how fast the peak is achieved. However, for large enough data sizes, the bandwidth will be close to the peak bandwidth. For this reason, it makes sense to assume that the peak bandwidth is the main characteristic of the interconnect.

Definition 4.2. For an interconnect IC, we define the characteristic interconnect efficiency χ_{ic} as:

$$\chi_{ic} = \lim_{s \to \infty} B(s) = B_{max}, \tag{5}$$

where a higher value of χ_{ic} indicates more efficient interconnects.

GPU Devices and Queries. GPU devices are often characterized by their peak performance, typically measured in gigaflops (GFlop/s), which indicates the number of floating-point operations they can execute per second. This metric is commonly evaluated with matrix multiplication benchmarks and aligns well with high-performance computing (HPC) applications or ML training, involving extensive linear algebra computations. However, data analytics workloads are typically memory-bound rather than compute-intensive. Consequently, gigaflops may not accurately reflect the GPU's performance for these workloads. Similar challenges arise when trying to characterize the query complexity when executed on GPUs. In previous works, such as [13], the queries were characterized by the arithmetic intensity, measured in operations per byte, in order to perform a roofline analysis using a black-box model [51]. While this approach is insightful, it was observed that the arithmetic intensity for most queries tended to be lower compared to typical GPU kernels. This observation suggests that, while arithmetic intensity offers valuable insights, exploring additional or alternative metrics might enhance our understanding of query complexity when executed on a GPU. To provide new insights, in this section we propose a novel approach for characterizing both GPU efficiency and query complexity on GPUs.

Equation (3) indicates that the execution time of operators on the GPU is mainly determined by the slope of the linear curve $a_{q,d}$ for large enough s. However, this parameter captures both the GPU efficiency and the query complexity when executed on the GPU. Our main idea is to decouple this slope parameter $a_{q,d}$ into two parameters, one that captures the inherent query complexity and the other that captures the GPU efficiency.

Definition 4.3. Let $A = [a_{q,d}]_{q \in Q, d \in D}$ be a slope matrix (see Definition 4.1) over queries Q and GPU devices D. Let $\chi_Q = \{\chi_q\}_{q \in Q}$ and $\chi_D = \{\chi_d\}_{d \in D}$ be sets of strictly positive values such that:

$$a_{q,d} = \frac{\chi_q}{\chi_d}, \quad \forall q \in Q, \forall d \in D. \tag{6}$$

Then χ_Q is called the *query characterization set*, and χ_q is called the *characteristic query complexity* of query q. Similarly, χ_D is called the *device characterization set*, and χ_d is called the *characteristic device efficiency* of device d.

Intuitively, higher values of the characteristic query complexity χ_q correspond to more complex queries, and higher values of characteristic device efficiency χ_d indicate a more efficient GPU device. The slope $a_{q,d}$, that determines the runtime, is higher when either the query is more complex (higher χ_q) or the device is less efficient (lower χ_d). Below, we provide sufficient and necessary conditions when such characterizations exist.

THEOREM 4.4. *Let $A = [a_{q,d}]_{q \in Q, d \in D}$ be a slope matrix over queries Q and GPU devices D. The sets of strictly positive numbers $\chi_Q = \{\chi_q\}_{q \in Q}$ and $\chi_D = \{\chi_d\}_{d \in D}$ satisfying Equation (6) exist, if and only if matrix A has rank 1.*

PROOF. We will prove only the ($\longleftarrow$) part of the proof, as the other part follows analogously. Let A be an arbitrary slope matrix (see Definition 4.1) of rank 1. By using the Singular Value Decomposition (SVD), this matrix can be written as: $A = U \Sigma V^*$, where Σ is the diagonal matrix containing the singular values. Since A has rank 1, there is only one non-zero singular value σ in Σ. Then, matrix A can be written as the outer product $A = U \Sigma V^T = \sigma \mathbf{u} \mathbf{v}^T$, where $\mathbf{u}$ is the first column of U, $\mathbf{v}$ is the first column of V and σ is a scalar. This further means that each entry $a_{q,d}$ in A can be written as $a_{q,d} = \sigma u_q v_d$ for the corresponding values $u_q \in \mathbf{u}$ and $v_d \in \mathbf{v}$. Since matrix A has all non-zero elements, each u_q and v_d are non-zero, so we can define $\chi_q = \sigma \cdot u_q$ and $\chi_d = 1/v_d$, which yields:

$$a_{q,d} = \sigma u_q v_d = \frac{\chi_q}{\chi_d},$$

thus satisfying the condition given by Equation (6). □

Table 1: Hardware Configurations for the empirical evaluation.

Configuration	C_1 = PCIe3+A100	C_2 = PCIe5+H100	C_3 = GH200 (NVLink4+H200)	C_4 = PCIe4+RTX3090
CPU	Intel Xeon Platinum 8171M	AMD EPYC 9124	NVIDIA Grace	AMD EPYC 7313
CPU cores	2x6	2x16	72 ARM Neoverse V2 cores	2x16
GPU	NVIDIA A100 40GB	NVIDIA H100 80GB	NVIDIA H200 96GB	NVIDIA RTX3090 24GB
GPU Mem. Bandwidth	1.55 TB/s	4 TB/s	4 TB/s	0.936 TB/s
GPU clock (base–boost)	765–1410 MHz	1080–1785 Mhz	1980–1980 Mhz	1395-2100 MHz
Power cap	400W	400W	624.15W / 900W	350W
Interconnect (IC)	PCIe 3.0	PCIe 5.0	NVLink 4.0	PCIe 4.0
IC Bandwidth (1-way)	16 GB/s	64 GB/s	450 GB/s	32 GB/s

This proof is constructive and provides a method for finding the device and query characterization sets, as given by Definition 4.3.

In more general cases, when the slope matrix A has rank k, we can define a more general characterization, by considering the first k singular values in the SVD decomposition. Concretely, each slope $a_{q,d}$ could be decomposed into:

$$a_{q,d} = \sum_{i=1,\dots,k} \frac{\chi_q^{(i)}}{\chi_d^{(i)}}, \tag{7}$$

where $\chi_q^{(i)}$ is i–th characteristic query complexity and $\chi_d^{(i)}$ is i–the characteristic device efficiency. Intuitively, each $\chi_d^{(i)}$ represents a feature or characteristic of the device and $\chi_q^{(i)}$ represents how sensitive the query is to the corresponding device feature.

4.5 Theoretical Limits

This cost model and the characterizations allow us to investigate the theoretical limits of costs as the data size tends to infinity. This further allows us to compare how the costs behave for different hardware configurations and to find the overhead of data transfers for an arbitrary hardware configuration.

The theoretical limits derived in this section are written in terms of characteristic efficiency and characteristic complexity, assuming the conditions of Theorem 4.4 hold. However, this condition is not necessary. All the formulas in this section can also be derived in terms of the slopes $a_{q,d}$ (see Equation (3)) and the peak bandwidth B_{max} (see Equation (1)).

Hardware Configurations Comparison. Let us assume that we are given two different hardware configurations, C' and C''. The configuration C' consists of an interconnect ic_1 with characteristic efficiency χ_{ic_1} and a GPU device d_1 with characteristic efficiency χ_{d_1} and similarly for C''. We are interested in the speedup that can be achieved when switching from configuration C' to C'' when executing a query q with characteristic complexity χ_q. Let $T'(s) = T'_{q,\mathsf{comm}}(s) + T'_{q,\mathsf{op}}(s)$ be the total time (including data transfers + operators time) for executing the query q on hardware C', and let $T''(s) = T''_{q,\mathsf{comm}}(s) + T''_{q,\mathsf{op}}(s)$ be the total time for executing the same query on hardware C''. We are interested in the speedup $\mathcal{S}_q^{C_1\to C_2} = \lim_{s\to\infty} T''(s)/T'(s)$. Following the cost model and the characterization definitions, we can derive:

$$\mathcal{S}_q^{C_1\to C_2} = \lim_{s\to\infty} \frac{T''_{q,\mathsf{comm}}(s) + T''_{q,\mathsf{op}}(s)}{T'_{q,\mathsf{comm}}(s) + T'_{q,\mathsf{op}}(s)} = \frac{\chi_{ic_2}\chi_{d_2}}{\chi_{ic_1}\chi_{d_1}} \cdot \frac{\chi_{d_1} + \chi_q\chi_{ic_1}}{\chi_{d_2} + \chi_q\chi_{ic_2}} \tag{8}$$

Data Transfers Overhead. In a similar fashion, we can estimate the data transfers overhead. For a given hardware configuration, consisting of an interconnect ic, with efficiency χ_{ic}, a GPU device d, with efficiency χ_d, and a given query q, with characteristic complexity χ_q, we can derive the ratio of time α_q^{comm} spent in data transfers, with respect to the total execution time, as data size tends to infinity. Similarly, as in the previous section, we have:

$$\alpha_q^{\mathsf{comm}} = \lim_{s\to\infty} \frac{T_{q,\mathsf{comm}}(s)}{T_{q,\mathsf{comm}}(s) + T_{q,\mathsf{op}}(s)} = \frac{\chi_d}{\chi_d + \chi_q \cdot \chi_{ic}} \tag{9}$$

In the same way, we can derive the ratio of time spent on the GPU execution, α_q^{op}, as:

$$\alpha_q^{\mathsf{op}} = \lim_{s\to\infty} \frac{T_{q,\mathsf{op}}(s)}{T_{q,\mathsf{comm}}(s) + T_{q,\mathsf{op}}(s)} = \frac{\chi_q \cdot \chi_{ic}}{\chi_d + \chi_q \cdot \chi_{ic}} \tag{10}$$

In practice, we are often interested in the speedup when executing multiple queries from some query set Q over a common dataset of size s. In that case, we have to be careful, as not each query is using the same size of the dataset. Let w_q be the data volume of query q relative to the full dataset size s. In that case, the limit of the ratio of time spent in data transfers α_Q^{comm}, for executing all queries in the query set Q, using the per-query copy policy, is:

$$\alpha_Q^{\mathsf{comm}} = \frac{W \cdot \chi_d}{W \cdot \chi_d + \chi_{ic} \cdot \sum_{q\in Q} w_q\chi_q}, \tag{11}$$

where $W = \sum_{q\in Q} w_q$. In the same way, we can derive the limit of the ratio of time spent on the GPU execution α_Q^{op}, when executing all queries from the query set Q:

$$\alpha_Q^{\mathsf{op}} = \frac{\chi_{ic} \cdot \sum_{q\in Q} w_q\chi_q}{W \cdot \chi_d + \chi_{ic} \cdot \sum_{q\in Q} w_q\chi_q} \tag{12}$$

In the previous derivations, we assumed a per-query CPU↔GPU copy policy, meaning that the required data is copied for each query. However, the equations for the per-dataset copy policy can be derived in the same manner. As previously mentioned, all the formulas in this section can be written in terms of the slopes $a_{q,d}$ (see Equation (3)) and the peak bandwidth B_{max} (see Equation (1)), in case the conditions of Theorem 4.4 do not hold.

5 EXPERIMENTAL ANALYSIS

Using MaxBench, together with the cost model and the characterizations described above, we have analyzed a number of relevant combinations of GPUs and interconnects.

Table 2: Runtime breakdown per query for each benchmark and hardware configuration. The slowdown w.r.t. to the fastest configuration (C3) is shown in brackets. OOM denotes out-of-memory errors.

TPC-H BENCHMARK, SF=10 (≈12 GB, CSV)

	Total Time [ms]				Compute Time [ms]				Communication Time [ms]			
	C1	C2	C3	C4	A100	H100	H200	RTX3090	PCIe3	PCIe5	NVLink4	PCIe4
Q1	704 (9.53x)	202 (2.74x)	74 (1x)	268 (3.63x)	122 (1.81x)	74 (1.11x)	67 (1x)	137 (2.04x)	582 (86.50x)	128 (18.97x)	7 (1x)	131 (19.48x)
Q2	77 (10.29x)	22 (2.96x)	7 (1x)	46 (6.18x)	12 (1.86x)	7 (1.10x)	6 (1x)	22 (3.47x)	65 (59.33x)	15 (13.81x)	1 (1x)	24 (21.94x)
Q3	455 (30.10x)	108 (7.18x)	15 (1x)	137 (9.08x)	19 (1.89x)	11 (1.08x)	10 (1x)	19 (1.92x)	436 (82.43x)	98 (18.48x)	5 (1x)	118 (22.34x)
Q4	298 (25.05x)	63 (5.28x)	12 (1x)	88 (7.44x)	17 (2.03x)	9 (1.09x)	9 (1x)	17 (1.99x)	280 (85.59x)	53 (16.31x)	3 (1x)	71 (21.78x)
Q5	516 (20.26x)	128 (5.03x)	25 (1x)	154 (6.05x)	43 (2.18x)	22 (1.10x)	20 (1x)	39 (1.97x)	473 (82.34x)	106 (18.55x)	6 (1x)	115 (20.05x)
Q6	364 (38.48x)	86 (9.04x)	9 (1x)	89 (9.40x)	10 (1.82x)	6 (1.09x)	5 (1x)	9 (1.66x)	355 (87.32x)	80 (19.64x)	4 (1x)	80 (19.71x)
Q7	534 (30.06x)	125 (7.01x)	18 (1x)	147 (8.27x)	24 (2.01x)	12 (1.05x)	12 (1x)	25 (2.11x)	511 (84.63x)	112 (18.61x)	6 (1x)	122 (20.24x)
Q8	614 (28.50x)	148 (6.86x)	22 (1x)	188 (8.74x)	30 (2.05x)	16 (1.09x)	15 (1x)	31 (2.12x)	584 (84.96x)	132 (19.18x)	7 (1x)	157 (22.87x)
Q9	765 (21.94x)	185 (5.32x)	35 (1x)	229 (6.58x)	58 (2.17x)	29 (1.09x)	27 (1x)	57 (2.15x)	707 (86.41x)	156 (19.09x)	8 (1x)	172 (21.04x)
Q10	496 (34.77x)	115 (8.08x)	14 (1x)	218 (15.27x)	16 (1.92x)	9 (1.08x)	9 (1x)	20 (2.41x)	479 (83.51x)	106 (18.47x)	6 (1x)	197 (34.35x)
Q11	295 (18.10x)	73 (4.51x)	16 (1x)	91 (5.61x)	23 (1.74x)	14 (1.08x)	13 (1x)	27 (2.11x)	272 (82.85x)	59 (18.10x)	3 (1x)	64 (19.50x)
Q12	2195 (56.26x)	495 (12.70x)	39 (1x)	606 (15.53x)	24 (1.76x)	15 (1.08x)	14 (1x)	28 (2.05x)	2171 (85.81x)	481 (19.00x)	25 (1x)	578 (22.85x)
Q13	644 (5.40x)	223 (1.87x)	119 (1x)	635 (5.32x)	208 (1.83x)	126 (1.10x)	114 (1x)	495 (4.34x)	436 (83.48x)	97 (18.55x)	5 (1x)	139 (26.70x)
Q14	1858 (61.32x)	416 (13.72x)	30 (1x)	485 (16.02x)	17 (1.83x)	10 (1.07x)	9 (1x)	21 (2.27x)	1841 (86.85x)	406 (19.14x)	21 (1x)	464 (21.91x)
Q15	1797 (62.15x)	405 (14.02x)	29 (1x)	465 (16.10x)	15 (1.74x)	9 (1.05x)	8 (1x)	18 (2.19x)	1782 (86.73x)	397 (19.30x)	21 (1x)	447 (21.75x)
Q16	388 (12.00x)	105 (3.24x)	32 (1x)	246 (7.60x)	60 (2.12x)	31 (1.10x)	28 (1x)	168 (5.95x)	329 (78.47x)	74 (17.63x)	4 (1x)	78 (18.70x)
Q17	1864 (51.29x)	433 (11.91x)	36 (1x)	526 (14.48x)	27 (1.85x)	16 (1.09x)	14 (1x)	37 (2.53x)	1838 (84.02x)	417 (19.07x)	22 (1x)	490 (22.40x)
Q18	2240 (63.23x)	499 (14.08x)	35 (1x)	564 (15.93x)	17 (1.90x)	10 (1.09x)	9 (1x)	22 (2.39x)	2222 (84.75x)	489 (18.64x)	26 (1x)	542 (20.69x)
Q19	2089 (13.79x)	557 (3.68x)	151 (1x)	OOM	249 (1.92x)	143 (1.10x)	130 (1x)	OOM	1840 (86.02x)	414 (19.37x)	21 (1x)	OOM
Q20	2141 (53.05x)	494 (12.23x)	40 (1x)	750 (18.57x)	30 (1.89x)	17 (1.09x)	16 (1x)	40 (2.54x)	2111 (86.21x)	476 (19.45x)	24 (1x)	709 (28.96x)
Q21	OOM	761 (2.55x)	299 (1x)	OOM	OOM	298 (1.09x)	273 (1x)	OOM	OOM	464 (18.24x)	25 (1x)	OOM
Q22	448 (37.92x)	105 (8.86x)	12 (1x)	156 (13.20x)	12 (1.95x)	7 (1.09x)	6 (1x)	23 (3.73x)	436 (78.88x)	98 (17.71x)	6 (1x)	132 (23.97x)

H2O-G BENCHMARK, G1_1e8_1e2_5_0 (≈4.9 GB, CSV)

	Total Time [ms]				Compute Time [ms]				Communication Time [ms]			
	C1	C2	C3	C4	A100	H100	H200	RTX3090	PCIe3	PCIe5	NVLink4	PCIe4
Q1	284 (24.88x)	66 (5.81x)	11 (1x)	100 (8.77x)	14 (1.75x)	6 (0.75x)	8 (1x)	18 (2.28x)	269 (82.12x)	60 (18.33x)	3 (1x)	81 (24.85x)
Q2	481 (30.26x)	113 (7.08x)	16 (1x)	138 (8.70x)	23 (2.20x)	11 (1.07x)	11 (1x)	25 (2.40x)	458 (86.18x)	101 (19.08x)	5 (1x)	113 (21.26x)
Q3	660 (14.85x)	168 (3.79x)	44 (1x)	408 (9.19x)	73 (1.95x)	38 (1.03x)	37 (1x)	168 (4.50x)	587 (82.08x)	130 (18.15x)	7 (1x)	241 (33.64x)
Q4	463 (18.11x)	112 (4.36x)	26 (1x)	172 (6.73x)	33 (1.61x)	16 (0.80x)	20 (1x)	29 (1.43x)	430 (83.46x)	95 (18.44x)	5 (1x)	143 (27.71x)
Q5	484 (20.61x)	116 (4.92x)	23 (1x)	265 (11.29x)	49 (2.71x)	19 (1.04x)	18 (1x)	137 (7.48x)	435 (83.23x)	97 (18.50x)	5 (1x)	129 (24.62x)
Q6	1262 (3.41x)	482 (1.30x)	369 (1x)	2140 (5.79x)	915 (2.51x)	405 (1.11x)	365 (1x)	2061 (5.64x)	346 (83.83x)	77 (18.57x)	4 (1x)	79 (19.07x)
Q7	561 (15.59x)	141 (3.91x)	36 (1x)	368 (10.23x)	59 (1.94x)	31 (1.04x)	30 (1x)	145 (4.82x)	503 (85.93x)	109 (18.72x)	6 (1x)	223 (38.07x)
Q9	531 (11.89x)	140 (3.13x)	45 (1x)	257 (5.76x)	85 (2.17x)	43 (1.08x)	39 (1x)	101 (2.57x)	446 (84.42x)	97 (18.41x)	5 (1x)	156 (29.51x)
Q10	2045 (8.48x)	626 (2.60x)	241 (1x)	1774 (7.36x)	460 (2.33x)	215 (1.09x)	197 (1x)	940 (4.77x)	1586 (35.91x)	412 (9.33x)	44 (1x)	835 (18.90x)

CLICKBENCH BENCHMARK, ATHENA (≈14.8 GB, CSV)

	Total Time [ms]				Compute Time [ms]				Communication Time [ms]			
	C1	C2	C3	C4	A100	H100	H200	RTX3090	PCIe3	PCIe5	NVLink4	PCIe4
Q4	42 (6.48x)	14 (2.09x)	7 (1x)	27 (4.15x)	8 (1.35x)	6 (0.98x)	6 (1x)	14 (2.32x)	34 (84.60x)	8 (19.01x)	0.40 (1x)	13 (31.91x)
Q7	49 (4.23x)	19 (1.61x)	12 (1x)	21 (1.78x)	16 (1.37x)	11 (1.00x)	11 (1x)	10 (0.86x)	34 (82.95x)	8 (18.43x)	0.41 (1x)	11 (27.25x)
Q9	113 (4.66x)	37 (1.51x)	24 (1x)	177 (7.26x)	63 (2.64x)	26 (1.08x)	24 (1x)	160 (6.73x)	51 (83.93x)	11 (18.63x)	0.60 (1x)	17 (27.96x)
Q10	132 (5.24x)	41 (1.64x)	25 (1x)	182 (7.22x)	65 (2.64x)	26 (1.08x)	24 (1x)	163 (6.68x)	68 (82.65x)	15 (18.58x)	0.82 (1x)	19 (23.11x)
Q11	60 (12.87x)	16 (3.37x)	5 (1x)	34 (7.29x)	8 (2.11x)	4 (1.07x)	4 (1x)	16 (4.00x)	51 (78.79x)	11 (17.45x)	0.65 (1x)	18 (27.46x)
Q12	69 (12.80x)	18 (3.37x)	5 (1x)	35 (6.46x)	9 (1.99x)	5 (1.09x)	5 (1x)	15 (3.30x)	60 (79.61x)	13 (17.51x)	0.75 (1x)	20 (25.96x)
Q13	97 (21.25x)	23 (5.03x)	5 (1x)	50 (10.96x)	6 (2.20x)	3 (1.05x)	3 (1x)	7 (2.34x)	91 (54.44x)	20 (11.95x)	2 (1x)	43 (25.98x)
Q14	185 (7.38x)	45 (1.80x)	25 (1x)	209 (8.32x)	95 (4.05x)	25 (1.08x)	23 (1x)	156 (6.66x)	91 (53.15x)	20 (11.63x)	2 (1x)	53 (31.19x)
Q15	106 (21.46x)	27 (5.54x)	5 (1x)	45 (9.18x)	7 (2.17x)	3 (1.05x)	3 (1x)	7 (2.34x)	100 (54.79x)	24 (13.29x)	2 (1x)	38 (20.99x)
Q16	56 (13.58x)	14 (3.38x)	4 (1x)	38 (9.32x)	7 (2.22x)	3 (1.00x)	3 (1x)	16 (5.18x)	49 (46.55x)	11 (10.31x)	1 (1x)	23 (21.36x)
Q17	124 (13.71x)	35 (3.85x)	9 (1x)	88 (9.75x)	12 (2.00x)	7 (1.05x)	6 (1x)	23 (3.64x)	111 (39.90x)	28 (10.10x)	3 (1x)	65 (23.41x)
Q22	1311 (2.53x)	672 (1.30x)	519 (1x)	1404 (2.71x)	865 (1.68x)	575 (1.12x)	513 (1x)	1254 (2.44x)	446 (85.31x)	97 (18.57x)	5 (1x)	150 (28.72x)
Q23	1214 (6.30x)	397 (2.06x)	193 (1x)	871 (4.52x)	318 (1.74x)	204 (1.12x)	182 (1x)	652 (3.58x)	897 (85.06x)	193 (18.34x)	11 (1x)	219 (20.73x)
Q24	1341 (2.59x)	672 (1.30x)	518 (1x)	1363 (2.63x)	875 (1.71x)	571 (1.11x)	513 (1x)	1254 (2.45x)	466 (85.96x)	101 (18.55x)	5 (1x)	109 (20.09x)
Q25	99 (23.52x)	26 (6.24x)	4 (1x)	51 (12.13x)	5 (2.09x)	2 (1.08x)	2 (1x)	7 (3.06x)	94 (48.11x)	24 (12.16x)	2 (1x)	44 (22.55x)
Q26	148 (7.07x)	37 (1.78x)	21 (1x)	187 (8.89x)	88 (4.54x)	21 (1.09x)	19 (1x)	141 (7.25x)	60 (39.42x)	16 (10.65x)	2 (1x)	45 (29.86x)
Q27	102 (17.94x)	28 (4.85x)	6 (1x)	57 (9.95x)	8 (2.07x)	4 (1.07x)	4 (1x)	14 (3.66x)	94 (48.82x)	24 (12.23x)	2 (1x)	43 (22.20x)
Q31	102 (18.18x)	26 (4.68x)	6 (1x)	44 (7.90x)	9 (2.02x)	5 (1.07x)	4 (1x)	13 (3.13x)	93 (67.78x)	22 (15.75x)	1 (1x)	31 (22.55x)
Q32	132 (19.22x)	34 (4.93x)	7 (1x)	66 (9.64x)	10 (2.03x)	5 (1.07x)	5 (1x)	18 (3.58x)	122 (64.71x)	28 (15.13x)	2 (1x)	48 (25.65x)
Q33	182 (6.00x)	65 (2.15x)	30 (1x)	247 (8.13x)	66 (2.58x)	28 (1.08x)	26 (1x)	150 (5.87x)	116 (24.36x)	38 (7.92x)	5 (1x)	97 (20.28x)
Q34	578 (23.61x)	137 (5.61x)	24 (1x)	248 (10.14x)	30 (2.18x)	15 (1.07x)	14 (1x)	30 (2.18x)	548 (51.29x)	122 (11.47x)	11 (1x)	218 (20.42x)
Q35	584 (23.93x)	137 (5.61x)	24 (1x)	267 (10.96x)	31 (2.23x)	15 (1.08x)	14 (1x)	30 (2.17x)	553 (52.17x)	122 (11.51x)	11 (1x)	237 (22.40x)
Q36	25 (8.88x)	8 (2.74x)	3 (1x)	24 (8.72x)	5 (2.16x)	2 (0.99x)	2 (1x)	14 (5.70x)	20 (45.62x)	5 (12.29x)	0.43 (1x)	11 (25.22x)

5.1 Experimental Setup

Hardware. We performed the experiments on 4 different configurations, including different interconnects and NVIDIA GPUs: C_1 = PCIe3 + A100, C_2 = PCIe5 + H100, C_3 = GH200 (NVLink4 + H200) and C_4 = PCIe4 + RTX3090 (Table 1). The GPUs in configurations C_2 and C_3 are often both denoted as H100. However, some of their characteristics are slightly different, such as the GPU clock rate or the power cap. We denote the GPU in configuration C_3 with H200 to differentiate it from the H100 in C_2.

Software. We used Maximus `v0.2`, compiled with Apache Arrow `v17.0.0` [46] for CPU processing, cuDF `v24.08.03` [3] for GPU processing, and Caliper `v2.12.1` [7] for code instrumentation. The libraries were compiled using the GNU GCC compiler `v13.3` and the CUDA `v12.5` (driver `v550.54.15`).

Workloads. The experiments include all supported queries from the TPC-H, H2O-G (G1_1e8_1e2_5_0 dataset) and ClickBench (Athena) benchmarks. This is a total of 54 queries: 22 TPC-H queries, 9/10 H2O-G queries and 23/43 ClickBench queries (Table 2). The omitted queries involve some of the unsupported expressions like

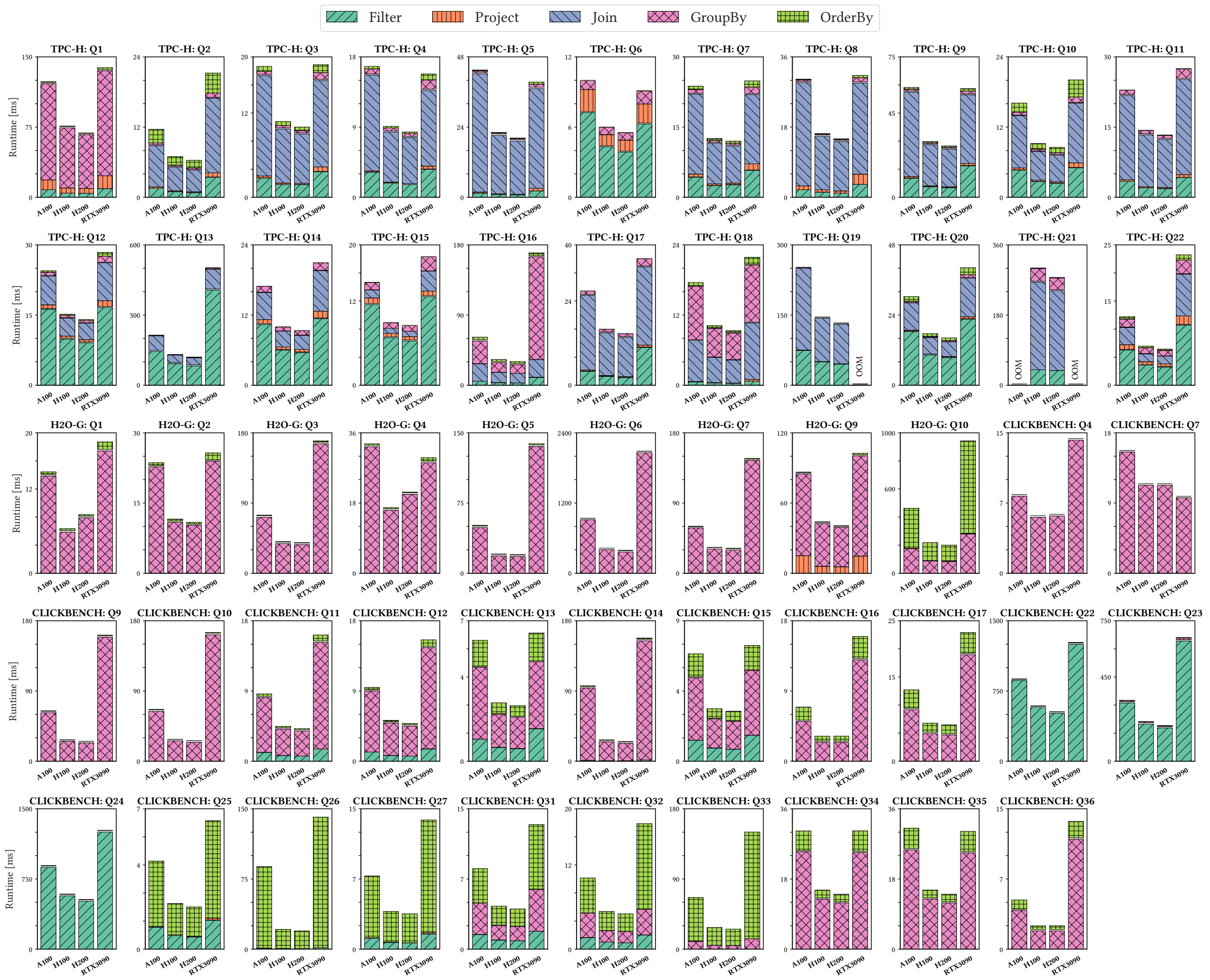
Figure 2: The breakdown of the GPU execution, when the data resides on the GPU.

PARTITION-BY or unsupported data types, which is why they are not included. Each query is repeated 10 times and the best time is reported to provide a lower bound on performance.

5.2 Data Transfers vs. Operators Time

The times spent on data transfers and on GPU query execution are shown in Table 2, including the following benchmarks:TPC-H with scaling factor 10 (≈12GB dataset), H2O-G (G1_1e8_1e2_5_0 ≈4.7GB dataset) and ClickBench Athena (≈14GB dataset), on hardware configurations C_1, C_2, C_3 and C_4. For each query, the data starts and ends on a CPU, using the per-query copy policy. The results contain the breakdown of the total runtime on data transfers and operators time. The difference between the total time and the sum of the compute and communication time represents the scheduling and execution overheads (less than 1% of the runtime). The observed variability in total time over repetitions (omitting the warm-up run), measured as a weighted average coefficient of variation (CV), averaged over all queries from all workloads is: 3.48% on C1, 24.2% on C2, 4.98% on C3 and 23.09% on C4.

Data Transfer Overhead with PCIe. For PCIe configurations , C_1, C_2, C_3 and C_4, the total runtime is dominated by CPU↔GPU data transfers. In the TPC-H benchmark, on C_1, data transfers take from 67% of the total runtime (Q13) up to more than 98% of the total runtime (Q12, Q14, Q15, Q18, Q20), and even staying out of the GPU memory for Q21. On C_2, data transfers take from 33% of runtime (Q13), but still go up to 97% (Q15, Q18). The data transfer time on C_2 (PCIe 5.0) is faster than the data transfers on C_1 (PCIe 3.0) by factors ranging from 3.83x to 4.56x. This roughly corresponds to the ratio of the peak bandwidths of these interconnects. Similarly, in the H2O-G benchmark, in 8 out of 9 queries, data transfers take from 77% (Q10) up to 95% (Q2) of the total runtime on C_1, while taking from 65% to 90% of the total runtime on C_2. In the ClickBench benchmark, in 18 out of 23 queries, data transfers take from 51% (Q10) – 95% (Q25) on C_1. On C_2, data transfers are dominant in 15 out of 23 queries, taking from 55% (Q4) – 90% (Q25).

Insight 1: For the majority of queries, data transfers dominate the total runtime with PCIe. Extrapolating, for many

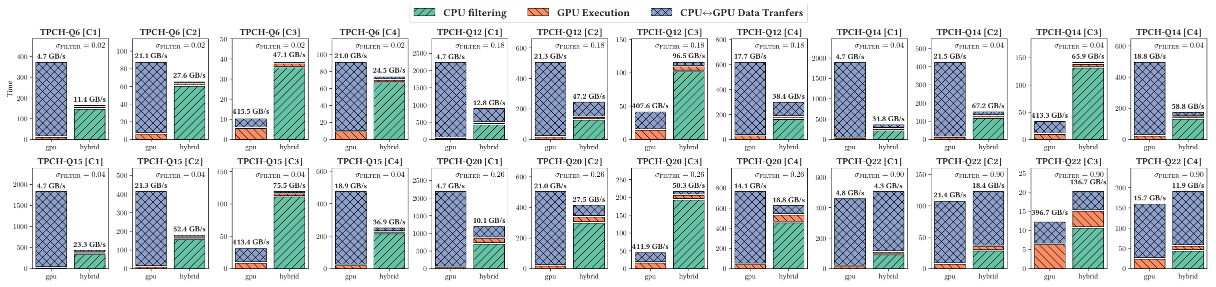

Figure 3: The actual and perceived bandwidths when the CPU-side filtering is employed before offloading the data to GPU.

queries (e.g., transfer time is > 90%), PCIe 7.0 (doubling the unidirectional bandwidth) will still be a major factor in the execution time, especially for less powerful GPUs.

Data Transfer Overhead with NVLink. For configuration C_3 (NVLink4), for most queries, the time spent executing the operators on the GPU is larger than the time spent in data transfers. In the TPC-H benchmark, in configuration C_3, the communication takes as low as 4.38% of the total runtime for query Q13, 8-15% for Q1, Q2, Q16, Q19, Q21, 15-30% for Q4, Q5, Q9, Q11, 30-50% for Q3, Q6, Q7, Q8, Q10, Q22 and goes up to 73% for Q18, which has the largest data volume. With NVLink 4.0 and H200, there are only 6 out of 22 queries, in which the data transfers take more than 50% of the total runtime. In the other 16 TPC-H queries, the data transfers take on average 24% of the total runtime. Similarly, in the H2O-G benchmark, in configuration C_3, the data transfers take as low as 1% (Q6) – 33% (Q2) of the total runtime. In the ClickBench queries, the data transfers take from 1% (Q24) – 45% (Q25) of the total runtime.

Insight 2: With high-bandwidth interconnects, we transition from being interconnect-bound to most queries being GPU-bound, changing the optimizations needed to improve performance.

Comparing GPU Execution Times. The time spent in the execution of the operators on the GPU varies significantly for different GPU types. The H200 GPU was faster than the A100 by 1.74–2.18x (TPC-H), 1.6–2.7x (H2O-G) and 1.34–4.54x (ClickBench). The results indicate that the GPU model can have an increasing impact on performance with faster interconnects.

Insight 3: The GPU efficiency will impact performance when used in conjunction with fast interconnects boosting data transfers by a factor of 80x or more.

The Breakdown of Operators. To understand how the GPU execution time is distributed, we used the breakdown of operator's profiling. Observe that some of the benchmarked queries also involve the distinct and the limit operators, but their runtimes were negligible, which is why they are omitted (Figure 2). On A100, the TPC-H Q21 was out of memory, as marked in the figure.

The results indicate that the join operator is a bottleneck in more than a half of the TPC-H queries (Q2, Q3, Q4, Q5, Q7, Q8, Q9, Q10, Q11, Q17, Q19, Q21). The group-by is a bottleneck in some TPC-H queries (Q1, Q16, and Q18), 8 out of 9 H2O-G queries (Q1–Q9) and 14 out of 23 ClickBench queries (Q4 – Q17, Q34–Q36). The order-by was a bottleneck in 1 out of 9 H2O-G queries and 6 out of 23 ClickBench queries. Surprisingly, in a substantial number of queries, the filter operator is a bottleneck (TPC-H: Q6, Q12, Q13, Q14, Q15, Q20, Q22, ClickBench: Q22–Q24). These filters often involve expressions with string comparisons or regex matching. For example, TPC-H Q12 involves comparing the `o_orderpriority` attribute to string literals `1-URGENT` or `2-HIGH` within an `if_else` expression, in addition to checking `l_shipmode in ('MAIL', 'SHIP')`. Similarly, TPC-H Q13 involves string regex matching in order to check `o_comment not like '%special%requests%'`, whereas TPC-H Q14 involves checking `p_type like 'PROMO%'`. Computing these expressions on GPUs is challenging as they might involve irregular memory access patterns, which are not coalesced. This is especially problematic when combined with branching expressions, such as the `if_else` expression, as found in TPC-H Q12.

Insight 4: Although joins are the most studied operator in GPUs, they are not the only bottleneck.

Insight 5: Efficient filtering is challenging on GPUs if string comparisons or regex matching are involved, especially when used within branching expressions such as the `if_else` expression. These expressions involve irregular memory access patterns, which are not necessarily coalesced.

5.3 Optimizations: Selective Data Transfers

In this section, we assess the impact of executing selection predicates on the CPU, which can potentially reduce the data transferred as well as the GPU execution time. We ran the queries where filtering was a bottleneck, using the per-query copy policy (Figure 3 showing the actual and the perceived bandwidth, as well as the the filter selectivity (σ)). The results indicate that high selectivity queries, such as Q6 ($\sigma = 0.02$), on slow interconnects such as PCIe 3.0 (C1), the CPU filtering can improve the performance more than 2x, by increasing the perceived bandwidth from 4.7GB/s to 11.4GB/s. On fast interconnects, such as NVLink 4.0 (C3), performance degrades by more than 2x. For low selectivity queries, such as Q22 ($\sigma = 0.9$), CPU filtering degrades performance on all configurations.

Insight 6: For high selectivity predicates, selective data transfers improve the performance on slow interconnects,

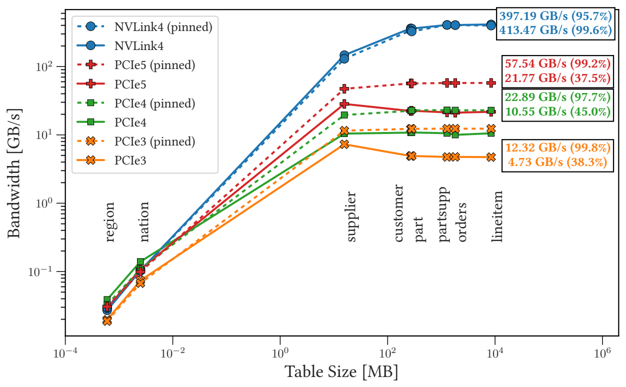

Figure 4: The achieved CPU↔GPU data transfers bandwidth for different TPC-H tables (SF=10) when tables are stored in pinned and non-pinned CPU memory regions.

but degrade performance on fast interconnects. For low selectivity predicates, selective data transfers degrade performance regardless of interconnect.

5.4 Cost Modeling & Characterization

Data Transfers. To measure the interconnect bandwidth, we transferred the TPC-H tables (SF = 10) from the CPU to the GPU, enabling the breakdown of the data transfers profiling level in MaxBench. The TPC-H tables cover a wide spectrum of sizes, ranging from 4KB (the *region* table) up to 8.445 GB (the *lineitem* table). Moreover, table schemas involve various types, including `int32`, `int64`, `float64`, `date32` and `string`. The copy is performed using an asynchronous `cudaMemCpyAsync`, one for each column. For variable-length data types, such as strings, an additional offset array is copied. Additionally, when dealing with nullable columns, a separate null mask array must be transferred.

We considered two different scenarios: one where all the tables are initially stored in a non-pinned CPU memory region and the other one where the tables are stored in a pinned memory. We performed this benchmark on each of the four different hardware configurations: C_1 , C_2, C_3, and C_4. The achieved bandwidth was compared with the bandwidth measured in isolation, using the NVIDIA nvbandwidth benchmark [34] (Figure 4).

The maximum achieved bandwidth with PCIe 3.0 was 12.32 GB/s (out of theoretical 16 GB/s), with PCIe 4.0 was 22.89 GB/s (out of theoretical 32GB/s, with PCIe 5.0 was 57.52 GB/s (out of theoretical 64 GB/s), and with NVLink 4.0 was 413.45 GB/s (out of theoretical 450 GB/s). The achieved maximum bandwidth was more than 97% of the peak bandwidth achieved in isolation on all configurations.

Observe that the expected peak bandwidths on configurations C_1, C_2, C_3 and C_4, with PCIe interconnects, are only achievable when the data is stored in a page-locked (pinned) memory region on the CPU. This is expected because Direct Memory Access (DMA) requires the data to be page-locked before the transfer to the GPU can start. If the memory region is not pinned, an additional step is required where the data is first copied to an intermediary pinned buffer before transferring it to the GPU. This effect may be minimal in database engines where the buffer cache is typically pinned but is an important design consideration (as the private memory of query threads with intermediate results might not be pinned). The configuration C_3 with GH200 features an NVLink-C2C interconnect that enables direct, high-bandwidth, and cache-coherent memory access between the Grace CPU and the Hopper GPU. In this configuration, the GPU can directly access system memory without the memory being pinned. For this reason, there was no significant performance difference between pinned and non-pinned memory.

Insight 7: On configurations with PCIe, the peak bandwidth can only be achieved with pinned memory, which drops to less than 40% of the peak bandwidth without pinned memory. On NVLink configurations, memory does not need to be pinned. This is an important insight for the database architectures needed for CPU-GPUs data processing.

Another important observation is that faster interconnects require large data sizes to achieve the peak bandwidth. PCIe 3.0 requires 0.43 MB size to reach 90% of the peak bandwidth, PCIe 4.0 requires 0.62 MB size to reach its peak, PCIe 5.0 requires 1.2 MB to reach 90% of its peak, whereas NVLink 4.0 requires 280 MB to reach 90% peak bandwidth.

Insight 8: NVLink interconnects offer more bandwidth than PCIe for any data size, but require two orders of magnitude larger data sizes to reach the peak bandwidth.

For modeling purposes, we derived an individual curve for each interconnect using the formula $B(s) = B_{max} \cdot s/(s+s_0)$, as specified in Equation (1), to fit the experimental data. We then used the B_{max} value to calculate the characteristic interconnect efficiencies χ_{PCIe3}, χ_{PCIe4}, χ_{PCIe5} and $\chi_{NVLink4}$, for each hardware configuration.

Operators Time. To model the operators time, we ran the TPC-H, H2O-G and ClickBench benchmarks for a variety of dataset sizes. The TPC-H benchmark was run for scaling factors ranging from 1 to 30, which corresponds to the dataset sizes ranging from 1 GB to 30 GB on all configurations. Since the H2O-G and ClickBench benchmarks have a fixed data size, we sampled datasets of different sizes, uniformly at random, from their full datasets. For H2O-G benchmark the sample sizes varied from 10%, 20%,…,100% of the full dataset size (4.9GB). For ClickBench, the sample sizes varied from 10%, 12%, 14%, …, 30% of the full dataset size (74GB), due to the GPU memory constraints. Due to memory constraints, not all GPUs could execute all queries. For example, for TPC-H scaling factor 30, even H200 was not able to execute queries Q12-Q22. Figure 5 shows the time spent in executing the operators on each GPU with respect to the input data size of each query. The points where the system ran out of memory are omitted.

The results indicate a linear relationship between GPU execution time and data volume across all benchmarked GPUs. This linearity is not surprising. Due to the massive parallelism offered by thousands of CUDA cores, combined with the limited compute intensity inherent in these workloads, execution becomes constrained by how quickly data can be transferred between memory and compute units rather than by computational complexity. Consequently, these workloads are memory bandwidth-bound, causing the runtime to be proportional to the ratio of data size to GPU memory bandwidth,

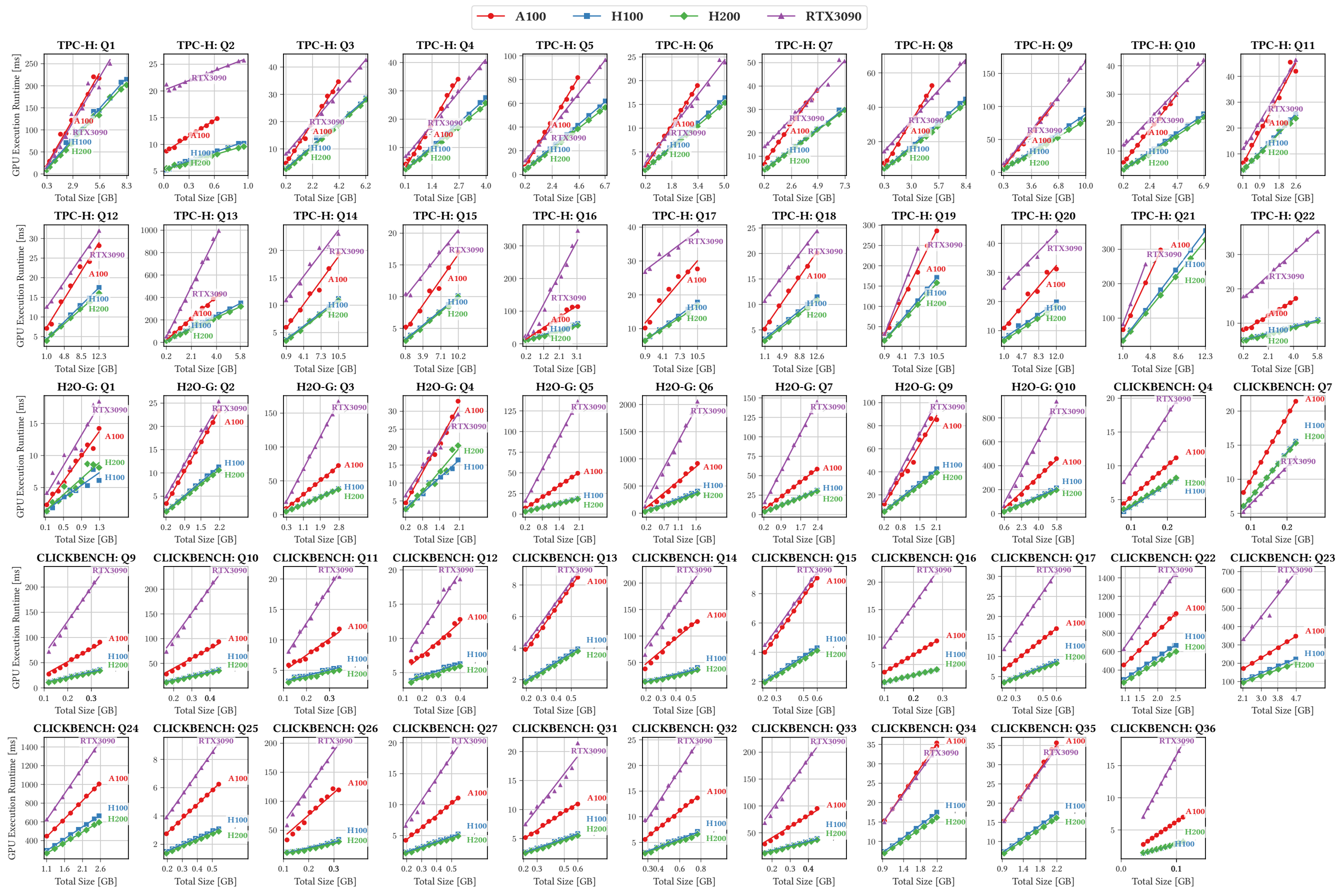
Figure 5: GPU execution time [ms] across various GPU types relative to the input data size [GB].

which is a linear function of data size. This justifies our assumption of linearity when modeling the operator cost, as defined by Eq. (3).

Insight 9: For all benchmarked TPC-H, H2O-G and ClickBench queries, the GPU execution time of each query scales linearly with respect to the input data size.

Figure 5 shows that RTX3090 was sometimes faster than the A100 GPU (see, e.g., TPC-H Q4), generally considered more powerful. This is caused by an architectural difference between these two GPUs: the L2 cache of A100 is split into two parts, as opposed to the single unit found in RTX3090. This can result in a NUMA effect in the L2 cache of A100. This can further lead to a higher cache miss rate on A100 [35]. This explains why the hash join was sometimes faster on RTX3090 (Figure 2) than on A100.

For modeling purposes, we derived a separate linear curve for each query and each GPU type, as given by Equation (3), to fit the experimental data. We then constructed a slope matrix A, as given by Definition 4.1 and performed an SVD decomposition of this matrix, as described in Theorem 4.4, to get characteristic efficiencies $\chi_{RTX3090}$, χ_{A100}, χ_{H100} and χ_{H200}, for each GPU device and characteristic query complexity for each query.

Cost Model Evaluation. To evaluate the cost model (Section 4), we compared data transfers time and operators time predicted by the model, using Equations (2) and (3), with the empirically measured times for TPC-H (SF=10), H2O-G (4.9GB dataset size) and ClickBench (14.8GB Athena dataset size) queries. We evaluated different configurations: (1) when the query inputs and outputs are on the CPU and the per-query policy is used; (2) when the query inputs and outputs are on the CPU and the per-dataset policy is used; (3) when the query inputs are already preloaded in a GPU memory and the outputs are kept on the GPU. The results are shown in Figure 6. The results indicate that the cost model is able to estimate both the data transfers and the operators time with less than 10% error in most cases. The only case where the error was larger, was estimating the communication cost with the PCIe4 interconnect on TPC-H queries. The interconnect bandwidth on this machine varied between 20 and 25GB/s, resulting in less predictable behavior.

Insight 10: The cost model presented in Section 4 is able to accurately predict the performance of the supported TPC-H, H2O-G and ClickBench queries on given hardware.

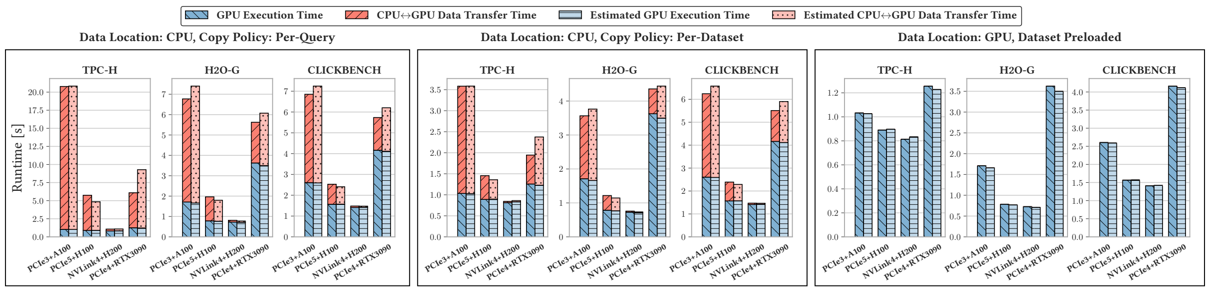

Figure 6: Comparison of the empirical performance data vs. the performance estimated by the cost model.

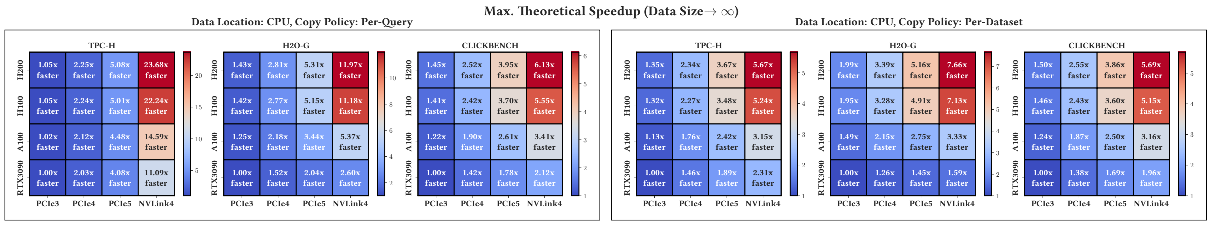

Figure 7: Predicted speedups for different hardware configurations, w.r.t. the slowest configuration (PCIe3 + RTX3090).

5.5 Powerful GPUs vs. Fast Interconnects

To assess how different types of GPUs and interconnects affect the end-to-end performance, we used the cost models derived in the previous section to estimate the performance for each hardware, workloads and copy-policy combination: {PCIe 3.0, PCIe 4.0, PCIe 5.0, NVLink 4.0} × {RTX3090, A100, H100, H200} × {TPC-H, H2O-G, ClickBench} × {per-query, per-dataset}. Within each benchmark, the queries are assumed to be executed one after the other. We used the theoretical limits from Section 4.5 to compare different hardware configurations at limits, i.e., when the data sizes approach infinity. Figure 7 shows the speedups achieved, compared to the slowest configuration (PCIe 3.0 + RTX3090).

The results indicate that, with slow interconnects, such as PCIe 3.0, switching from the slowest (RTX3090) to the fastest (H200) GPU would improve the performance from 1.05x (TPC-H) to 1.45x (H2O-G) with the per-query copy policy and from 1.35x (TPC-H) to 1.99x (H2O-G) with the per-dataset copy policy. With fast interconnects, such as NVLink 4.0, switching from the slowest to the fastest GPU would improve the performance from 2.13x (TPC-H) to 4.6x (H2O-G) with the per-query copy policy and from 2.45x (TPC-H) to 4.82x (H2O-G) with the per-dataset copy policy. This is expected, as the runtime is dominated by data transfers on slow interconnects.

Insight 11: With slow interconnects, the GPU compute power has less impact on performance than with fast interconnects. For example, switching from the slowest to the fastest GPU, would improve the performance by at most 1.99x (H2O-G) with PCIe 3.0. The same change, with NVLink 4.0, would improve the performance 4.82x.

The reason why H2O-G benchmark benefits more from a faster GPU than, e.g., TPC-H, is because the ratio of the time spent in operators execution vs. the time spent in data transfers is higher for H2O-G than for TPC-H. The same trend can also be observed when changing the copy policy: switching from a per-query to per-dataset copy policy generally increases the impact a faster GPU has on the performance. This is expected, since using the per-dataset copy policy usually reduces the data transfer costs.

Insight 12: Improving the GPU compute power has more impact when the ratio of computation vs. data transfers is high, such as in H2O-G. Improving the interconnect bandwidth has more impact when the ratio is lower, such as TPC-H.

5.6 Future Trends

In the previous analysis, we have only focused on interconnects and GPUs that have been evaluated. However, the cost model is not limited to these and can be used to predict the performance for configurations with arbitrary efficiency. To explore how the interconnect bandwidth (= interconnect efficiency χ_{ic}) and the GPU device efficiency (χ_d) influence the trade-offs between data transfer and GPU execution costs, we focus on two scenarios: *(i)* the interconnect bandwidth is varied beyond current capabilities, while keeping the GPU efficiency fixed, and *(ii)* the GPU efficiency is varied beyond current capabilities, while keeping the interconnect bandwidth fixed. In both of these scenarios, we focus on executing all supported queries from TPC-H, H2O-G and ClickBench benchmarks, using different copy policies, where the data size is assumed to tend to infinity. We use the theoretical limits from Section 4.5 and, in particular, Equations (11) - (12).

Varying the Interconnect Efficiency. Here, we vary the interconnect bandwidth from 0 to 1000 GB/s (unidirectional) while keeping the GPU efficiency at the values of RTX3090, A100, H100, and H200 GPUs. The upper limit of 1000 GB/s is more than 2x faster than currently possible with NVLink 4.0. Figure 8 shows the

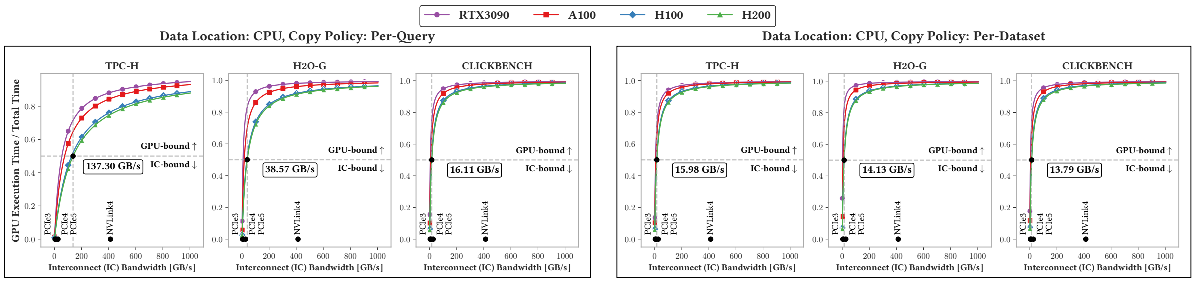

Figure 8: The ratio of the GPU query execution time vs. the total execution time for different interconnect bandwidths.

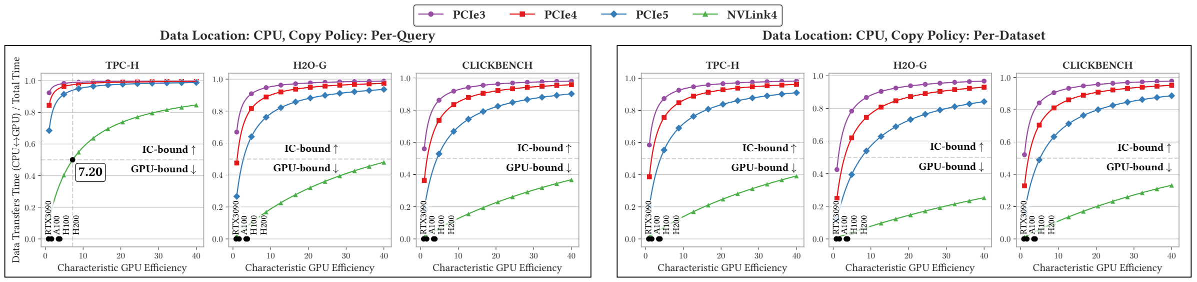

Figure 9: The ratio of communication vs. total execution time for different characteristic GPU efficiencies.

ratio of the GPU execution time and the total time, for a range of interconnect bandwidths. The interconnect bandwidths of PCIe 3.0, PCIe 4.0, PCIe 5.0, and NVLink 4.0 are marked on the x-axis.

Insight 13: For interconnect bandwidths > 137.30GB/s (unidirectional), the execution is mostly GPU-bound, for all evaluated benchmarks (TPC-H, H2O-G, ClickBench) and copy policies (per-query, per-dataset), even on H200.

Varying the GPU Efficiency. We vary the characteristic GPU efficiency from 0 to 10, almost 3x more than χ_{H200}, while keeping the interconnect bandwidths to the values that correspond to PCIe 3.0, PCIe 4.0, PCIe 5.0, and NVLink 4.0. Figure 9 shows the ratio of the data transfer time and the total time, for a range of GPU efficiency values. The characteristic GPU efficiencies of A100, H100, and H200 are marked on the x-axis for reference. For the TPC-H benchmark with the per-query copy policy, the results indicate that when the characteristic GPU efficiency is 7.2, the GPU execution time is equal to the data transfer time for NVLink 4.0. This means it would require the GPU to be almost 2x more efficient than H200 in order for data transfer to match the operator time on NVLink 4.0.

Insight 14: With interconnects such as NVLink 4.0, a GPU with characteristic efficiency almost 2x higher than H200 is needed, for the execution not to be GPU-bound for the TPC-H benchmark with the per-query copy policy. For H2O-G and ClickBench, the execution remains GPU-bound even with fast interconnects such as NVLink 4.0.

The Interconnect Outlook. The latest PCIe interconnect is PCIe 7.0 with a maximum theoretical peak of 242 GB/s (unidirectional). Taking into account Insight 7 and communication overheads, the expected maximum theoretical peak, without using pinned memory would be at most 96 GB/s. For NVLink interconnects, starting from NVLink 2.0 [30], the peak bandwidth is at least 150 GB/s (unidirectional).

Insight 15: For PCIe interconnects, the execution is likely to remain interconnect-bound for some workloads, even with the future PCIe 7.0 interconnect. For NVLink interconnects, the execution is likely to remain GPU-bound.

6 CONCLUSION

We present MaxBench, a framework for benchmarking, profiling, and modeling relational data analytics workloads. Using MaxBench, we explore how different GPU models and interconnects affect the performance for various data analytics workloads. We provide detailed metrics on data transfers and GPU execution times, and estimate query performance on different workloads, across different combinations of GPUs and interconnects, with a cost model that allows us to estimate the relative impact of the GPU computing power and the interconnect bandwidth on query execution. Using this cost model, we explored how increasing the interconnect bandwidth or the GPU efficiency beyond current capabilities would affect the overall performance, while providing various insights to help understand how powerful GPUs and fast interconnects are affecting the performance for data analytics workloads. In the future, more efficient GPU operators, such as [52], may also be considered.

ACKNOWLEDGMENTS

This work was supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID *sm94* on Alps. We thank the whole CSCS team for their excellent technical support.

REFERENCES

[1] Daniel Abadi, Anastasia Ailamaki, David G. Andersen, Peter Bailis, Magdalena Balazinska, Philip A. Bernstein, Peter A. Boncz, Surajit Chaudhuri, Alvin Cheung, AnHai Doan, Luna Dong, Michael J. Franklin, Juliana Freire, Alon Y. Halevy, Joseph M. Hellerstein, Stratos Idreos, Donald Kossmann, Tim Kraska, Sailesh Krishnamurthy, Volker Markl, Sergey Melnik, Tova Milo, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Jignesh M. Patel, Andrew Pavlo, Raluca A. Popa, Raghu Ramakrishnan, Christopher Ré, Michael Stonebraker, and Dan Suciu. 2022. The Seattle report on database research. *Commun. ACM* 65, 8 (2022), 72–79.

[2] Azim Afroozeh, Lotte Felius, and Peter Boncz. 2024. Accelerating GPU Data Processing using FastLanes Compression. In *Proceedings of the 20th International Workshop on Data Management on New Hardware* (Santiago, AA, Chile) *(DaMoN '24)*. Association for Computing Machinery, New York, NY, USA, Article 8, 11 pages. https://doi.org/10.1145/3662010.3663450

[3] RAPIDS AI. [n.d.]. cuDF: A GPU DataFrame Library. https://github.com/rapidsai/cudf. Accessed: 2025-02-19.

[4] Iya Arefyeva, David Broneske, Gabriel Campero Durand, Marcus Pinnecke, and Gunter Saake. 2018. Memory Management Strategies in CPU/GPU Database Systems: A Survey. In *Beyond Databases, Architectures and Structures. Facing the Challenges of Data Proliferation and Growing Variety - 14th International Conference, BDAS 2018, Held at the 24th IFIP World Computer Congress, WCC 2018, Poznan, Poland, September 18-20, 2018, Proceedings (Communications in Computer and Information Science)*, Vol. 928. Springer, 128–142.

[5] Abhinav Bhatele, Stephanie Brink, and Todd Gamblin. 2019. Hatchet: pruning the overgrowth in parallel profiles. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '19)*. Association for Computing Machinery, New York, NY, USA, Article 20, 21 pages. https://doi.org/10.1145/3295500.3356219

[6] BlazingDB. [n.d.]. BlazingSQL. https://github.com/BlazingDB/blazingsql. Accessed: February 27, 2025.

[7] David Boehme, Todd Gamblin, David Beckingsale, Peer-Timo Bremer, Alfredo Gimenez, Matthew LeGendre, Olga Pearce, and Martin Schulz. 2016. Caliper: performance introspection for HPC software stacks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, Utah) *(SC '16)*. IEEE Press, Article 47, 11 pages.

[8] Nils Boeschen, Tobias Ziegler, and Carsten Binnig. 2024. GOLAP: A GPU-in-Data-Path Architecture for High-Speed OLAP. *Proc. ACM Manag. Data* 2, 6, Article 237 (Dec. 2024), 26 pages. https://doi.org/10.1145/3698812

[9] Peter A Boncz, Marcin Zukowski, and Niels Nes. 2005. MonetDB/X100: Hyper-Pipelining Query Execution.. In *Cidr*, Vol. 5. 225–237.

[10] Sebastian Breß, Max Heimel, Norbert Siegmund, Ladjel Bellatreche, and Gunter Saake. 2014. GPU-Accelerated Database Systems: Survey and Open Challenges. *Trans. Large Scale Data Knowl. Centered Syst.* 15 (2014), 1–35.

[11] Sebastian Breß. 2014. The Design and Implementation of CoGaDB: A Column-oriented GPU-accelerated DBMS. *Datenbank-Spektrum* 14, 3 (2014), 199–209. https://doi.org/10.1007/s13222-014-0164-z

[12] Jiashen Cao, Rathijit Sen, Matteo Interlandi, Joy Arulraj, and Hyesoon Kim. 2023. GPU Database Systems Characterization and Optimization. *Proc. VLDB Endow.* 17, 3 (2023), 441–454.

[13] Jiashen Cao, Rathijit Sen, Matteo Interlandi, Joy Arulraj, and Hyesoon Kim. 2023. Revisiting Query Performance in GPU Database Systems. *CoRR* abs/2302.00734 (2023). https://doi.org/10.48550/ARXIV.2302.00734 arXiv:2302.00734

[14] Periklis Chrysogelos, Manos Karpathiotakis, Raja Appuswamy, and Anastasia Ailamaki. 2019. HetExchange: encapsulating heterogeneous CPU-GPU parallelism in JIT compiled engines. *Proc. VLDB Endow.* 12, 5 (Jan. 2019), 544–556. https://doi.org/10.14778/3303753.3303760

[15] ClickHouse Inc. [n.d.]. ClickBench — a Benchmark For Analytical DBMS. https://benchmark.clickhouse.com/. Accessed: 2025-02-19.

[16] NVIDIA Corporation. [n.d.]. NVIDIA Nsight Systems. https://developer.nvidia.com/nsight-systems. Accessed: 2025-02-19.

[17] Harish Doraiswamy, Vikas Kalagi, Karthik Ramachandra, and Jayant R. Haritsa. 2023. A Case for Graphics-Driven Query Processing. *Proc. VLDB Endow.* 16, 10 (June 2023), 2499–2511. https://doi.org/10.14778/3603581.3603590

[18] Rui Fang, Bingsheng He, Mian Lu, Ke Yang, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. 2007. GPUQP: query co-processing using graphics processors. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (Beijing, China) *(SIGMOD '07)*. Association for Computing Machinery, New York, NY, USA, 1061–1063. https://doi.org/10.1145/1247480.1247606

[19] Henning Funke, Sebastian Breß, Stefan Noll, Volker Markl, and Jens Teubner. 2018. Pipelined Query Processing in Coprocessor Environments. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1603–1618. https://doi.org/10.1145/3183713.3183734

[20] H2O.ai. [n.d.]. Database-like Ops Benchmark. https://h2oai.github.io/db-benchmark/. Accessed: 2025-02-19.

[21] Bingsheng He, Mian Lu, Ke Yang, Rui Fang, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. 2009. Relational query coprocessing on graphics processors. *ACM Trans. Database Syst.* 34, 4, Article 21 (Dec. 2009), 39 pages. https://doi.org/10.1145/1620585.1620588

[22] Dong He, Supun C Nakandala, Dalitso Banda, Rathijit Sen, Karla Saur, Kwanghyun Park, Carlo Curino, Jesús Camacho-Rodríguez, Konstantinos Karanasos, and Matteo Interlandi. 2022. Query processing on tensor computation runtimes. *Proceedings of the VLDB Endowment* 15, 11 (July 2022), 2811–2825. https://doi.org/10.14778/3551793.3551833

[23] Jiong He, Mian Lu, and Bingsheng He. 2013. Revisiting Co-Processing for Hash Joins on the Coupled CPU-GPU Architecture. *Proc. VLDB Endow.* 6, 10 (aug 2013), 889–900. https://doi.org/10.14778/2536206.2536216

[24] HEAVY.AI. [n.d.]. HeavyDB. https://www.heavy.ai/product/heavydb. Accessed: February 27, 2025.

[25] Max Heimel, Michael Saecker, Holger Pirk, Stefan Manegold, and Volker Markl. 2013. Hardware-oblivious parallelism for in-memory column-stores. *Proc. VLDB Endow.* 6, 9 (July 2013), 709–720. https://doi.org/10.14778/2536360.2536370

[26] Marko Kabić, Shriram Chandran, and Gustavo Alonso. 2025. Maximus: A Modular Accelerated Query Engine for Data Analytics on Heterogeneous Systems. *Proc. ACM Manag. Data* 3, 3, Article 187 (June 2025), 25 pages. https://doi.org/10.1145/3725324

[27] Tomas Karnagel, René Müller, and Guy M. Lohman. 2015. Optimizing GPU-accelerated Group-By and Aggregation. In *ADMS@VLDB*. https://api.semanticscholar.org/CorpusID:5017248

[28] Kinetica. 2024. Kinetica: The Database for Time & Space. https://www.kinetica.com/ Accessed: 2024-12-02.

[29] Artem Kroviakov, Petr Kurapov, Christoph Anneser, and Jana Giceva. 2024. Heterogeneous Intra-Pipeline Device-Parallel Aggregations. In *Proceedings of the 20th International Workshop on Data Management on New Hardware* (Santiago, AA, Chile) *(DaMoN '24)*. Association for Computing Machinery, New York, NY, USA, Article 3, 10 pages. https://doi.org/10.1145/3662010.3663441

[30] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2020. Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 1633–1649.

[31] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2022. Triton Join: Efficiently Scaling to a Large Join State on GPUs with Fast Interconnects. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 1017–1032.

[32] Hubert Mohr-Daurat, Xuan Sun, and Holger Pirk. 2023. BOSS - An Architecture for Database Kernel Composition. *Proc. VLDB Endow.* 17, 4 (Dec. 2023), 877–890. https://doi.org/10.14778/3636218.3636239

[33] Hamish Nicholson, Aunn Raza, Periklis Chrysogelos, and Anastasia Ailamaki. 2023. Hetcache: synergising NVMe storage and GPU acceleration for memory-efficient analytics. In *13th Annual Conference on Innovative Data Systems Research (CIDR 2023)*.

[34] NVIDIA. 2024. nvbandwidth. https://github.com/NVIDIA/nvbandwidth. Accessed: 26-Feb-2025.

[35] NVIDIA Developers Forum. [n.d.]. The L2 cache hit rate of A100(A800) is very low compared to RTX3090. https://forums.developer.nvidia.com/t/the-l2-cache-hit-rate-of-a100-a800-is-very-low-compared-to-rtx3090/320271. Accessed: 2025-02-19.

[36] Johns Paul, Jiong He, and Bingsheng He. 2016. GPL: A GPU-based Pipelined Query Processing Engine. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 1935–1950. https://doi.org/10.1145/2882903.2915224

[37] Johns Paul, Shengliang Lu, Bingsheng He, and Chiew Tong Lau. 2021. MG-Join: A Scalable Join for Massively Parallel Multi-GPU Architectures. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1413–1425. https://doi.org/10.1145/3448016.3457254

[38] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James R. Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. In *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*. IEEE Computer Society, 13–24.

[39] Syed Mohammad Aunn Raza, Periklis Chrysogelos, Panagiotis Sioulas, Vladimir Indjic, Angelos Christos Anadiotis, and Anastasia Ailamaki. 2020. GPU-accelerated data management under the test of time. In *Online proceedings of the 10th Conference on Innovative Data Systems Research (CIDR)*.

[40] Viktor Rosenfeld, Sebastian Breß, and Volker Markl. 2023. Query Processing on Heterogeneous CPU/GPU Systems. *ACM Comput. Surv.* 55, 2 (2023), 11:1–11:38.

[41] Viktor Rosenfeld, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2019. Performance Analysis and Automatic Tuning of Hash Aggregation on GPUs. In *Proceedings of the 15th International Workshop on Data Management on New Hardware* (Amsterdam, Netherlands) *(DaMoN'19)*. Association for Computing Machinery, New York, NY, USA, Article 8, 11 pages. https://doi.org/10.1145/3329785.3329922

[42] Ran Rui, Hao Li, and Yi-Cheng Tu. 2020. Efficient Join Algorithms for Large Database Tables in a Multi-GPU Environment. *Proc. VLDB Endow.* 14, 4 (dec 2020), 708–720. https://doi.org/10.14778/3436905.3436927

[43] Ran Rui and Yi-Cheng Tu. 2017. Fast Equi-Join Algorithms on GPUs: Design and Implementation. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management* (Chicago, IL, USA) *(SSDBM '17)*. Association for Computing Machinery, New York, NY, USA, Article 17, 12 pages. https://doi.org/10.1145/3085504.3085521

[44] Anil Shanbhag, Samuel Madden, and Xiangyao Yu. 2020. A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1617–1632. https://doi.org/10.1145/3318464.3380595

[45] Panagiotis Sioulas, Periklis Chrysogelos, Manos Karpathiotakis, Raja Appuswamy, and Anastasia Ailamaki. 2019. Hardware-Conscious Hash-Joins on GPUs. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 698–709. https://doi.org/10.1109/ICDE.2019.00068

[46] The Apache Software Foundation. [n.d.]. A cross-language development platform for in-memory analytics. https://arrow.apache.org/. Accessed: 2025-02-19.

[47] The Transaction Processing Council. [n.d.]. The TPC-H Benchmark. https://www.tpc.org/tpch/. Accessed: 2025-02-19.

[48] Lasse Thostrup, Gloria Doci, Nils Boeschen, Manisha Luthra, and Carsten Binnig. 2023. Distributed GPU Joins on Fast RDMA-capable Networks. *Proc. ACM Manag. Data* 1, 1, Article 29 (may 2023), 26 pages. https://doi.org/10.1145/3588709

[49] Diego G. Tomé, Tim Gubner, Mark Raasveldt, Eyal Rozenberg, and Peter A. Boncz. 2018. Optimizing Group-By and Aggregation using GPU-CPU Co-Processing. In *ADMS@VLDB*. https://api.semanticscholar.org/CorpusID:52895287

[50] Ben van Werkhoven, Jason Maassen, Frank J. Seinstra, and Henri E. Bal. 2014. Performance Models for CPU-GPU Data Transfers. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26-29, 2014*. IEEE Computer Society, 11–20.

[51] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (April 2009), 65–76. https://doi.org/10.1145/1498765.1498785

[52] Bowen Wu, Dimitrios Koutsoukos, and Gustavo Alonso. 2025. Efficiently Processing Joins and Grouped Aggregations on GPUs. *Proc. ACM Manag. Data* 3, 1, Article 39 (Feb. 2025), 27 pages. https://doi.org/10.1145/3709689

[53] Bobbi W. Yogatama, Weiwei Gong, and Xiangyao Yu. 2022. Orchestrating data placement and query execution in heterogeneous CPU-GPU DBMS. *Proc. VLDB Endow.* 15, 11 (July 2022), 2491–2503. https://doi.org/10.14778/3551793.3551809

[54] Yichao Yuan, Advait Iyer, Lin Ma, and Nishil Talati. 2025. Vortex: Overcoming Memory Capacity Limitations in GPU-Accelerated Large-Scale Data Analytics. *Proc. VLDB Endow.* 18, 4 (May 2025), 1250–1263. https://doi.org/10.14778/3717755.3717780

[55] Yuan Yuan, Rubao Lee, and Xiaodong Zhang. 2013. The Yin and Yang of Processing Data Warehousing Queries on GPU Devices. *Proc. VLDB Endow.* 6, 10 (2013), 817–828.

[56] Shuhao Zhang, Jiong He, Bingsheng He, and Mian Lu. 2013. OmniDB: towards portable and efficient query processing on parallel CPU/GPU architectures. *Proc. VLDB Endow.* 6, 12 (Aug. 2013), 1374–1377. https://doi.org/10.14778/2536274.2536319

[57] Marcin Żukowski et al. 2009. *Balancing vectorized query execution with bandwidth-optimized storage*. SIKS.