OLAP on Modern Chiplet-Based Processors

Alessandro Fogli
Imperial College London
a.fogli18@imperial.ac.uk

Bo Zhao
Aalto University
bo.zhao@aalto.fi

Peter Pietzuch
Imperial College London
prp@imperial.ac.uk

Maximilian Bandle
TU Munich
bandle@in.tum.de

Jana Giceva
TU Munich
jana.giceva@in.tum.de

ABSTRACT

Chiplet-based CPUs, which combine multiple independent chips on a single die, allow hardware to scale to higher core counts at the cost of more memory heterogeneity and performance variability. This raises challenges when existing query engines are deployed on chiplet-based CPUs: current designs make assumptions about uniform memory access, cache locality and consistent core performance, which lead to ineffective CPU utilization.

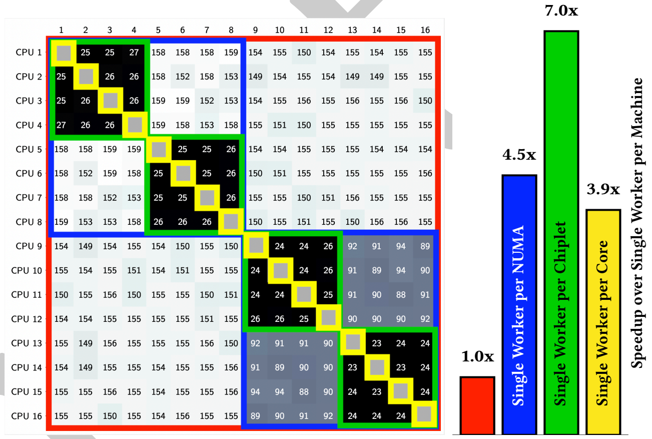
In this paper, we analyse these performance impact when query engines ignore chiplet-specific properties. We show that naive deployment approches result in a significant degradation in query processing efficiency, resulting in non-linear scaling even within a single CPU socket domain. Based on a comprehensive experimentation evaluation, we propose approaches to deploy engines on chiplet-based CPUs with maximum performance. We show that distributing processing tasks according to chiplet topology ensures optimal resource utilization, maximizing performance and scalability. Such a chiplet-conscious deployment strategy of a shared-nothing query engine can enhance performance by up to 7× compared to hardware-oblivious approaches.

PVLDB Reference Format:
Alessandro Fogli, Bo Zhao, Peter Pietzuch, Maximilian Bandle, and Jana Giceva. OLAP on Modern Chiplet-Based Processors. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

Chiplet-based CPUs are a new technology adopted by most major processor manufacturers. Such CPUs are made up of multiple smaller chips, known as *chiplets*, that are interconnected with a high-bandwidth fabric to operate seamlessly as a single multicore CPU [9, 35, 40, 41]. This modular approach allows for better yields during manufacturing, offering a practical solution to scale to higher core counts. Chiplet-based designs also allow for individual chiplets to be tailored to specific functions, from computation to memory access, thus enhancing the performance of bespoke workloads [39, 49].



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPU 1		25	25	27	158	158	158	159	154	155	150	154	155	154	155	155
CPU 2	25		26	26	158	152	158	153	149	154	155	154	149	149	155	155
CPU 3	25	26		26	159	159	152	153	154	155	156	155	156	155	156	150
CPU 4	27	26	26		159	158	153	158	155	151	150	155	155	155	155	155
CPU 5	158	158	159	159		25	25	26	154	154	155	155	155	154	154	154
CPU 6	158	152	159	158	25		25	26	150	151	155	155	155	155	155	156
CPU 7	158	158	152	153	25	25		26	155	155	150	151	156	155	155	155
CPU 8	159	153	153	158	26	26	26		150	155	151	155	150	156	156	155
CPU 9	154	149	154	155	154	150	155	150		24	24	26	92	91	94	89
CPU 10	155	154	155	151	154	151	155	155	24		24	26	91	89	94	90
CPU 11	150	155	156	150	155	155	150	151	24	24		25	91	90	88	91
CPU 12	154	154	155	155	155	155	151	155	26	26	25		90	90	90	92
CPU 13	155	149	156	155	155	155	156	150	92	91	91	90		23	24	24
CPU 14	154	149	155	155	154	155	155	156	91	89	90	90	23		23	24
CPU 15	155	155	156	155	154	155	155	156	94	94	88	90	24	23		24
CPU 16	155	155	150	155	154	156	155	155	89	90	91	92	24	24	24	

1.0x
4.5x
Single Worker per NUMA
7.0x
Single Worker per Chiplet
3.9x
Single Worker per Core
Speedup over Single Worker per Machine

(a) Core-to-core latency heatmap (AMD Ryzen 1950) **(b)** Speedup (TPC-H)

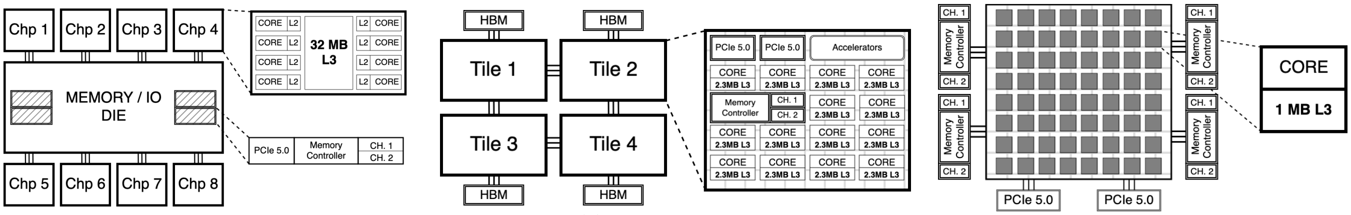
Fig. 1: Different deployment strategies on chiplet-based CPUs

Chiplet-based CPUs, however, introduce new types of heterogeneity: (1) they exhibit diverse access times to partitioned L3 caches across chiplets; and they have a range of inter-core (2) latencies and (3) bandwidths. In Fig. 1a, we show that inter-core latencies can vary by up to 6×, even within the same CPU socket domain. As we demonstrate experimentally in this paper, this heterogeneity of chiplet-based CPUs imposes new challenges on the performance of multi-core query engines, going beyond the challenges introduced by non-uniform memory access (NUMA) [12, 28, 33, 44].

To date, there has been little attention paid to answer *how to deploy query engines efficiently on chiplet-based CPUs.* Existing commercial designs of multi-core query engines employ different approaches for mapping worker instances (tasks) to CPU cores (see Fig. 1a): Redshift [6, 19] and Greenplum [2] (shown in yellow) partition resources into slices, made up of individual cores, and assign tasks to these slices; SingleStore [24] (shown in blue) is NUMA-aware, and hence assigns tasks based on NUMA domains; other systems, e.g., Presto [46] and SparkSQL [18], (shown in red) use all CPU cores, permitting the allocation of any task to any CPU core. We conclude that there is no agreement on a single best allocation approach, and none of the above-mentioned approaches explicitly consider the architectural features of chiplet-based CPUs.

In NUMA systems, the best approach is to assign tasks to the CPU cores that are closest to the main memory where the accessed data resides [44]. Such an assignment respects data locality and minimizes remote memory accesses, thus leveraging the large local memory available per NUMA domain. In contrast, chiplet-based

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX



(a) AMD EPYC Milan **(b)** Intel Sapphire Rapids **(c)** ARM Graviton 3

Fig. 2: Chiplet-based architectures

CPUs partition a more constrained L3 cache, even at a core level. Therefore, assigning tasks to a subset of CPU cores, as for NUMA, improves cache efficiency but restricts the available memory bandwidth; assigning tasks to use only a single chiplet's cores results in limited bandwidth to other chiplets' L3 caches. This reduces the available cache size and leads to slower use of main memory for data exceeding local cache capacity. This trade-off becomes more pronounced with multiple chiplets in a single NUMA node, i.e., with a reduced core count relative to a full NUMA domain.

We empirically explore the impact of chiplet-based architectures on multi-core query engines and aim to identify efficient resource allocation strategies. Our study offers practical recommendations on how to optimize query engine designs for chiplet-based CPUs:

(1) Task deployment at chiplet granularity. We demonstrate that users should deploy the tasks of a query engine on a chiplet-based machine at *chiplet granularity* for the best performance. As shown in Fig. 1b, this chiplet-aware deployment policy boosts performance for queries by up to 7× compared to approaches that use a database worker instance per CPU core and up to 2× compared to NUMA-aware assignments.

A deployment approach at chiplet granularity has several advantages: (i) it avoids an uneven data distribution, which otherwise leads to excessive inter-chiplet communication and congestion, reducing performance and scalability; (ii) it is suitable for complex queries that exploit efficient TableScan and Repartition operations; and (iii) it benefits from the adoption of Bloom filters and hash joins instead of nested loop joins, as they are easily parallelizable and distributable across worker instances, cutting down the data transfer volume.

(2) Cache consciousness. When the amount of accessed data is below the capacity of the local L3 cache in a single chiplet, users should deploy tasks at a chiplet granularity where they only use cores within a single chiplet and local cache. Conversely, when the data amount is larger than the L3 cache size but smaller than the combined size of L3 caches, users should assign tasks to a select group of cores, ensuring one core per chiplet. In a scenario with large amounts of accessed data, i.e., cache misses happen frequently, the performance of both approaches becomes similar.

(3) Data skew. Queries involving extensive data scans or significant data shuffling benefit from the enhanced locality that allocation at chiplet granularity can provide. Overall, all queries see improvements with such an assignment, even on complex benchmark suites such as TPC-DS [14]. Data skew, however, impacts how effectively the system can leverage the hardware topology, reducing the performance gains of per-chiplet assignment in the presence of skewed data distributions. We relate this primarily to increased cross-chiplet communication, which congests the interconnect congestion, and redistribution traffic to address data skewness.

The rest of the paper and its contributions are structured as follows:

- We give background on chiplet architectures, analyze their inter-core latencies and bandwidth, and provide insights into their performance traits (§2).
- We present a detailed exploration of the design space for multi-core query engines, considering designs that deploy a single worker instance (task) per CPU core to assigning tasks to the whole machine (§3).
- We empirically evaluate different deployments of query processing engines (§4.2), their scalability (§4.3), and efficiency across different worker counts, workloads, and data distributions (§4.5).

2 BACKGROUND AND MOTIVATION

Chiplet-based processors represent a paradigm shift in semiconductor design. They depart from monolithic integrated circuits by employing a modular approach. These processors integrate multiple smaller semiconductor dies, called *chiplets*, onto a single package or substrate to create a functional unit. This new design aims to improve performance, lower production costs, and solve problems that older, single-piece designs face, such as limited scalability and high power consumption.

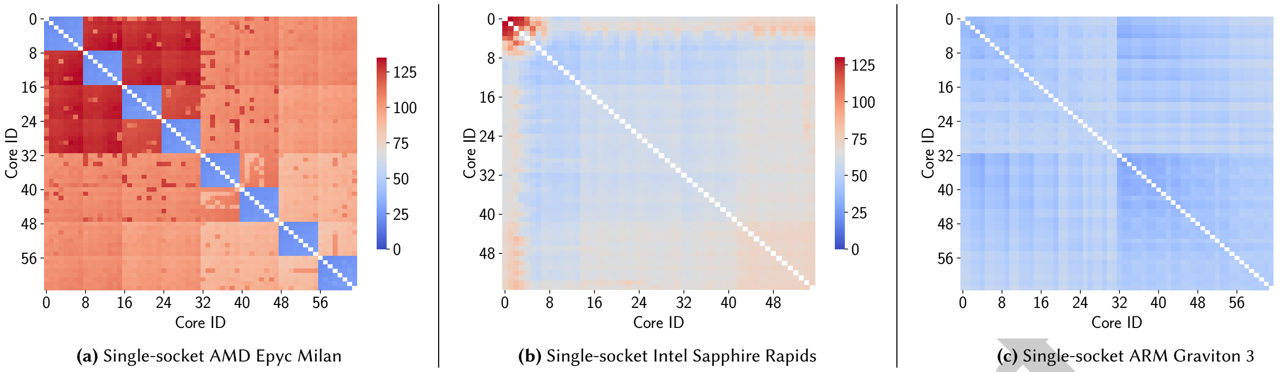
This modern architecture is gaining popularity among major semiconductor manufacturers, each one using its own unique technology and design. Leaders in this field include AMD, Intel, and ARM. In this section, we will take a closer look at three leading architectures from these companies: AMD's Epyc Milan, Intel's Sapphire Rapids, and ARM's Graviton 3 processors.

2.1 Overview of chiplets-based architectures

Table 1 presents a comparison of the processors we analyzed and Fig. 2 shows their respective architectures.

AMD EPYC Milan. The first one is the AMD EPYC Milan processor [35]. It employs a chiplet-based architecture for enhanced scalability, featuring eight 7 nm compute chiplets linked to a 14 nm central I/O die for memory and I/O management. In this design, chiplets, smaller silicon pieces containing CPU cores, differ from the broader term "die," which can refer to any silicon part, including the I/O die. Each processor can support up to 64 cores, 128 threads, and offers up to 256 MB of L3 cache across the socket, with up to 32 MB per chiplet.

Despite the L3 cache being local to each chiplet, AMD's Infinity Fabric enables cores to access cache across chiplets, but with increased latency. The processor supports PCIe 4.0, allowing up to 32 GT/s per lane, and integrates eight DDR4 memory channels,



(a) Single-socket AMD Epyc Milan

(b) Single-socket Intel Sapphire Rapids

(c) Single-socket ARM Graviton 3

Fig. 3: Core-to-Core Latency (ns).

Tab. 1: Comparative Architectural Features and Specifications of AMD Milan, Intel Sapphire Rapids and ARM Graviton 3 Processors.

	EPYC Milan	Sapphire Rapids	Graviton 3
Vendor	AMD	Intel	ARM
Core Count	Up to 64 cores	Up to 56 cores	64 cores
Number of Chiplets	Up to 8 chiplets	Up to 4 chiplets	Seven chiplets
L3 Cache Size	Up to 256 MB	Up to 112.5 MB	64MB
L3 Size Per Core	Up to 4 MB/core	Up to 2 MB/core	1MB
Memory Support	DDR4, up to 8 channels per CPU	DDR5, up to 8 channels per CPU	DDR5, up to 300 GB/sec bandwidth
Interconnect Technology	Infinity Fabric	EMIB	LIPINCON
Features	Unified 32MB L3 cache per 8 cores; PCI-Express Gen 4.0 support;	PCI Express 5.0; HBM2 memory (L4 cache) on specific models;	No need to go through the package substrate when communicating between chiplets;

providing shared access from the I/O die to the memory pool without dedicated DRAM banks per chiplet. The newest AMD EPYC Milan processors feature support for DDR5 memory. The access efficiency and latency can vary depending on the physical distance and routing through the Infinity Fabric.

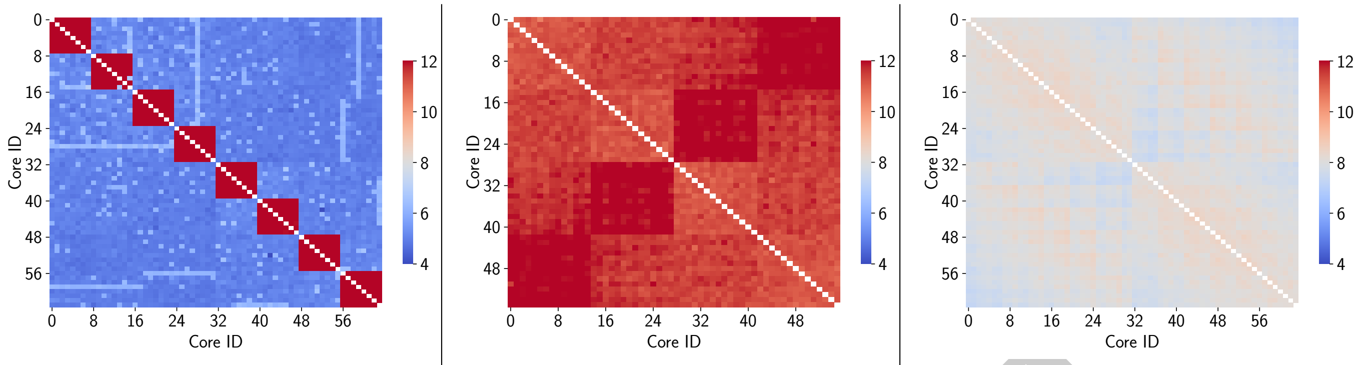
Intel Sapphire Rapids. The Intel CPU introduces a new 4-tile design featuring up to 60 cores. Intel's use of the term 'tile' refers to a concept very similar to AMD's 'chiplet'. While AMD's chiplets typically separate computational cores from I/O functions across different dies, Intel's 'tiles' can include a broader range of computing functions within each of them. In fact, each tile houses up to 14 cores and connects to two DDR5 memory channels, contributing to a total bandwidth of over 307.2 GB/s at DDR5-4800 speeds. The CPU offers 2 MB of dedicated L2 cache per core and a shared 28.125 MB of L3 cache per tile, totaling 112.5 MB across the processor. All cores can access the cache across tiles, mirroring AMD Milan's design [41].

The latest Sapphire Rapids CPUs also integrate up to 16 GB of High-Bandwidth Memory (HBM) per tile, reaching a 64 GB total [38]. This proximity of memory to the core greatly benefits tasks that handle large data sets, effectively acting like a vast "L4" cache but introducing variable access times similar to NUMA systems. The processor also supports PCIe 5.0, achieving up to 32 GT/s per lane.

Intel's Sapphire Rapids CPU utilizes Embedded Multi-die Interconnect Bridge (EMIB) for horizontal integration and Foveros for vertical stacking to improve data transmission. EMIB acts as a bridge for horizontal integration, connecting different semiconductor components side by side within the same package. Foveros complements EMIB by adding vertical integration into the mix. With Foveros, Intel stacks different tiles on top of one another, reducing data travel distance. This combination minimizes latency and boosts data speed across the processor.

ARM Graviton 3. The ARM processor adopts a unique chiplet design, comprising seven different silicon components arranged in a manner unlike the chiplet arrangements seen in AMD and Intel processors [9]. Unlike AMD's approach, where a central I/O and memory die are surrounded by multiple computing chiplets, the Graviton 3 takes a different path and places all 64 of its cores in a single chiplet. Each core has a separate L3 cache of 1 MB size for a total of 64 MB of L3 cache. The DDR5 memory controllers and the PCI-Express 5.0 peripheral controllers are located separately from these central cores. This design emphasizes a distinct separation of the core processing area from the memory and peripheral control sections. Communication between cores takes place through the use of LIPINCON (Low-latency Inter-chiplet Passive Interconnect) communication technology. LIPINCON serves to seamlessly link the processor's silicon components, ensuring rapid and coherent data exchange across the CPU.

Key Differences Among Chiplet-Based Processor. The chiplet-based processor landscape is as diverse as it is innovative. All the processors, while similar in core count, take a unique approach to modular design as seen in their varying number of chiplets. One notable distinction is the cache architecture. For instance, AMD's Milan offers up to 256 MB of L3 cache, which is significantly more compared to its counterparts, impacting data retrieval efficiency and hence throughput and latency. In addition, in Intel and ARM processors, L3 cache is allocated per core, making it local exclusively to that core. Conversely, AMD assigns L3 cache to chiplets, making it local to all cores within the same chiplet. This significantly affects core-to-core communication, as only in AMD two cores can communicate through a cache shared locally between them. Memory support also varies; Sapphire Rapids and Graviton 3 support the newer DDR5, while Milan, still on DDR4, balances this with its larger LLC. Another point of difference is in their inter-chiplet communication technologies, where each processor has its own method to optimize data transfer and integration. AMD's Infinity Fabric aim to facilitate scalable multi-core integration with shared L3 cache



(a) Single-socket AMD Epyc Milan

(b) Single-socket Intel Sapphire Rapids

(c) Single-socket ARM Graviton 3

Fig. 4: Core-to-Core Bandwidth (GB/s).

across chiplets, enhancing core scalability at the cost of variable inter-chiplet latency. Intel combines EMIB for efficient horizontal tile connectivity with Foveros for vertical stacking, reducing data travel distances and minimizing latency. ARM's strategy focuses on low latency, keeping all the cores on the same computing chiplet, connected to the other silicon components via the LIPINCON.

Even subtle differences in design can lead to notable variations in performance. Therefore, transitioning from these design specifics, we next delve into a detailed performance evaluation of these processors. We use microbenchmarks to explore how these different chiplet architectures influence core-to-core latency, bandwidth, and overall system throughput. Our goal is to offer a comprehensive understanding of how these chiplet-design differences might affect application performance and clear and actionable insights on how to use these modern processors.

2.2 Testbed

We conduct the experiments on a diverse set of machines:
(1) AMD EPYC Milan features a single-socket AMD EPYC 7713 processor with one NUMA domain equipped with 64 CPU cores and 256 MB of L3 cache divided among 8 chiplets, and 512 GB RAM. It also has 64 MB of L2 cache and runs Ubuntu 23.04.
(2) Intel Sapphire Rapids consists of a single NUMA Domain Intel Xeon Platinum 8480+ processor with 56 CPU cores, 192 GB of RAM, 1.875 MB of L3 cache and 2 MB of L2 cache per core, running the OS of Ubuntu 22.04.6.
(3) ARM Graviton 3 has a single-socket ARM Graviton 3 processor with one NUMA domain, providing 64 CPU cores, 128 GB of RAM, 32 MB of L3 cache and 1 MB of L2 cache per core with Ubuntu 22.04.
(4) Intel Xeon Gold features a single-socket 3rd Gen. Intel Xeon Gold processor with one NUMA domain, providing 24 CPU cores, 128 GB of RAM, 36 MB of L3 cache and 30 MB of L2 cache with the OS of Ubuntu 22.04.

2.3 Core-to-core latency

This section evaluates the core-to-core latency within the three chiplet-based architectures that we discussed in the previous section. More specifically, for this evaluation we used: the dual-socket AMD EPYC Milan 7713; the single-socket Intel Sapphire Rapids; and the ARM Graviton 3 processor. Figure 3 illustrates the heatmap of core-to-core latency for each processor.

The AMD EPYC Milan processor shows a wide range in latency, with some core pairs differing by more than ten times, as illustrated in Figure 3a. On average, latency within the same socket is 106 ns, and it increases to 182 ns for transfers between sockets. However, within a single chiplet, the average latency drops to just 24 ns.

In contrast, the Intel Sapphire Rapids processor demonstrates a more consistent latency across its cores, averaging 59 ns within the same socket. The heatmap in Figure 3b for this processor shows closely grouped latency values, indicating uniform core-to-core communication times across the chip. This uniformity suggests that Intel's design favors consistent data transfer speeds, which can be beneficial for parallel processing tasks and maintaining efficient performance in multi-threaded applications.

Figure 3c shows that the average latency on Graviton 3 is 48 ns, and this latency remains consistently below 59 ns throughout the entire chip. The heatmap shows a mixed pattern of red and blue squares with 4 groups of thread pairs with generally lower latency than the others. This variation is like due to the chip's internal architecture: i.e., the arrangement of core clusters, the layout of interconnects and the proximity to shared resources.

Insights: *AMD processors exhibit varying latency due to their chiplet design, affecting core-to-core communication and cache access times. This variability requires careful resource management for applications sensitive to latency. Conversely, Intel Sapphire Rapids and ARM Graviton processors achieve consistent latency by avoiding shared caches between cores. The AMD CPU minimizes latency by using a single computing chiplet, eliminating cross-chiplet communication. Intel reduces latency in its designs with Foveros technology, which employs vertical stacking to shorten distances between chiplets.*

2.4 Core-to-core bandwidth

We continue our analysis of the previously mentioned processors by examining the bandwidth sustained between core-to-core communication. In Figure 4a, the AMD Milan processor exhibits a distinct and consistent bandwidth pattern in core-to-core communication, mirroring the findings from the latency analysis. Within the same chiplet, cores can communicate at a rate of 12 GB/s. However, this speed decreases by 50% to 6 GB/s for communication between different chiplets, indicating significant variation based on the core location. As seen during the core-to-core latency test, one NUMA domain has a slightly higher average bandwidth than the other. We attribute that to the specific processor's topology. In both sockets, computational chiplets link to a memory controller die, which connects to memory banks. Likely, the memory banks tied to one controller are more distant than those linked to the other controller.

On the other hand, Intel's processor exhibits a more even distribution of bandwidth. From Figure 4b, we observe distinct groups of cores where bandwidth peaks at 12.5 GB/s. Beyond these four groups, bandwidth decreases modestly, averaging 11 GB/s and occasionally falling to 10.5 GB/s, marking a reduction of up to 16%. Despite the overall uniformity in bandwidth, like the variability seen in the core-to-core latency test, the delineation of the four chiplets constituting the processor is visible even if negligible.

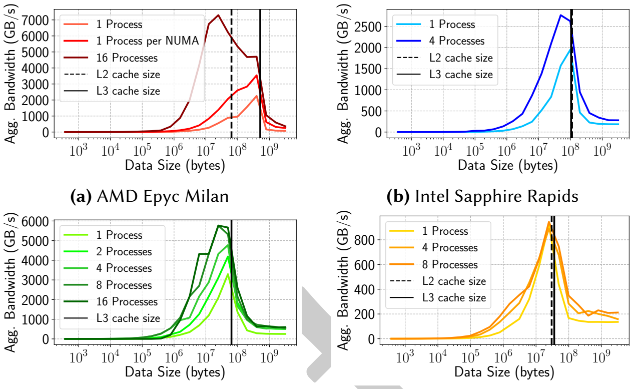
The ARM Graviton 3 processor also demonstrates a notably consistent communication bandwidth profile between cores. This is evident from the heatmap in Figure 4c, which shows no significant discrepancies, but rather a bandwidth fluctuating within a narrow range of 7.2 GB/s to 8.6 GB/s. Additionally, Figure 4c highlights that the regions closer to the diagonal primarily display a red coloration, signifying enhanced bandwidth. This suggests that the bandwidth tends to be higher when communication occurs between cores that are in close proximity to each other.

Insights: *As expected, the inter-core bandwidth tests align with previous latency observations for AMD Milan, Intel Sapphire Rapids and ARM Graviton 3, highlighting the impact of their architectural designs.*

2.5 Aggregate memory bandwidth

In our final evaluation, we analyze the memory bandwidth of chiplet-based processors running the STREAM benchmark which measures sustainable memory bandwidth. It works by performing a simple operation on stored arrays of data. We evaluate the COPY function, which consists of copying the contents of one array into another, element by element. We run the workload under a variety of configurations to assess how the aggregate bandwidth changes when operating with single or multiple parallel processes that conform to the same benchmark standards. The rationale behind using multiple processes is to examine if such a setup could take advantages of chiplet architectures, where each chiplet operates more autonomously. With multiple processes we expect to see enhanced efficiency within the chiplet boundaries, reducing L3 cache contention. However, in AMD's and ARM's processors, memory controller dies are shared between chiplets and we can expect more stress on them due to concurrent accesses. In contrast, a process able to use all available resources can optimize memory access patterns and cache usage, resulting in more efficient use of memory bandwidth.

Figure 5 shows the results. When the array size is too small, no difference in bandwidth is observed across the evaluated configurations. As the array size increases, the three chiplet-based processors exhibit higher aggregate bandwidths when utilizing multiple processes. Specifically, the AMD Milan and Intel Sapphire Rapids processors display the best aggregate bandwidths using one process per chiplet, achieving 7297 GB/s and 2768 GB/s, respectively. Moreover, the AMD Milan processor, the only one tested with 2 NUMA domains, shows that this configuration outperforms even the one that accounts for the presence of NUMA architecture, reaching only 3517 GB/s. Even the ARM Graviton 3 processor, despite having a single computational chiplet, shows the best bandwidth size with 16 processes, peaking at 5784 GB/s. As expected, the Intel Xeon Gold, featuring a monolithic processor design, stands out as the exception, displaying no variation in performance across different configurations, underscoring a distinct behavior from chiplet-based processors.



(a) AMD Epyc Milan

(b) Intel Sapphire Rapids

(c) ARM Graviton 3

(d) 3rd Gen. Intel Xeon Gold

Fig. 5: STREAM Benchmark

The reason a single process cannot fully exploit the machine's bandwidth is due to the chiplet-based architecture of these processors. Accessing partitioned cache and memory across chiplets and NUMA domains introduces higher latency compared to local accesses, thus reducing achievable bandwidth. However, as data sizes exceed cache capacities, all processors experience a decline in bandwidth due to increased reliance on slower main memory.

The different values of aggregate bandwidth are attributed to the varying characteristics of the machines used. For example, AMD Epyc Milan has a significantly higher number of cores compared to the others. The different values of aggregate bandwidth indeed depend on the total number of cores and the configuration of NUMA domains. The AMD Milan Processor, having the highest number of cores and, with two sockets, the most memory channels available, achieves increased bandwidth. Among the processors tested, the Intel processors have the fewest cores, with 48 in the Sapphire Rapids processor and 24 in the Xeon Gold processor.

One more notable difference is observed in the AMD Milan processor, which is the only one among the tested to feature a smaller L2 cache compared to its L3 cache, unlike other processors where the L2 cache is similar or larger in size. With 16 processes, the optimal bandwidth is reached with a 20 MB array, split into 16 parts of 1.25 MB for each process, ensuring each part fits within the faster L2 cache to enhance data access speed and efficiency. As expected, when the data size exceeds the size of the L2 cache, the bandwidth decreases. The same effect is not seen in the configurations since, with fewer processes, the array is not divided sufficiently to fit into the L2 cache.

Insights: *Contrary to expectations, directing tasks to specific core subsets boosts aggregate bandwidth in chiplet-based architectures. This finding defies the assumption that aggregate bandwidth efficiency* *depends on uniform communication between cores, as evidenced by similar improvements in Intel and ARM processors, despite their uniform inter-core latency and bandwidth. The key factor is the more efficient cache utilization and improved data locality achieved through process assignment at chiplet granularity or finer.*

3 DESIGN SPACE OF QUERY ENGINES

In the previous section, we examined chiplet-based processors through targeted microbenchmarks, demonstrating the substantial impact of their distinctive design on bandwidth and latency-sensitive applications. Now, we shift our focus to distributed query engines, which frequently rely on these chiplet-based processors, to explore how their deployment tactics, design fundamentals and strategies for resource allocation and data communication complement the capabilities of chiplet-based technology.

3.1 Architecture of distributed query engines

From a structural and design perspective, distributed cloud-based query engines (e.g., AWS Redshift [30], Athena [8], Google's Big Query [27], Microsoft's Polaris [17]), on-premise data warehouses (e.g., Exadata [7] and Teradata [4]) and big data systems (e.g., Hadoop [1], Presto [46] and Spark [18]) all share some similarities. These systems are designed with scalability and cost-effective customizations in mind, featuring a compute layer made up of *database worker instances*, which execute the incoming queries (i.e., a pipeline of the query [17], or a big data job). They operate on data fragments, which are assigned to each database worker instance. The worker instances process many fragments in parallel by executing a dedicated list of tasks, as assigned by the coordinator. The coordinator is responsible for admitting, parsing, planning and optimizing the queries, in addition to orchestrating how the query's tasks are distributed across worker instances.

The worker instances communicate with each other over the network, either to exchange data when performing joins over multiple tables or to send the results to the coordinator. The general rule is to minimize network communication as much as possible [22]. Hence, each database worker instance processes its own queue of tasks on its own data partition (fragments), which both maximizes data locality and reduces network traffic. The coordinator computes an optimal distributed cost-based query plan, which accounts for all factors involved, including data distribution (to minimize the overhead of communication among the database worker instances) and data placement within multi-processors [13, 24, 46].

Current distributed query engines have different design configurations for their worker instance deployment within a single multiprocessor machine. In the next section, we describe the main characteristics of the most widely adopted designs, highlighting their advantages and disadvantages.

3.2 Design space overview and trade-offs

When considering the adaptation of query engine systems for use with heterogeneous chiplet-based processors, we face the challenge of how to effectively allocate the workers across varying sizes of computing resources. This entails a detailed exploration of the design space, which encompasses a range of scenarios from deploying a single worker instance on a single CPU core to utilizing the entire machine for a single worker. Our aim is to investigate the full spectrum of granularity, understanding how each impacts performance and efficiency. Thus, the design space is categorized into three main areas: Single-Worker Instance per Machine (`WIM`), Single-Worker Instance per NUMA (`WIN`), and Single-Worker Instance per CPU Core (`WIC`). Based on our findings in Section 2, it is clear that adopting a new category that has a Single-Worker Instance per Chiplet (`WICP`) could be a promising approach. The main differences between these approaches concern the number of database worker instances spawned on a single machine, the way computing resources are allocated to each worker instance, and the data allocation policies. By data allocation policies, we refer specifically to the methodologies employed for distributing and managing data across the machine's memory hierarchy. This includes the initial distribution of data fragments within the system's memory, aiming for optimal performance by considering factors such as NUMA architectures and cache utilization, but also the management of data generated or transferred as queries are executed, specifically during the shuffle or join phases.

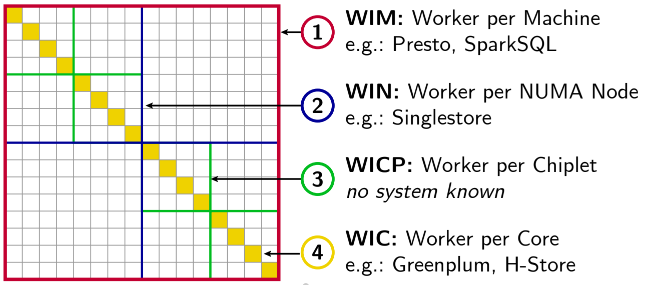


Fig. 6: Different design configurations for distributed query engines in a dual-socket processor diagram.

Different designs affect the degree of parallelism, amount of data to be exchanged and efficient use of NUMA and caches.

(1) `WIM`: Single-Worker Instance per Machine. The Single-Worker Instance per Machine policy is widely adopted by existing distributed query engines [11, 13, 18, 46], such as Presto or SparkSQL. The deployment policy is independent of the topology of the underlying hardware. Basically, a single database worker instance manages all resources, as shown in Figure 9. In addition, there are no restrictions on how memory is accessed. All threads can access the L3 cache within each chiplet, in addition to the local and remote main memory. This means the systems need to manage their resources internally to improve performance.

Advantages: The implementation is not dependent on the physical system architecture. Threads have the entire L3 cache at their disposal to store and access data efficiently. Moreover, with just a single database worker instance for each machine in the cluster, communication between nodes can be minimised [20, 43].

Disadvantages: Maximising the data and threads locality is challenging. Inadequate allocation of data between the chiplets' L3 cache risks congesting interconnects, leading to severe performance reductions. Redesigning the database engine to make it more hardware-aware requires cost and effort, and, in many cases, this is an unviable option. The high variability in systems, as shown in Section 2, can result in inconsistent results.

(2) WIN: Single-Worker Instance per NUMA. In the Single-Worker Instance per NUMA design, there are as many database worker instances as NUMA nodes. The worker instances are placed considering the physical layout of the multiprocessor machine and can only use the computational resources of the NUMA nodes that they are associated with. Communication between instances takes place through the network stack. The OS can migrate threads but only within the NUMA node to which they are bound. In terms of data allocation, the WIN design relies on the Membind policy, which forces each instance to only use its local memory. An example of this design can be found in SingleStore [10]. SingleStore uses each NUMA domain as a separate database worker instance, and its developers suggest using this policy when deploying their database in multi-processor systems. [3].

Advantages: This policy directly addresses some concerns of the previous sections regarding high latency and low bandwidth during inter-core communication, minimizing cross-domain accesses and interconnect congestion. The L3 cache size is large and unpartitioned within each domain, allowing all chiplet cores access for data storage. This can reduce cache constraints and improve data handling for cache-intensive applications. In addition, a NUMA domain typically encompasses multiple chiplets, providing a larger pool of resources. Finally, the work is partitioned by the database engine, thus generating a fair use of the resources shared in NUMA domains.

Disadvantages: As shown in Section 2, communication variability can be significant even within a NUMA domain. An instance may experience higher latency when accessing memory located on a different chiplet within the same NUMA domain. The number of chiplets within can also be considerable, and managing memory across multiple chiplets can be complex. Ensuring optimal cache locality requires sophisticated scheduling, which can add overhead and complexity to system operation. Also, intra-machine communication takes place via the network stack with less bandwidth than what can be achieved through interconnects. In addition, a larger number of database worker instances increases inter-machine communication and may vary the degree of parallelism generated by the database engines.

(3) WICP: Single-Worker Instance per Chiplet. Figure 9 illustrates also the WICP design. This design assigns the computational resources of each chiplet to a distinct database worker instance. Similar to WIN, all instances interact via the network stack, with each one focusing on using the cache and memory that are local to its specific chiplet for optimal performance. As far as we know, this design is not currently adopted by any major distributed database or data processing system.

Advantages: By dedicating a database worker instance to each chiplet, Single-Worker Instance per Chiplet offers a localized approach that can minimize inter-core latency and increase inter-core bandwidth for machines with high heterogeneity in core-to-core communication (see §2.3 and §2.4). With each instance prioritizing local cache and memory, the Single-Worker Instance per Chiplet design eliminates cross-chiplet communication, maximizes data locality and favors a more balanced data allocation thus avoiding possible interconnection congestion.

Disadvantages: Force instances to run on a specific subset of cores can restrict their memory bandwidth. Moreover, forcing a task to use only a single chiplet's cores limits access to other chiplets' L3 caches, shrinking available cache size and necessitating slower main memory use for data exceeding local cache capacity. The high number of chiplets, even within a single NUMA node, could lead to significant overhead in ensuring data consistency and synchronization across them.

(4) WIC: Single-Worker Instance per CPU Core. The Single-Worker Instance per CPU Core design involves using one database worker instance per CPU core and is adopted by systems such as Greenplum [2, 5], and H-Store [32]. The deployment varies according to the number of effective CPUs or cores. A rule of thumb is to use as many database workers as CPU cores. As in the WIN design, all instances exchange data using the network stack. In addition, there are no constraints on data placement strategies, but each instance prioritizes memory local to the core on which it runs.

Advantages: The policy provides improved data and thread placement. It prevents cores from communicating through interconnections and avoids potential congestion. As a result, WIC-based systems exhibit high single-thread performance.

Disadvantages: The Single-Worker Instance per CPU Core design increases the amount of communication required for data processing. In addition, as with WICP, each process is limited to only using the cache available in its own chiplet and thereby relying on the main memory when LLC misses occur.

4 EXPERIMENTAL ANALYSIS

We conduct a range of experiments to comprehend the behavior of different deployment policies for distributed query engines on modern chiplet-based machines (§4.1). Specifically, we answer the following questions.

Q1: What is the impact of deployment policies on the performance of distributed query engines over a modern chiplet-based machine ? (§4.2)

Q2: How do different deployment policies affect a query engine's intra-machine multi-core scalability ? (§4.3)

Q3: What is the trade-off between improving cache locality through the chiplet's local L3 cache and expanding the cache size accessible to the process at the cost of reduced bandwidth ? (§4.4)

Q4: How sensitive are deployment policies regarding different characters (§4.5) including

- the number of worker instances of a query engine (§4.5.1)
- the variety of query workload (§4.5.2)
- the data skewness (§4.5.3)

4.1 Experimental setup

Testbed. The experiments are conducted on a diverse set of chiplet-based machines. This testing environment differs from the one described in §2, as it now includes machines equipped with two NUMA domains. Specifically, we used the following:

(1) AMD EPYC Milan features a dual-socket AMD EPYC 7713 processor with two NUMA domains, each socket being equipped with 64 CPU cores, 512 GB RAM. It runs the OS of Ubuntu 23.04.

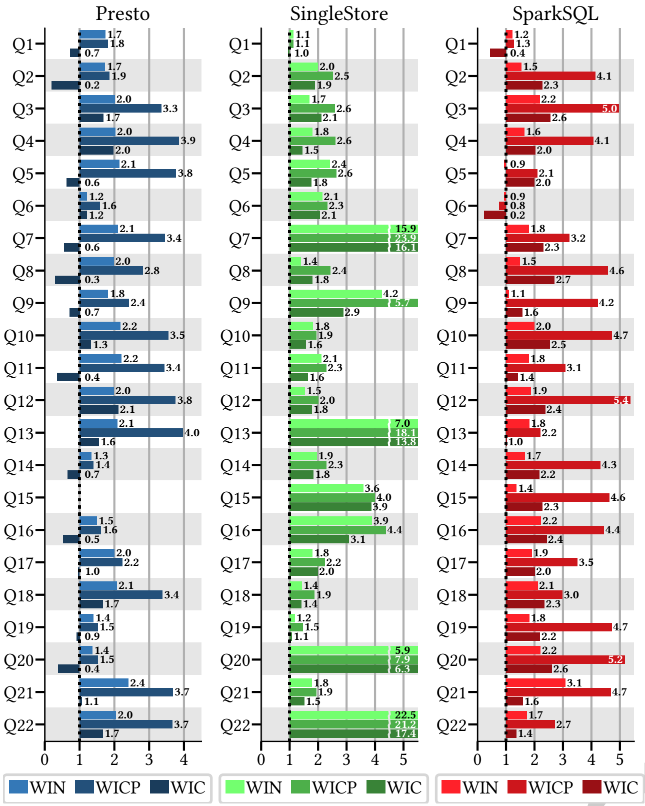


Fig. 7: TPC-H Speedups on AMD EPYC Milan with scale factor 100.

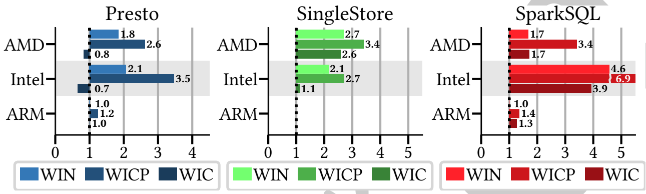


Fig. 8: Geomean Speedups over WIM.

(2) Intel Sapphire Rapids contains a dual-socket Intel Sapphire Rapids Xeon Platinum 8480+, with 112 CPU cores and 512 GB of RAM. It has two NUMA domains running Ubuntu 22.04.

(3) ARM Graviton 3 has a single-socket ARM Graviton 3 processor with one NUMA domain, providing 64 CPU cores and 128 GB of RAM with the OS of Ubuntu 22.04.

Query engines. We use common distributed query engines including Presto [46], SingleStore [24] and SparkSQL [18]. They have the same default deployment policies–*Single-Worker Instance per Machine* (WIM).

Benchmark suites. We select a set of benchmarks including TPC-H [15], TPC-DS [14], JCC-H [23] and the STREAM benchmark [36, 37]. Such benchmarks contain 143 queries and represent various types of workloads, including ad-hoc analytical queries and decision support queries and therefore, ensure a comprehensive evaluation of the distributed query engines on chiplet-based machines.

To assess various deployment policies, we leverage the *libnuma* library [16] to initiate worker instances for thread binding and data placement. In our evaluation, we utilize datasets with a scale factor of 100 (i.e., 100 GB of data size). All reported results are averaged over five runs.

Measures. Our focus is the performance of different deployment policies of query evaluation on chiplet-based machines. To this end, we measure the query execution time of all deployment policies, WIM, WIN, WICP and WIC for each query engine on different machines. For better visualization, we compare speedups of WIN, WICP and WIC over the baseline–the default deployment of WIM.

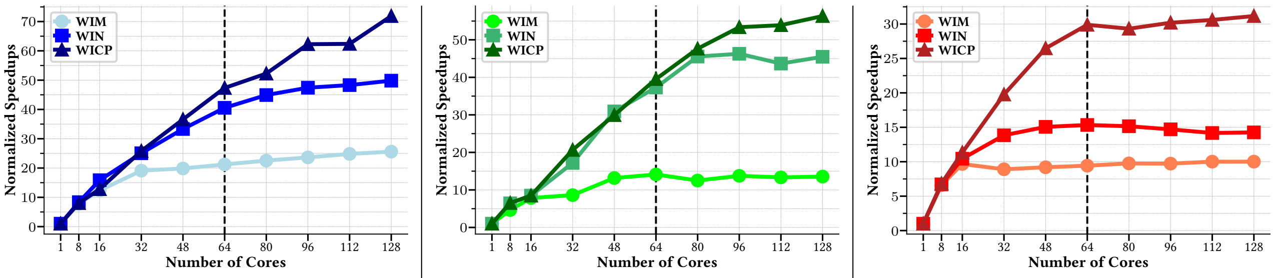
4.2 Deployment policies on chiplet-based machines

We investigate the performance impact of four deployment policies (see §3.2) on the chiplet-based machines. To this end, we evaluate the TPC-H workload using Presto, SingleStore and SparkSQL query engines and compare the speedups of Single-Worker Instance per CPU Core (WIC), Single-Worker Instance per NUMA (WIN) and Single-Worker Instance per Chiplet (WICP) over the default deployment policy, Single-Worker Instance per Machine (WIM). We start by exploring the general trends observed on Intel Sapphire Rapids, ARM Graviton 3 and AMD EPYC Milan machines, then delve into a detailed investigation of results specifically on the latter.

Deployment Efficiency Across Chiplet Architectures. Figure 8 showcases the geometric mean speedup of TPC-H queries across different deployment policies: WIC, WICP, and WIN, with WIM as the baseline. The standout policy, WICP, consistently outperforms others, notably enhancing SparkSQL's performance with significant speedups on AMD EPYC Milan (3.41×), Intel Sapphire Rapids (6.97×), and ARM Graviton 3 (1.36×) processors. SparkSQL also shows performance gains with the WIC policy, showing improved efficiency across the board: 1.71× on AMD, 3.92× on Intel, and 1.27× on ARM. The reason is that SparkSQL unevenly allocates data between memory and caches with the WIM policy. This imbalance leads to higher LLC miss rates and interconnect congestion, hindering efficiency. WICP mitigate these issues by evenly spreading data among database worker instances, enhancing data locality and overall performance.

SingleStore and Presto also see benefits under the WICP policy, with respective speedups of 2.64× and 3.40× on AMD EPYC Milan, and 3.64× and 2.70× on Intel Sapphire Rapids. This suggests that even though Intel Sapphire Rapids offers better inter-core communication than AMD EPYC Milan, as outlined in Section 2, achieving peak performance still demands a resource allocation strategy that takes the processor's topology into account. In addition, Presto does not perform as well with the WIC policy, experiencing a drop in performance (0.84× on AMD and 0.65× on Intel). This occurs because, in the WIC policy, the database worker instance responsible for coordinating the database worker instances, is restricted to using just one core, thereby slowing down the overall execution speed during aggregations.

The ARM Graviton 3 processor, unsupported by SingleStore, shows limited improvements compared to AMD and Intel. This results from its architecture, which includes a single computational chiplet, mirroring the design of monolithic processors. Moreover, on the ARM Graviton 3, WIN and WIM represent identical deployment



(a) Presto **(b)** SingleStore **(c)** SparkSQL

Fig. 9: Multicore Scalability on AMD Epyc Milan.

strategies due to the single-NUMA processor configuration of the CPU.

TPC-H Queries Analisys. We also note varied performance across individual queries as shown in Fig. 7. We now focus on a detailed investigation of all queries on each evaluated system.

(1) SingleStore. SingleStore's remarkable performance lies in specialized query plans under the WICP policy, which significantly enhance data processing speeds by optimizing data locality for table scans and employing parallelizable and distributable query operators (e.g., Bloom filters and hash joins) over conventional ones.

Specifically, Q7, Q9, Q13, Q15, Q16, Q20, and Q22 are identified as the most efficient queries, with Q13, Q15, Q16, Q20, and Q22 sharing a uniform query plan across all deployment scenarios that utilizes TableScan and Repartition operations. We focus on Q13 because of its extended execution time and substantial transient memory usage [26]. Notably, under WICP policy, Q13's Orders table scan is significantly faster, taking only 0.4 seconds, as opposed to 5.36 seconds with the WIM policy. Similarly, Repartition operations are expedited, completing in 0.25 seconds under WICP compared to 5.06 seconds under WIM. This enhanced efficiency is attributed to WICP's superior data locality with SingleStore, optimizing performance by minimizing data access times and improving overall query execution speed. Consequently, Q13, Q20, and Q22 exhibit better performance over Presto and SparkSQL, with speedup of 18.13×, 7.943× and 21.16× respectively.

SingleStore also outperforms Presto and SparkSQL in executing Q7 and Q9, achieving speedups of 23.9× and 5.7× under the WICP policy, respectively. This performance leap is attributed to the adoption of specialized query plans in the WICP, WIN, and WIC policies that favor bloom filters and hash joins over the conventional nested loop joins used in the WIM policy. The SingleStore coordinator opts for Bloom filters and hash joins as they are easily parallelizable and distributable across several worker instances, significantly cutting down the data transfer volume and execution time.

(2) Presto. In contrast to SingleStore, Presto uses a consistent query plan across all policies, but the number of data fragments it generates depends on the number of worker instances and the amount of computing resources assigned. As the number of workers increases, the number of smaller fragments also goes up, improving how well they fit into the L3 cache and boosting data locality. The query engine's efficiency is particularly evident in the execution of the ScanFilterTable operator, which dominates Presto's query execution time. By refining this process, Presto significantly reduces execution times, leading to superior performance across all benchmark queries. In fact, Q5 and Q18 show the best speedup against SingleStore and SparkSQL, as they spend respectively 88% and 96% of their execution time on the ScanFilter operation on Orders. Q1, Q2, Q6, Q14, Q16, Q19 and Q20 exhibit less than a 2× speedup because they don't involve the Orders table.

Interestingly, despite the LineItem table being larger than the Orders table in the TPC-H benchmark, similar performance gains are not observed. This discrepancy arises because, during scanning, Presto conducts a memory-intensive, sequentially executed predicate check phase, which is the most time-consuming aspect of the ScanFilter execution. Presto bypasses this predicate check for the LineItem table but not for the Orders table. Consequently, queries utilizing the Orders table (e.g., Q5 and Q18, but also Q3, Q4, Q7, Q10, Q12, Q13, Q21 and Q22) exhibit superior performance compared to those using the LineItem table (e.g., Q1, Q6, Q14, Q19 and Q20). We believe this decision is made for performance reasons, particularly due to the LineItem table's large size.

(3) SparkSQL. Unlike SingleStore and Presto, SparkSQL is particularly affected by the negative effects of unbalanced data distribution. Fig. 7 shows that SparkSQL significantly outperforms Presto and SingleStore on queries Q2, Q3, Q8, Q10, Q12, Q14, Q15, Q17, Q19, and Q21, achieving speed improvements ranging from 3.51 to 5.15 times in the WICP deployment over WIM. This notable performance boost is primarily attributed to the inefficient handling of data by SparkSQL under the WIM configuration, where it tends to distribute data unevenly across chiplets. This results in excessive communication between chiplets and congestion in the interconnect, which was observed to be congested for an average of 34% of the query execution time.

Such congestion not only hampers the data scanning process but also impacts the join operations, particularly because SparkSQL relies on SortMerge Join which necessitates memory access for sorting. The WICP deployment strategy effectively mitigates these issues by forcing the use of local cache and memory, which ensures a balanced distribution of data, thereby enhancing performance.

Insights. *Users should always deploy the query engines on chiplet-based machines using* WICP *for the best performance.* WICP *enhances data locality, boosting the efficiency of queries involving extensive data scanning or shuffling. It supports specialized query plans with easily parallelizable operators, thus reducing execution times. Additionally,* WICP *effectively addresses the issue of uneven data allocation in processor caches, eliminating interconnect congestion.*

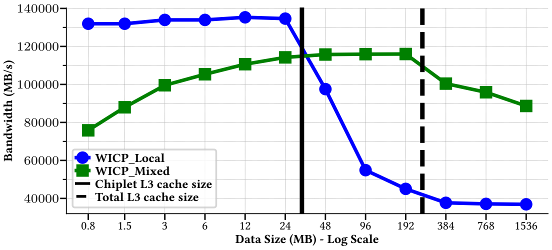


Fig. 10: AMD Milan: STREAM Benchmark with 8 cores (1 Chiplet).

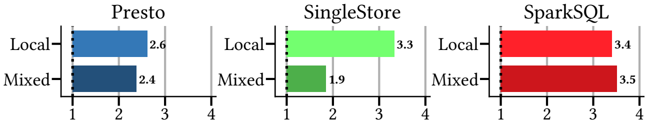


Fig. 11: Geomean Speedups over WIM on AMD EPYC Milan.

Tab. 2: Peak memory usage of selected TPC-H queries.

Query ID	Presto (GB)	SingleStore (GB)	SparkSQL (GB)
2	2.16	0.02	2.10
4	2.20	0.02	9.10
5	17.65	0.29	3.40
7	9.13	0.32	3.40
8	17.86	0.05	10.10
9	26.24	0.44	9.10
10	3.80	0.49	2.60
11	0.10	0.00	1.40
17	1.71	0.32	6.80
18	16.58	0.53	4.30
19	0.51	0.02	0.43
21	32.80	0.30	12.60
Geomean	**4.48**	**0.09**	**3.87**

4.3 Multi-core scalability

Next, we examine the scalability of different deployment policies on a chiplet-based AMD EPYC Milan machine. We evaluate TPC-H using three query engines (i.e., Presto, SingleStore and SparkSQL) and measure their speedups ranging the core counts from 1 to 128, as demonstrated in Fig. 9.

WICP scales the best for all three query engines, achieving near-linear scalability up to 64 cores (i.e., the number of cores in a single NUMA domain). The performance keeps improving beyond 64 cores but at a lower rate, reaching speedups of 71× (Presto), 55× (SingleStore), and 31× (SparkSQL) at 128 cores.

WIN also presents scalability up to 64 cores, but its performance increases slower than WICP. With more cores, WIN's scalability flattens. In contrast, WICP maintains increasing performance speedups, outperforming WIN by 2.15× for SparkSQL (Fig. 9c), 1.44× for Presto (Fig. 9a) and 1.25× for SingleStore (Fig. 9b). This superior scalability of WICP, especially in SparkSQL, is attributed to improved cache utilization within chiplets and minimized interconnect congestion. Additionally, the high inter-core communication variability even within a single NUMA domain on the AMD EPYC Milan machine, discussed in Section 2, further reduces WIN's scalability.

As expected, WIM demonstrates the worst performance. For all three query engines, WIM only scales to 16 cores, corresponding to the combined core count of just two chiplets. Beyond this, WIM's scalability significantly diminishes and reaches a plateau.

4.4 Chiplet L3 cache efficacy

For the WICP deployment policy, we further investigate the trade-off between only fetching data from a chiplet's local cache (coined WICP_Local) and also fetching from other chiplets' caches (coined WICP_Mixed). As explained in §2, each chiplet is equipped with its own local L3 cache to speed up the data access, while fetching data from other chiplets incurs inter-chiplet communication overhead. WICP_Local enforces fast data access on the local L3 cache to exploit the locality, while WICP_Mixed exploits the capacity of L3 caches of all chiplets but bears the overhead of inter-chiplet communication.

In our setup, WICP_Local uses CPU cores within a single chiplet while WICP_Mixed uses the same number of cores but evenly distributed across chiplets. We assess the impact of these two deployment policies when running the STREAM benchmark on a single socket of an AMD Epyc Milan processor. Fig. 10 demonstrates their aggregated bandwidth for when increasing the size of an array from 0.8 MB to 1536 MB. The bandwidth is lower than previously shown in Fig. 5a §2.5, due to the reduced number of cores used.

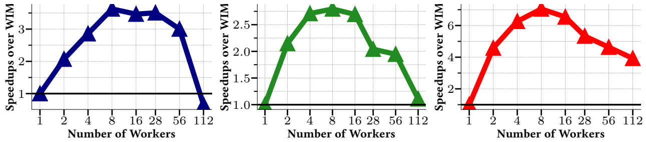
WICP_Local presents higher bandwidth than WICP_Mixed until the array size increases to 32 MB, the single chiplet's L3 cache capacity. This is because WICP_Local avoids inter-chiplet communication. However, for larger array sizes, WICP_Local's bandwidth suddenly drops, as it needs to fetch the data from the main memory. In contrast, WICP_Mixed shows more stable bandwidth due to its ability to leverage the total capacity of L3 caches from all chiplets, 256 MB. Beyond this size, WICP_Mixed's bandwidth also declines.

We further examine WICP_Local's and WICP_Mixed's impact on query evaluation performance using Presto, Singlestore, and SparkSQL over the TPC-H benchmark on the same AMD EPYC Milan processor. To this end, we compare their geomean speedups over the default WIM deployment, as illustrated in Fig. 11. We observe the performance speedups for both WICP_Local and WICP_Mixed. But different query engines show very different impacts.

For SingleStore, WICP_Local achieves much higher speedups than WICP_Mixed, 3.32× vs 1.85×. In the same vein, Presto presents the preference for WICP_Local over WICP_Mixed, but the gap is smaller, 2.61× vs 2.38×. However, SparkSQL shows a different trend–WICP_Mixed (3.40×) slightly outperforms WICP_Local (3.32×).

Such different behaviors stem from how these query engines manage the data movement during the shuffle and join. Tab. 2 shows the peak memory footprints when executing of the most relevant queries for each query engine (TPC-H with scale factor 50). SingleStore stands out as the only system that transfers data within the capacity of an L3 cache (i.e., WICP_Local performs much better). In contrast, both Presto and SparkSQL incur larger data movement beyond the combined L3 cache capacity, resulting in data fetching from the main memory. Most of the execution time is spent on accessing memory and therefore, the performance gap between WICP_Local and WICP_Mixed becomes small.

Insights. *Users should use* WICP_Local *deployment policy when the amount of accessed data is below the capacity of a single chiplet's L3 cache. When the amount of data is larger than a single local L3 cache size but is smaller than the size of combined L3 caches, users should use the* WICP_Mixed *deployment policy. When accessing a very large amount of data (i.e., cache misses happen frequently), the performances of* WICP_Local *and* WICP_Mixed *are similar.*



(a) Presto **(b)** SingleStore **(c)** SparkSQL

Fig. 12: Varying the number of database worker instances on a dual-socket Intel Sapphire Rapids. Geomean speedup using TPC-H with scale factor 100.

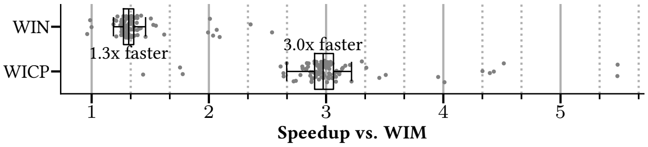


Fig. 13: Presto: TPC-DS on AMD EPYC Milan

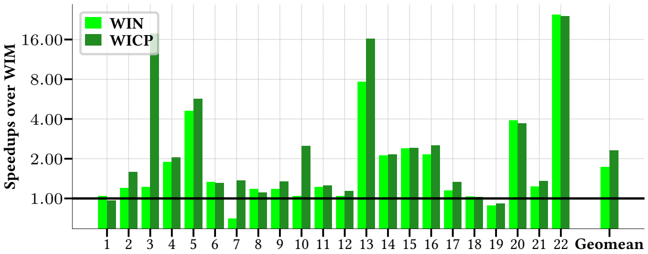


Fig. 14: SingleStore: JCC-H on AMD EPYC Milan

4.5 Sensitivity analysis

In this section, we showcase the sensitivity and real-world applicability of our findings. We ensure their validity across diverse workloads and configurations, illustrating that their applicability persists.

4.5.1 ***Number of worker instances*.** We first explore how the number of worker instances affects query performance of WICP on a chipet-based machine i.e., Intel Saphhire Rapids (eight chiplets, 112 cores). To this end, we control the number of worker instances n, each worker instance being assigned $\lceil 112/n \rceil$ cores. Whenever possible, we map cores to a worker instance from a single chiplet and NUMA domain. We measure the speedup of WICP over the default deployment policy, WIM for all three query engines.

Fig. 12 illustrates the geometric mean of speedups on TPC-H. Here, Presto (Fig. 12a), SingleStore (Fig. 12b), and SparkSQL (Fig. 12c) present the same trend that the speedup keeps increasing to eight worker instances, reaching 3.62×, 2.75× and 7.04×, respectively. Their speedups plateau at eight worker instances and start to deteriorate from 28 worker instances. Note that Intel Saphhire Rapids has eight chiplets. This means query engines achieve ideal balance of computation and communication granularity when the number of worker instances equals the number of chiplets.

Insights. *It improves the query engine performance to deploy with increasing worker instances on chiplet-based machines. The sweet spot of the number of worker instances is the number of chiplets.*

4.5.2 ***Query variety*.** Next, we investigate how the diversity of queries affects the performance of query engines on a chiplet-based machine i.e., AMD EPYC Milan. To this end, we use SingleStore to evaluate the TPC-DS workload because its queries are more various than TPC-H. For each query, we measure the performance speedups of WICP and WIN over the default WIM, as illustrated in Fig. 13.

We observe little impact of the query diversity on query performance. All queries present speedups with the WICP and WIN deployment policies. Queries 1, 21, 22, 37, 39, 81, 83, 84 and 91 demonstrate the highest speedups, because they spend most of the execution time in scanning large tables (e.g., the inventory table) which benefit from WICP (see 4.2). By examining the execution plans we found that such scan phases cause expensive remote memory accesses in WIM. In contrast, WICP avoids these remote accesses by enforcing workers to use local cache.

Insights. *Queries involving extensive data scans or significant data shuffling benefit from the enhanced locality that* WICP *can provide. Overall, all queries see improvements with* WICP*, even on complex benchmark suites like TPC-DS.*

4.5.3 ***Data skew*.** Finally, we explore the impact of skewed workload on the performance of deployment policies. To achieve this, we use SingleStore to evaluate the JCC-H benchmark [23] which introduces the join-crossing-correlations and injects skewness into its dataset and query workload. We measure the speedups of WICP and WIN over the default WIM.

Fig. 14 shows the speedups for individual queries and their geometric mean speedups. Here, we observe that skewed workload affects the performance of the WICP. The geometric mean speedups for WICP and WIN are 2.31× and 1.72×, compared to TPC-H's 3.40× and 2.68×. The reason is that data skewness challenges the efficiency of WICP by leading to imbalanced workload distribution and underutilization of optimized data locality, without the issue of cache contention that is specific to WIN. Specifically, data skewness implies that certain data items are accessed more frequently than others. With the WIN policy, this frequent access to specific data items leads to increased cross-chiplet communication and thereby interconnects congestion. In contrast, while the WICP policy eliminates cross-chiplet communication, it encounters limitations when the data exceeds the capacity of a chiplet's smaller cache or is absent from it, necessitating reliance on slower main memory accesses. In addition, depending on the system, to manage or mitigate the effects of data skewness, there might be a need for more frequent data exchanges, even through the network stack, increasing the overhead from inter-chiplet communication. This communication can reduce the benefits of local optimization, as chiplets spend more time exchanging data than processing it.

Insights. *The skewness impacts how effectively the system can leverage its hardware topology awareness, resulting in diminished performance gains with* WICP *in the presence of skewed data distributions. This is primarily due to the increased need for cross-chiplet communication, resulting in interconnect congestion, and data redistribution to address skewness. However,* WICP *still performs better than* WIN *for most queries.*

5 DISCUSSION

The Chiplet Age. The trend towards chiplet-based processors is a clear sign of a significant shift in the world of computing. The key factor behind this movement is the flexibility that chiplets offer. Manufacturers can mix and match different components like CPUs, GPUs, and memory to create configurations that meet specific performance needs for a variety of applications. The introduction of standards like the Universal Chiplet Interconnect Express (UCIe) further cements the future of chiplets in the industry. UCIe standardizes the interconnects between different chiplets, facilitating interoperability between components made by various manufacturers. This standardization is critical as it lowers barriers and speeds up the adoption of chiplet technology.

Another exciting development is the integration of High Bandwidth Memory (HBM) into chiplet designs, like those used in Intel's processors. This integration significantly increases memory bandwidth, making chiplet-based processors especially effective for data-intensive tasks. The trend towards integrating various memory types onto the CPU package promises further enhancements in performance. In summary, the increasing use of chiplets in various areas is a clear indicator of their growing role in the future of computing, a trend backed by their versatility, standardization efforts, performance benefits, and the need to overcome the limitations of traditional architectures.

NUMA Domain Configuration. Non-Uniform Memory Access (NUMA) is an integral part of modern multiprocessing systems, especially in the context of high-performance computing. AMD and Intel, have developed specific features that can be enabled by changing BIOS configurations to enhance NUMA configurations: AMD's NUMA Per Socket (NPS) and Intel's Sub-NUMA Clustering (SNC). AMD's NPS, in EPYC processors, divides sockets into multiple NUMA nodes, each with its own memory, optimizing memory bandwidth and latency. On the other hand, Intel's SNC, starting from Xeon Scalable processors, splits processor resources into smaller NUMA domains, focusing on cache and memory efficiency to reduce latency.

These features aim to boost performance by improving memory and cache utilization, especially in demanding applications like large-scale databases and data analytics. However, this is not a complete solution since many systems, like Presto or SparkSQL, do not exploit multiple NUMA domains, leading to inefficiencies. In conclusion, these innovative features, while reflecting a shift towards smarter and topology-aware resource allocation and potentially marking significant advancements, can only realize their full potential when paired with software that is equally attuned to the nuances of the hardware it runs on.

Inter-machine Scalability. When discussing distributed query engines, it's essential to recognize that they are designed for multi-machine environments. We actually exploit this capability when applying the `WICP` deployment design and leave all the effort in managing the partitioning and distribution of data, along with the creation of the appropriate query plan, to the query execution engine. Compared to the traditional `WIM` design, `WICP` employs significantly more worker instances, leading to increased communication overhead during query processing. Our hypothesis is that we will observe linear horizontal scalability as we increase the number of machines until communication and resource limitations intervene. Integrating multiple high-performance network cards, supporting over 200 Gbps, with the database's worker instances during setup can mitigate this bottleneck requiring no source code changes.

Furthermore, when comparing `WICP` and `WIM` deployments, the single-machine scenario is the least favorable for `WICP` due to its need for inter-worker communication, unlike `WIM` which requires none. However, as more machines are added, the performance gap between `WICP` and `WIM` decreases. For instance, with a two-chiplet machine, `WIM` needs no communication, while `WICP` requires 50% communication for processing. With two machines, the gap narrows further: `WIM`'s communication need rises to 50%, whereas `WICP`'s increases to 75%. For this reason, we believe that speedup improves as the number of machines increases.

6 RELATED WORK

Research on memory allocation and data partitioning highlights varied strategies' impacts on multiprocessor platforms, with studies like Yang et al. [48] advocating shared resource distribution. However, these approaches may not fully align with the needs of chiplet-based architectures. While Psaroudakis et al. [45] emphasize physical partitioning within NUMA domains for SAP HANA, their findings do not directly apply to chiplet systems, indicating a necessity for architecture-specific strategies.

Some query engines, like ERIS by Kissinger et al. [34], are designed with the physical CPU layout in mind, specifically targeting NUMA architectures but not chiplet-based processors. ERIS optimizes for NUMA by using adaptive partitioning to mitigate synchronization overheads and improve performance, yet does not extend its focus to chiplet systems.

Container resource allocation is crucial for cloud services, with research focusing on energy efficiency [21, 29, 47] and resource allocation models based on priority [25, 42]. However, these studies overlook optimization for chiplet-based systems. Jebalia et al. [31] introduce a neural network and genetic algorithm approach for resource maximization, effective in low contention scenarios.

7 CONCLUSION

Our work focuses on the critical influence of chiplet-based CPU architectures on the performance of multi-core query engines, a timely study given the recent adoption of chiplet designs by leading processor manufacturers including AMD, Intel, and ARM. We propose a chiplet-granularity deployment strategy for query engines, optimizing task allocation to exploit the architectural benefits of chiplets, such as improved scalability and cache locality. This approach significantly enhances query performance, demonstrating the importance of adapting query engine designs to the evolving landscape of CPU architecture.

Our proposed guidelines directly apply to on-premise distributed query engines, enabling them to maximize their resource utilization without requiring significant changes to the engine. In the case of engines deployed in the cloud, our insights may enable better future provisioning and assignment of hardware resources. Overall, we conclude that query engines need to rethink resource allocation strategies to fully exploit the underlying hardware resources.

REFERENCES

[1] 2023. Apache Hadoop. https://hadoop.apache.org.. Accessed: 2023-6-19.

[2] 2023. Introduction to Greenplum. https://docs.greenplum.org/6-10/install_guide/preinstall_concepts.html. Accessed: 2023-6-19.

[3] 2023. SingleStore Documentation. https://docs.singlestore.com/v7.3/introduction/documentation-overview/. Accessed: 2023-6-19.

[4] 2023. Teradata online documentation. https://docs.teradata.com/r/Teradata-VantageTM-SQL-Fundamentals/June-2022/Introduction-to-SQL-Fundamentals. Accessed: 2023-6-19.

[5] 2023. VMware Greenplum 6.24 documentation. https://docs.greenplum.org/6-12/common/gpdb-features.html. Accessed: 2023-6-19.

[6] 2024. https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-clusters.html. Accessed: 2023-6-19.

[7] 2024. https://www.oracle.com/technetwork/database/exadata/exadata-x8-2-ds-5444350.pdf. Accessed: 2023-6-19.

[8] 2024. Amazon Athena. https://docs.aws.amazon.com/whitepapers/latest/big-data-analytics-options/amazon-athena.html. Accessed: 2023-6-19.

[9] 2024. Analyzing Unconventional Logic Semiconductors – A Shift Away from Semiconductor Manufacturers. https://hacarus.com/ai-lab/03312022-graviton3/. Accessed: 2024-3-1.

[10] 2024. Configuring NUMA for SingleStore. https://support.singlestore.com/hc/en-us/articles/360058633252-Configuring-NUMA-for-SingleStore. Accessed: 2024-3-1.

[11] 2024. MySQL Documentation. https://dev.mysql.com/doc/. Accessed: 2024-3-1.

[12] 2024. NUMA Balancing. https://www.kernel.org/doc/html/latest/admin-guide/sysctl/kernel.html#numa-balancing. Accessed: 2023-6-19.

[13] 2024. Overview of ClickHouse Architecture. https://clickhouse.com/docs/en/development/architecture. Accessed: 2024-3-1.

[14] 2024. *TPC Benchmark™ DS*. Accessed: 2024-3-1.

[15] 2024. *TPC Benchmark™ H*. Accessed: 2024-3-1.

[16] Kleen A. 2005. A NUMA API for Linux.

[17] Josep Aguilar-Saborit, Raghu Ramakrishnan, Krish Srinivasan, Kevin Bocksrocker, Ioannis Alagiannis, Mahadevan Sankara, Moe Shafiei, Jose Blakeley, Girish Dasarathy, Sumeet Dash, Lazar Davidovic, Maja Damjanic, Slobodan Djunic, Nemanja Djurkic, Charles Feddersen, Cesar Galindo-Legaria, Alan Halverson, Milana Kovacevic, Nikola Kicovic, Goran Lukic, Djordje Maksimovic, Ana Manic, Nikola Markovic, Bosko Mihic, Ugljesa Milic, Marko Milojevic, Tapas Nayak, Milan Potocnik, Milos Radic, Bozidar Radivojevic, Srikumar Rangarajan, Milan Ruzic, Milan Simic, Marko Sosic, Igor Stanko, Maja Stikic, Sasa Stanojkov, Vukasin Stefanovic, Milos Sukovic, Aleksandar Tomic, Dragan Tomic, Steve Toscano, Djordje Trifunovic, Veljko Vasic, Tomer Verona, Aleksandar Vujic, Nikola Vujic, Marko Vukovic, and Marko Zivanovic. 2020. POLARIS: The Distributed SQL Engine in Azure Synapse. *Proceedings VLDB Endowment* 13, 12 (Aug. 2020), 3204–3216.

[18] Michael Armbrust, Reynold Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei A. Zaharia. 2015. Spark SQL: Relational Data Processing in Spark. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015). https://api.semanticscholar.org/CorpusID:2560213

[19] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chainani, Kiran Chinta, Venkatraman Govindaraju, Todd J Green, Monish Gupta, Sebastian Hillig, Eric Hotinger, Yan Leshinksy, Jintian Liang, Michael McCreedy, Fabian Nagel, Ippokratis Pandis, Panos Parchas, Rahul Pathak, Orestis Polychroniou, Foyzur Rahman, Gaurav Saxena, Gokul Soundararajan, Sriram Subramanian, and Doug Terry. 2022. Amazon Redshift Re-invented. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia PA USA). ACM, New York, NY, USA.

[20] Claude Barthels, Ingo Müller, Timo Schneider, Gustavo Alonso, and Torsten Hoefler. 2017. Distributed join algorithms on thousands of cores. *Proceedings VLDB Endowment* 10, 5 (Jan. 2017), 517–528.

[21] Belén Bermejo, Carlos Guerrero, Isaac Lera, and Carlos Juiz. 2016. Cloud Resource Management to Improve Energy Efficiency Based on Local Nodes Optimizations. In *ANT/SEIT*.

[22] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. 2015. The end of slow networks: It's time for a redesign. (April 2015). arXiv:1504.01048 [cs.DB]

[23] Peter Boncz, Angelos-Christos Anatiotis, and Steffen Kläbe. 2018. JCC-H: Adding join crossing correlations with skew to TPC-H. In *Performance Evaluation and Benchmarking for the Analytics Era*. Springer International Publishing, Cham, 103–119.

[24] Jack Chen, Samir Jindel, Robert Walzer, Rajkumar Sen, Nika Jimsheleishvilli, and Michael Andrews. 2016. The MemSQL query optimizer. *Proceedings VLDB Endowment* 9, 13 (Sept. 2016), 1401–1412.

[25] K. Dinesh, G R Poornima, and K. Kiruthika. 2012. Efficient Resources Allocation for Different Jobs in Cloud. *International Journal of Computer Applications* 56 (2012), 30–35.

[26] Markus Dreseler, Martin Boissier, Tilmann Rabl, and Matthias Uflacker. 2020. Quantifying TPC-H choke points and their optimizations. *Proceedings of the VLDB Endowment* 13 (2020), 1206 – 1220.

[27] Sérgio Fernandes and Jorge Bernardino. 2014. What is BigQuery?. In *Proceedings of the 19th International Database Engineering & Applications Symposium on - IDEAS '15* (Yokohama, Japan). ACM Press, New York, New York, USA.

[28] Fabien Gaud, Baptiste Lepers, Justin R. Funston, Mohammad Dashti, Alexandra Fedorova, Vivien Quéma, Renaud Lachaize, and Mark Roth. 2015. Challenges of memory management on modern NUMA systems. *Commun. ACM* 58 (2015), 59 – 66.

[29] Guilherme Arthur Geronimo, Jorge Werner, Carla Merkle Westphall, and Leonardo Defenti. 2013. Provisioning and Resource Allocation for Green Clouds.

[30] Anurag Gupta, Deepak Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. 2015. Amazon redshift and the case for simpler data warehouses. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne Victoria Australia). ACM, New York, NY, USA.

[31] Maha Jebalia, Asma BEN LETAIFA, Mohamed Hamdi, and Sami Tabbane. 2013. A Comparative Study on Game Theoretic Approaches for Resource Allocation in Cloud Computing Architectures. *2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (2013), 336–341.

[32] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, and Daniel J. Abadi. 2008. H-Store: A High-Performance, Distributed Main Memory Transaction Processing System. *Proc. VLDB Endow.* 1, 2 (aug 2008), 1496–1499. https://doi.org/10.14778/1454159.1454211

[33] T Kiefer, B Schlegel, and W Lehner. 2013. *Experimental Evaluation of NUMA Effects on Database Management Systems.* BTW.

[34] Thomas Kissinger, Tim Kiefer, Benjamin Schlegel, Dirk Habich, Daniel Molka, and Wolfgang Lehner. 2014. ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workload. In *ADMS@VLDB*.

[35] Michael Mattioli. 2021. Rome to Milan, AMD Continues Its Tour of Italy. *IEEE Micro* 41, 4 (2021), 78–83. https://doi.org/10.1109/MM.2021.3086541

[36] John D. McCalpin. 1991-2007. *STREAM: Sustainable Memory Bandwidth in High Performance Computers.* Technical Report. University of Virginia, Charlottesville, Virginia. http://www.cs.virginia.edu/stream/ A continually updated technical report. http://www.cs.virginia.edu/stream/.

[37] John D. McCalpin. 1995. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), 19–25.

[38] John D. McCalpin. 2023. Bandwidth Limits in the Intel Xeon Max (Sapphire Rapids with HBM) Processors. In *ISC Workshops*. https://api.semanticscholar.org/CorpusID:261344204

[39] Gabe Mounce, James Lyke, Stephen J. Horan, R. Doyle, Raphael R. Some, and Wesley A. Powell. 2016. Chiplet based approach for heterogeneous processing and packaging architectures. *2016 IEEE Aerospace Conference* (2016), 1–12.

[40] Samuel D. Naffziger, Noah Beck, Thomas D. Burd, Kevin M. Lepak, Gabriel H. Loh, Mahesh Subramony, and Sean White. 2021. Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product. *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)* (2021), 57–70. https://api.semanticscholar.org/CorpusID:235415451

[41] Nevine Nassif, Ashley Munch, Carleton L. Molnar, Gerald Pasdast, Sitaraman V. Lyer, Zibing Yang, Oscar Mendoza, Mark Huddart, Srikrishnan Venkataraman, Sireesha Kandula, Rafi Marom, Alexander M. Kern, William J. Bowhill, David Mulvihill, Srikanth Nimmagadda, Varma Kalidindi, Jonathan Krause, Mohammad MinHazul Haq, Roopali Sharma, and Kevin Duda. 2022. Sapphire Rapids: The Next-Generation Intel Xeon Scalable Processor. *2022 IEEE International Solid-State Circuits Conference (ISSCC)* 65 (2022), 44–46. https://api.semanticscholar.org/CorpusID:247523158

[42] Chandrashekhar S. Pawar and Rajnikant Bhagwan Wagh. 2013. Priority based dynamic resource allocation in Cloud computing with modified waiting queue. *2013 International Conference on Intelligent Systems and Signal Processing (ISSP)* (2013), 311–316.

[43] Orestis Polychroniou, Wangda Zhang, and Kenneth A. Ross. 2018. Distributed Joins and Data Placement for Minimal Network Traffic. *ACM Trans. Database Syst.* 43 (2018), 14:1–14:45. https://api.semanticscholar.org/CorpusID:56594570

[44] Danica Porobic, Ippokratis Pandis, Miguel Branco, Pınar Tözün, and Anastasia Ailamaki. 2012. OLTP on Hardware Islands. *Proc. VLDB Endow.* 5 (2012), 1447–1458.

[45] Iraklis Psaroudakis, Tobias Scheuer, Norman May, Abdelkader Sellami, and Anastasia Ailamaki. 2015. Scaling Up Concurrent Main-Memory Column-Store Scans: Towards Adaptive NUMA-aware Data and Task Placement. *Proc. VLDB Endow.* 8 (2015), 1442–1453.

[46] Raghav Sethi, Martin Traverso, Dain Sundstrom, Dave Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, and Christopher Berner. 2019. Presto: SQL on Everything. *2019 IEEE 35th International Conference on Data Engineering (ICDE)* (2019), 1802–1813. https://api.semanticscholar.org/CorpusID:174820394

[47] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. 2008. Energy aware consolidation for cloud computing. In *IEEE International Conference on Cluster Computing*.

[48] Maolin Yang, Wen-Hung Huang, and Jian-Jia Chen. 2019. Resource-Oriented Partitioning for Multiprocessor Systems with Shared Resources. *IEEE Trans. Comput.* 68 (2019), 882–898.

[49] Shiliang Zhu, Min Miao, Zhuanzhuan Zhang, and Xiaolong Duan. 2022. Research on A Chiplet-based DSA (Domain-Specific Architectures) Scalable Convolutional Acceleration Architecture. *2022 23rd International Conference on Electronic Packaging Technology (ICEPT)* (2022), 1–6.