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 dies on a single package, allow hardware to scale to higher CPU core counts at the cost of more memory heterogeneity and performance variability. This introduces challenges when existing query engines are deployed on chiplet-based CPUs, as current designs make assumptions about uniform memory access, cache locality and consistent core performance, e.g., leading to ineffective CPU utilization.

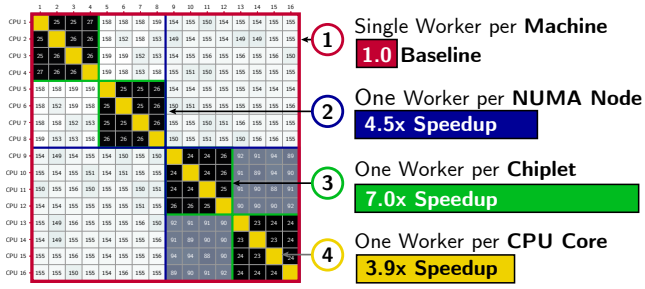
In this paper, we analyse the performance impact when query engines ignore chiplet-specific properties. We demonstrate that a naïve deployment can result in a significant degradation of query processing efficiency, exhibiting non-linear scaling even within a single CPU socket domain. Based on comprehensive experiments, we explore approaches to deploy query engines on chiplet-based CPUs with improved performance: we show that distributing processing tasks according to a chiplet-aware strategy achieves higher resource utilization and scalability, yielding an up to 7× speedup 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, 17(11): 3428 - 3441, 2024.
doi:10.14778/3681954.3682011

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Alessandro727/OLAP-on-Modern-Chiplet-Based-CPUs.

1 INTRODUCTION

Chiplet-based CPUs are a new technology adopted by most major processor manufacturers. They are made up of multiple smaller chips, known as *chiplets*, which are interconnected with a high-bandwidth fabric to operate seamlessly as a single multi-core CPU [8, 49, 54, 56]. This modular approach allows for better yields during manufacturing, offering a practical solution to scale to higher CPU core counts. Chiplet-based designs also support individual chiplets to be tailored to specific functions, from computation to memory access, thus enhancing the performance of specific workloads [53, 65].



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

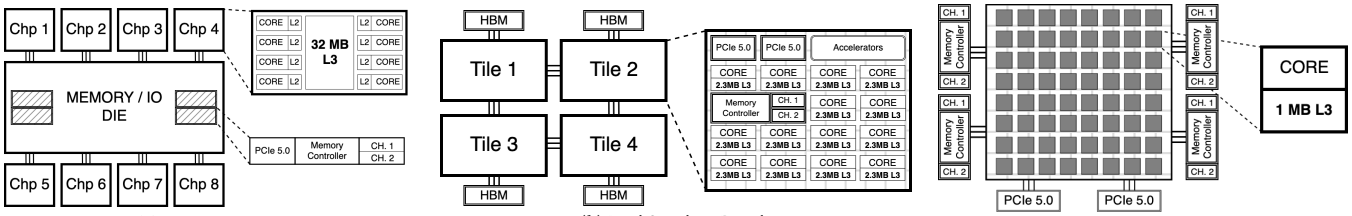
Figure 1: Deployment strategies on a chiplet-based CPU

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 (2) inter-core latencies and (3) bandwidths. For example, Fig. 1a shows the core-to-core latency of a dual-socket CPU (each socket is enclosed in a blue box). In this scenario, inter-core latencies can vary by up to 6× within the same CPU socket domain. As shown experimentally in this paper, this heterogeneity of chiplet-based CPUs impacts the performance of multi-core query engines, going beyond the effects introduced by non-uniform memory accesses (NUMA) [18, 40, 44, 58].

So far, little attention has been paid to the question *how to deploy query engines efficiently on chiplet-based CPUs*. Existing commercial designs of multi-core query engines employ different strategies to map worker instances (tasks) to CPU cores (see Fig. 1a): Redshift [7, 28] and Greenplum [2] (colored in yellow) partition resources into slices, consisting of individual cores, and assign tasks to these slices; SingleStore [34] (colored in blue) is NUMA-aware, and assigns tasks based on NUMA domains; other systems, e.g., Presto [61] and SparkSQL [27] (colored in red), use all CPU cores, permitting the allocation of any task to any CPU core. As such there is no consensus on an allocation approach that uniformly works best, and none of the above-mentioned approaches explicitly consider the architectural features of chiplet-based CPUs.

In NUMA systems, an effective approach is to assign tasks to the CPU cores that are closest to the main memory where the accessed data resides [58]. Such an assignment respects data locality and minimizes remote memory accesses. In contrast, chiplet-based CPUs partition a more constrained L3 cache, even at a core level [22, 49, 54]. Therefore, assigning tasks to a subset of CPU cores, as

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3682011



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

Figure 2: Chiplet-based architectures

done for NUMA, improves cache efficiency but restricts available memory bandwidth; assigning tasks to use only a single chiplet's cores limits 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 explore empirically the impact of chiplet-based architectures on multi-core OLAP query engines, which perform parallel processing and are memory intensive, and identify efficient resource allocation strategies. Although our focus is on OLAP systems, our insights apply to systems with similar properties, e.g., NoSQL and graph processing engines.

Our study provides recommendations on how to improve the efficiency of query processing on chiplet-based CPUs without modifications to the query engines. In §4, we offer insights and guidelines on how to design and configure future engines to exploit the potential of chiplet-based CPUs:

(1) Task deployment at chiplet-level granularity. We recommend deploying query engine workers at a *chiplet-level granularity* (or finer). Such a chiplet-aware deployment policy boosts performance by up to 7× compared to a deployment with one worker instance per CPU core and up to 2× compared to NUMA-aware deployment (see Fig. 1b).

A chiplet-aware deployment policy has several advantages: (i) it avoids uneven data distributions that lead to excessive inter-chiplet communication and congestion; (ii) it allows complex queries to exploit efficient `tablescan` and `repartition` operations; and (iii) it benefits from the adoption of parallelized and distributed Bloom filters and hash join operations instead of nested-loop joins across worker instances, reducing data transfer volumes.

(2) Cache consciousness. We present guidelines for task deployment strategies at the chiplet-level granularity when considering the sizes of local and combined L3 caches. When the amount of accessed data is below the capacity of the local L3 cache within a single chiplet, users should deploy tasks per chiplet, i.e., only use cores within a single chiplet (and, therefore, local cache). When the data amount is larger than the L3 cache size, but smaller than the combined size of all chiplets' L3 caches, users should assign tasks to a selected group of cores, ensuring one core per chiplet. In the case of large amounts of accessed data, i.e., cache misses occur frequently, the performance of both strategies is similar.

(3) Data skew. We describe how a skewed data distribution impacts the performance gains of a chiplet-aware deployment. In particular, queries with extensive data scans and significant data shuffling can benefit from the enhanced core-to-core and L3 cache locality of a chiplet-aware deployment. We observe such improvements for complex benchmarks such as TPC-DS [23]. However, data skew increases cross-chiplet communication and congests the interconnect, reducing performance gains. Users must redistribute the communication traffic to mitigate this.

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

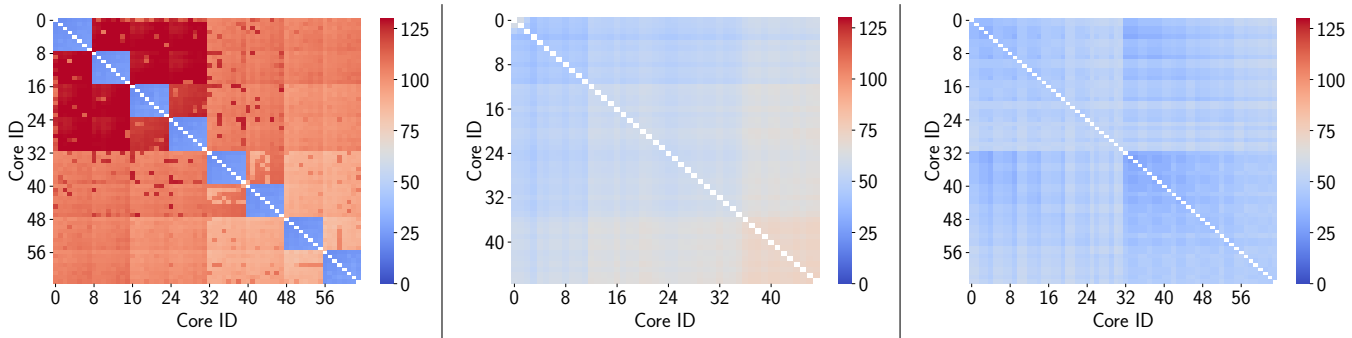
Chiplet-based CPUs introduce significant advancements in integrated circuit (IC) packaging and system integration, offering enhanced flexibility and configuration potential. They depart from monolithic ICs by employing a modular approach. These processors integrate multiple smaller semiconductor dies, called *chiplets*, onto a single package or substrate to create a functional unit.

This architecture has become increasingly popular among semiconductor manufacturers, including AMD, Intel and ARM, each using its own design. In this section, we investigate three different chiplet-based architectures (§2.1–§2.2) and show their impact on core-to-core latency (§2.3), bandwidth (§2.4), and aggregated memory bandwidth (§2.5).

2.1 Chiplet-based architectures

We examine the architectural features of chiplet-based CPUs from each major vendor (i.e., AMD EPYC Milan, Intel Sapphire Rapids, and ARM Graviton 3) and analyze their differences. Tab. 1 lists detailed specifications, and Fig. 2 shows the distinct architecture designs.

AMD EPYC Milan uses a chiplet-based architecture that features eight 7 nm compute chiplets, linked to a 14 nm central I/O die for both memory and I/O. 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. Such a processor supports 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 [49].



(a) Single-socket AMD EPYC Milan

(b) Single-socket Intel Sapphire Rapids

(c) Single-socket ARM Graviton 3

Figure 3: Core-to-core latency (ns)

Table 1: Architectural features of AMD Milan, Intel Sapphire Rapids and ARM Graviton 3 CPUs

	EPYC Milan	Sapphire Rapids	Graviton 3
Vendor	AMD	Intel	ARM
#Cores	Up to 64	Up to 56	64
#Chiplets	Up to 8	Up to 4	7
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	DDR5	DDR5
Max Memory Bandwidth	204.8 GB/s	307.2 GB/s	307.2 GB/s
Single Core Bandwidth	20-32 GB/s	17-23 GB/s	21.5 GB/s
Oversubscribed Bandwidth Per Core[1]	3.2 GB/s	5.49 GB/s	4.8 GB/s
Interconnect	Infinity Fabric	EMIB	LIPINCON

AMD's *Infinity Fabric* technique [49] enables cores to access L3 caches across chiplets at the expense of increased latency. The processor supports PCIe 4.0, allowing up to 32 GT/s per lane, and integrates eight DDR4 memory channels, providing shared access from the I/O die to the memory pool without dedicated DRAM banks per chiplet. The newest AMD EPYC Milan processor supports DDR5 memory. The latency varies based on the physical distance and routing through the Infinity Fabric.

Intel Sapphire Rapids (as used in this paper) introduces a new 4-tile design, known as the *XCC-tile architecture,* featuring up to 60 cores. Intel's "tile" refers to a modular unit similar to AMD's "chiplet", but with a broader range of computing functions. Each tile includes up to 14 cores that are connected to two DDR5 memory channels, resulting in a total bandwidth of more than 307 GB/s at DDR5-4800 speed. The architecture supports 2 MB L2 cache per core, and a shared 28.125 MB L3 cache per tile, amounting to 112.5 MB of combined non-inclusive L3 cache. Similar to AMD Milan, it allows cache access across tiles [56].

In addition, the MAX version of Sapphire Rapids incorporates up to 16 GB of High-Bandwidth Memory (HBM) per tile—64 GB in total [52]. This proximity of memory to the core greatly benefits tasks that handle large data sets, effectively acting like a large "L4" cache, but introduces variable access latency similar to NUMA systems. The processor supports PCIe 5.0, with up to 32 GT/s per lane.

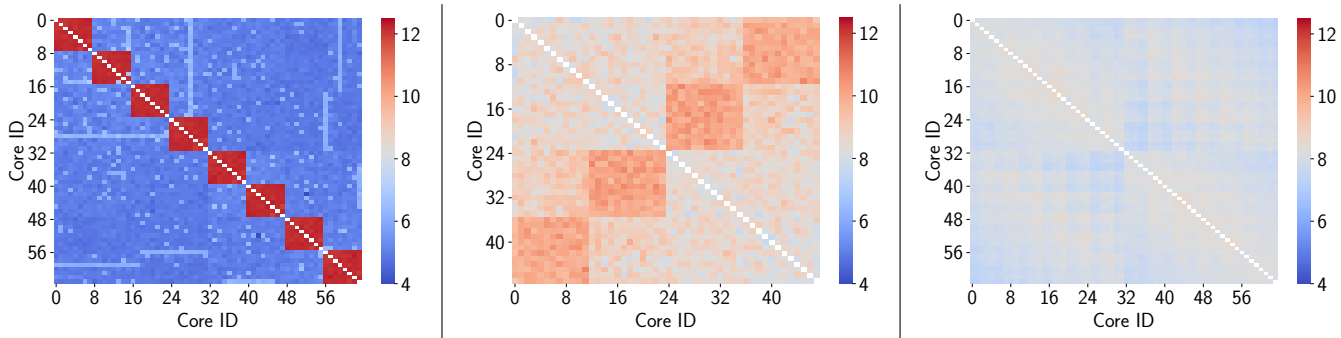
Intel uses the *Embedded Multi-die Interconnect Bridge* (EMIB) technique to connect semiconductor components side-by-side within one package [48], and the *Foveros* technique for vertical stacking that enables different tiles atop one another. Foveros reduces the data travel distance and thus transmission latency. Other variations of Sapphire Rapids, such as the MCC, feature a monolithic architecture with uniform interconnect among the cores. We exclude such a design, because it does not have multiple chiplets.

ARM Graviton 3 has a design with seven heterogeneous silicon components [8]. It places all 64 cores in a single chiplet. Each core has a separate 1 MB L3 cache, resulting in a combined 64 MB L3 cache. The DDR5 memory controllers and the PCI-Express 5.0 peripheral controllers are located separately from these central cores. Such a design differs from AMD and Intel, where a central I/O and memory die is surrounded by multiple computing chiplets. ARM's design emphasizes the separation of the core area from the memory and peripheral control sections. Core-to-core communication takes place via *Low-latency Inter-chiplet Passive Interconnect* (LIPINCON), which provides coherent data exchange across the processor.

Differences among chiplet-based processors. Despite having a similar core count, the existing chiplet architectures differ in modular design and numbers of chiplets:

- **Cache size.** AMD Milan offers up to 256 MB of combined L3 cache—significantly more than its Intel and ARM counterparts. This impacts data retrieval efficiency and latency.
- **Cache layout.** Intel and ARM processors bind L3 cache exclusively to cores. In contrast, AMD assigns L3 cache per chiplet, which is accessible to all cores within the same chiplet. This affects inter-core communication—only AMD support core-to-core communication via a shared cache between the two cores.
- **Memory support.** Sapphire Rapids and Graviton 3 support newer DDR5 memory, while AMD Milan only supports DDR4.
- **Inter-chiplet communication.** Processors optimize data transfer and integration in different ways. AMD's Infinity Fabric enables scalable multi-core integration with shared L3 cache across chiplets at the cost of heterogeneous inter-chiplet latency. Intel combines EMIB and Foveros to reduce data travel distances and latency. ARM focuses on low latency by keeping all cores on the same chiplet and uses LIPINCON to connect other components.

[1]Oversubscribed bandwidth per core refers to the reduced bandwidth available to each core when multiple cores simultaneously compete for access to the memory controller[12, 13].



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

Figure 4: Core-to-core bandwidth (GB/s).

- **Memory controller and bandwidth.** AMD and ARM separate memory controllers from the core dies. Cores across chiplets share the memory controller that may result in a reduced bandwidth given an unbalanced workload. Intel, on the other hand, integrates the memory controllers into the core dies. This enables a more equal workload distribution. Tab. 1 shows the maximum bandwidth of each processor that is achieved by a single core [12, 13]. When multiple cores share a memory controller, the memory bandwidth is limited by the controller's capacity. With oversubscription, the bandwidth to each core is reduced.

We observe that even subtle differences in the architectural design can lead to notable variations in performance. We next describe a detailed performance evaluation of the above three processors. We use a micro-benchmark, STREAM [50], to explore how the different chiplet architectures influence core-to-core latency, bandwidth, and overall system throughput.

2.2 Testbed configuration

We conduct our experiments on a diverse set of machines:

(1) Single-socket AMD EPYC Milan features an AMD EPYC 7713 processor with 64 CPU cores, 256 MB of L3 cache across 8 chiplets, and 512 GB of RAM. It has a 64 MB L2 cache and runs Ubuntu 23.04. We use this processor in §2.3 and §2.4, to highlight chiplet properties, as the effects of NUMA on inter-socket communication are well-known [18, 33, 40, 46].

(2) Single-socket Intel Sapphire Rapids has an Intel Xeon Platinum 8488C processor with 48 cores, 192 GB of RAM, 2.1875 MB of L3 cache and 2 MB of L2 cache per core, running Ubuntu 22.04.

(3) Single-socket ARM Graviton 3 has a processor with 64 cores, 128 GB of RAM, 32 MB of L3 cache and 1 MB of L2 cache per core, running Ubuntu 22.04.

In addition, we use the following two machines in §2.5:

(4) Dual-socket AMD EPYC Milan features two AMD EPYC 7713 processor, with a total of 128 CPU cores and 1024 GB of RAM. We use this machine instead of the single-socket AMD processor.

(5) Single-socket Intel Xeon Gold features a 3rd Generation Intel processor with 24 cores, 128 GB of RAM, 36 MB of L3 cache and 30 MB of L2 cache, running Ubuntu 22.04.

We use the dual-socket AMD processor to measure aggregate bandwidth variations across the NUMA domains, and the 3rd Gen. Intel machine to compare the results with a monolithic architecture.

2.3 Core-to-core latency

We measure the latency of sending a message from one core to another, by pinning two threads to separate cores. We perform multiple compare-and-swap (CAS) operations and measure the latency. Fig. 3 illustrates the heatmap of core-to-core latency for each processor.

AMD EPYC Milan shows a wide latency range, differing up to 10× (see Fig. 3a). On average, latency within the same socket is 106 ns. Within a single chiplet, the latency drops to just 24 ns.

In contrast, the Intel Sapphire Rapids processor shows more consistent latency across its cores, averaging 59 ns (see Fig. 3b). Such uniformity suggests that Intel's design favors consistent data transfer speeds, which is beneficial for stable performance of parallel tasks and multi-threaded applications.

The average latency of ARM Graviton 3 is 48 ns, and this latency remains consistently below 59 ns throughout the entire chip. The heatmap in Fig. 3c shows a mixed pattern of red and blue squares, with 4 blocks of thread pairs with lower latency than the others. This is due to the chip's internal architecture, i.e., the arrangement of cores within the chiplet, the layout of interconnects, and the proximity to shared resources. Our evaluation results align with those reported in [11].

Insights. *AMD processors exhibit varying latencies 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. In contrast, Intel Sapphire Rapids and ARM Graviton processors achieve consistent latency by avoiding shared caches between cores. The ARM 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

Next, we examine the bandwidth sustained between core-to-core communication. We use two threads, one sender and one receiver, each pinned to a specific CPU core. To avoid cache conflicts, we align memory access to the hardware cache line size. After warming

up the cache by writing random data into an 8192 byte-sized buffer, the sender populates the buffer with random data, and the receiver times the accesses. Effective bandwidth is measured as the average latency of 10 trials. Fig. 4 shows the results.

AMD Milan exhibits a distinct and consistent bandwidth pattern in core-to-core communication (see Fig. 4a), mirroring the results of the core-to-core latency analysis. Within the same chiplet, cores communicate at 12 GB/s. However, this speed decreases to 6 GB/s between cores from different chiplets, indicating significant variation based on the core location.

Sapphire Rapids, in contrast, exhibits a more homogeneous bandwidth distribution. In Fig. 4b we notice distinct groups of cores where bandwidth peaks at 12.5 GB/s. Otherwise, it averages at 11 GB/s and occasionally falls to 10.5 GB/s. Despite the overall uniformity in bandwidth, the delineation of the four chiplets is visible, even if negligible.

An interesting observation is that cores in the chiplets farthest apart have a slightly higher bandwidth, forming an inverted diagonal pattern on the heatmap. The architecture of the Intel Sapphire Rapids includes high bandwidth cross-die connections [22, 51]. These high bandwidth connections are likely the reason for the improved performance when cores communicate across chiplets [48].

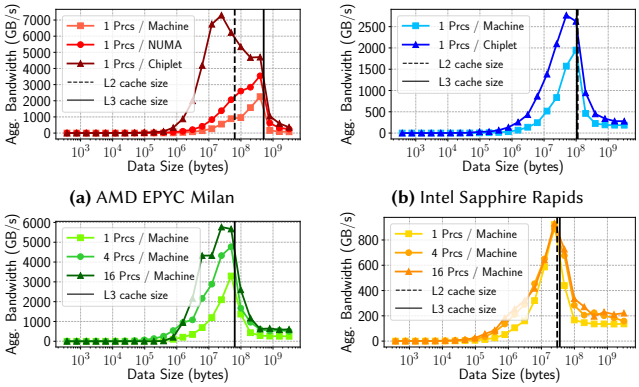
The ARM Graviton 3 processor demonstrates a notably consistent communication bandwidth profile between cores (see Fig. 4c), in a range between 7.2 GB/s and 8.6 GB/s. The regions closer to the diagonal display a red coloration, signifying enhanced bandwidth. This suggests that the bandwidth is higher when cores communicate with their neighbours.

Insights. *The core-to-core bandwidth analysis aligns with the trends of the previous latency analysis for AMD Milan, Intel Sapphire Rapids and ARM Graviton 3, highlighting the impact of architectural designs.*

2.5 Aggregate memory bandwidth

We also investigate the aggregate memory bandwidth of chiplet-based processors. We use the STREAM benchmark [50] to measure sustainable memory bandwidth by performing simple operations on stored arrays of data. We focus on the `COPY` function, which copies the contents of one array to another, with a direct memory-to-memory data transfer implemented via a loop-based copy operation. We vary the number of processes, ranging from a single process that uses all available resources to multiple processes bounded to specific chiplets or subsets of chiplet's cores. Each process runs the STREAM benchmark and spawns multiple threads, which are restricted to the resources allocated to that process. For each configuration, all processes are initiated simultaneously, ensuring the total data array size is consistently and evenly distributed.

The goal is to examine which setup can take a better advantage of chiplet architectures. We expect to see better efficiency within the chiplet boundaries. Fig. 5 shows the measured aggregate memory bandwidth for all the processors in §2.2. We observe that, 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 display the highest aggregate bandwidths using one process per chiplet, achieving 7 GB/s and 2.7 GB/s, respectively. Moreover, the AMD Milan processor, the only one tested with 2 NUMA domains, shows



(a) AMD EPYC Milan **(b)** Intel Sapphire Rapids

(c) ARM Graviton 3 **(d)** 3rd Gen. Intel Xeon Gold

Figure 5: STREAM Benchmark (The darker the colour, the higher the number of parallel processes)

that this configuration outperforms even the one that accounts for the presence of NUMA architecture, reaching only 3.5 GB/s. The ARM Graviton 3 processor, despite having a single computational chiplet, achieves the highest bandwidth with 16 processes, peaking at 5.8 GB/s. The aggregate bandwidth of ARM demonstrates that using multiple processes can increase aggregate bandwidth, even with a single chiplet, due to the partitioned L3 cache at the core level. On the other hand, 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.

The different values of aggregate bandwidth can be attributed to the varying characteristics of the machines used, e.g., the number of cores and NUMA domains.

Insights. *Directing tasks to specific core subsets presents a trade-off in chiplet-based architectures: on one hand, it reduces the available L3 cache, which may be a concern for bandwidth-intensive workloads; on the other hand, our findings indicate that this approach can increase the aggregate bandwidth in chiplet-based architectures. The key factor is more efficient cache use and improved data locality.*

3 DEPLOYMENTS OF QUERY ENGINES

In this section, we focus on distributed query engines and explore how different deployment policies impact their performance on chiplet-based processors.

3.1 Architecture of distributed query engines

From a design perspective, distributed cloud-native *query engines*, e.g., AWS Redshift [41], Athena [6], Google's Big Query [38], Microsoft's Polaris [25], on-premise *data warehouses*, e.g., Exadata [20] and Teradata [4], and *big data systems*, e.g., Hadoop [1], Presto [61] and Spark [27], all share similarities. Since they are all designed for scalability, they feature a compute layer of *database worker instances*, which execute incoming queries that operate on data fragments. The worker instances are governed by a *coordinator*. The coordinator is responsible for admitting, parsing, planning and optimizing queries, in addition to orchestrating how a query's tasks are distributed among worker instances.

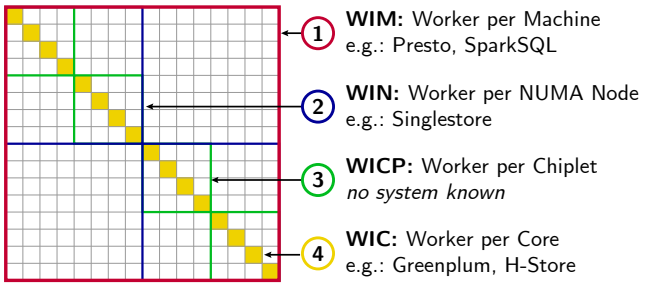


Figure 6: Design configurations for distributed query engines in a dual-socket processor

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. Although network technologies such as RDMA and InfiniBand significantly reduce network bottlenecks [31], conventional systems often do not use them and rather minimize network communication to avoid potential bottlenecks. As a result, each database worker instance handles its own task queue for its data partition (fragments) to achieve better data locality and reduce network traffic. The coordinator computes a cost-based query plan, which accounts for data distribution and tries to minimize the communication overhead [21, 34, 61].

Different query engines adopt different deployment policies for worker instances, even within a single multi-processor machine.

3.2 Design space overview and trade-offs

We explore several scenarios, from deploying a single worker instance on a single CPU core to utilizing the entire machine for a single worker. We want to investigate the full spectrum of deployment granularities and understand how each impacts performance and efficiency. We differentiate between three main options: 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 §2, it is clear that adopting a new category that has a Single-Worker Instance per Chiplet (`WICP`) is a promising approach. The main differences between these approaches concern (i) the number of database worker instances spawned on a single machine, (ii) the way computing resources are allocated to each worker instance, and (iii) the data allocation policies. By data allocation policies, we refer specifically to the policy 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 superior 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.

Different designs also affect the parallelism degree, e.g., the number of active worker instances, and the amount of exchanged data.

(1) `WIM`: Single-Worker Instance per Machine. The Single-Worker Instance per Machine policy is used by many distributed query engines [17, 21], in particular Presto [61] and SparkSQL [27]. The deployment policy is independent of the topology of the underlying hardware and a single database worker instance manages all resources, as shown in Fig. 9. It is up to the query engine to decide how to allocate or access the data fragments. 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 that the worker instances must manage their resources internally to get the most out of the underlying hardware.

Advantages: The implementation is independent from 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 is minimized [29, 57].

Disadvantages: Maximizing data and thread locality is non-trivial. Inadequate allocation of data between the chiplets' L3 caches risks congesting interconnects, leading to performance reductions. Redesigning the database engine to make it more hardware-aware requires cost and effort, and, in many cases, this is not viable.

(2) `WIN`: Single-Worker Instance per NUMA. In the Single-Worker Instance per NUMA, there are as many database worker instances as NUMA nodes. Each worker instance 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 the worker is bound. In terms of data allocation, the `WIN` design relies on the Membind policy [19], which forces each instance to only use its local memory. An example of this design can be found in SingleStore [10]. SingleStore places each database worker instance on a separate NUMA node [3].

Advantages: This policy avoids memory accesses across NUMA nodes and avoids interconnect congestion. The work is partitioned by the database engine, thus generating a fair use of the resources shared in NUMA domains.

Disadvantages: As shown in §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 can also be high, and managing memory across multiple chiplets can be complex. Ensuring good cache locality may require a chiplet-aware scheduling policy, which adds overhead and complexity to system operation. In addition, local inter-worker communication takes place via the network stack with less bandwidth than what can be achieved through interconnects. A larger number of database worker instances increases inter-machine communication.

(3) `WICP`: Single-Worker Instance per Chiplet. 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 exclusively using the cache and memory local to its chiplet. Data is allocated in the local main memory of the NUMA node where the chiplet resides, and can be cached only in the chiplet's local L3 cache. As far as we know, this deployment policy is not currently used 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 working

on its own cache, the WICP design eliminates cross-chiplet communication and favors a more balanced data allocation, avoiding congestion of the interconnect.

Disadvantages: By forcing instances to run on a specific subset of cores, we can restrict their memory bandwidth. Moreover, by forcing a worker instance to use only a single chiplet's cores, we limit the access to other chiplets' L3 caches, thereby shrinking the available cache size and necessitating slower main memory use for data exceeding the 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 uses one database worker instance per CPU core and is adopted by systems such as Greenplum [2, 5], and H-Store [43]. Here, we use as many database workers as there are CPU cores on the machine. All worker instances exchange data via the network stack. In addition, each instance prioritizes data placement in the memory and the cache local to the core of the worker instance.

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

Disadvantages: The WIC deployment policy increases the amount of communication and data exchange required for data processing. In addition, as with WICP, each process is limited to using only the relevant cache partition available in its chiplet.

4 EXPERIMENTAL ANALYSIS

We conduct a range of experiments to evaluate the behavior of the 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 on 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 with respect to different characteristics (§4.5) including (i) the number of worker instances of a query engine (§4.5.1); (ii) the properties of the query workload (§4.5.2); and (iii) the data skew (§4.5.3)?

4.1 Experimental setup

Testbed. The experiments are conducted on a diverse set of chiplet-based machines. This evaluation environment differs from the one described in §2, and now also includes two machines equipped with two NUMA domains. We selected dual-socket machines for our evaluation, because certain systems optimize their deployment strategies according to the number of NUMA nodes present [10]. Specifically, we use the following:

(1) AMD EPYC Milan features a dual-socket AMD EPYC 7713 processor, with 64 CPU cores per socket, 512 GB of RAM, and 8 chiplets. It runs Ubuntu 23.04.

(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, each containing 4 tiles, and runs Ubuntu 22.04.

(3) ARM Graviton 3 has a single-socket processor with 64 CPU cores and 128 GB of RAM, running Ubuntu 22.04.

Query engines. We use common distributed query engines including Presto [61], SingleStore [34] and SparkSQL [27]. The default deployment policy is *Single-Worker Instance per Machine* (WIM).

We aim to show how existing engines can manage the heterogeneity of chiplet-based architectures, without modifying the systems' source code. We note, however, that tailored in-engine optimizations can offer further performance improvements, but this is beyond the scope of this paper.

Benchmark suites. We use a set of benchmarks including TPC-H [24], TPC-DS [23], and JCC-H [32]. Overall, they contain 143 queries and represent various types of workloads, including ad-hoc analytical queries and decision support queries. In the TPC-H, TPC-DS, and JCC-H evaluations, queries are executed sequentially.

To explore various deployment policies, we leverage the *libnuma* library [45] to spawn worker instances on a core/NUMA/chiplet and enforce a given data placement. In our evaluation, unless specified otherwise, we utilize datasets with a scale factor of 100 (i.e., 100 GB of data size). All results are averaged over five runs.

Metrics. We measure the query execution time of all deployment policies (WIM, WIN, WICP and WIC) for each query engine on each machine. For better visualization, we compare the speedups of WIN, WICP and WIC over the baseline—the default deployment of WIM.

4.2 Deployment on chiplet-based machines

We investigate the performance impact of the 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 and compare the speedups of WIC, WIN and WICP over the default deployment policy, WIM. We first explore the general trends before delving into a detailed investigation of results, specifically on the AMD EPYC Milan.

Recall that the Graviton 3 is a single-socket design composed of seven chiplets, with only one containing the computational cores. Consequently, the WIM, WIN and WICP policies ultimately correspond to the same configuration. We use a modified version of WICP with multiple worker instances per chiplet. The ARM variant of WICP employs 16 worker instances, each assigned 4 cores, similar to the setup used in the STREAM benchmark from §2.5.

Deployment efficiency across chiplet architectures. Fig. 7 shows the geometric mean speedup of TPC-H queries for the different deployment policies: WIC, WICP, and WIN, with WIM as the baseline.

The WICP policy consistently outperforms the others, notably enhancing SparkSQL's performance with 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, achieving improved efficiency across the board:

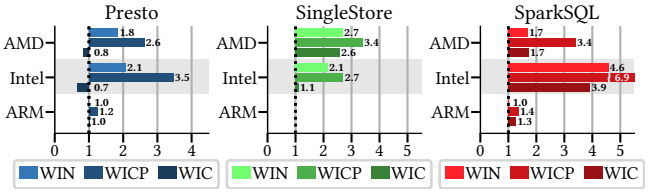


Figure 7: Geomean speedups over `WIM`

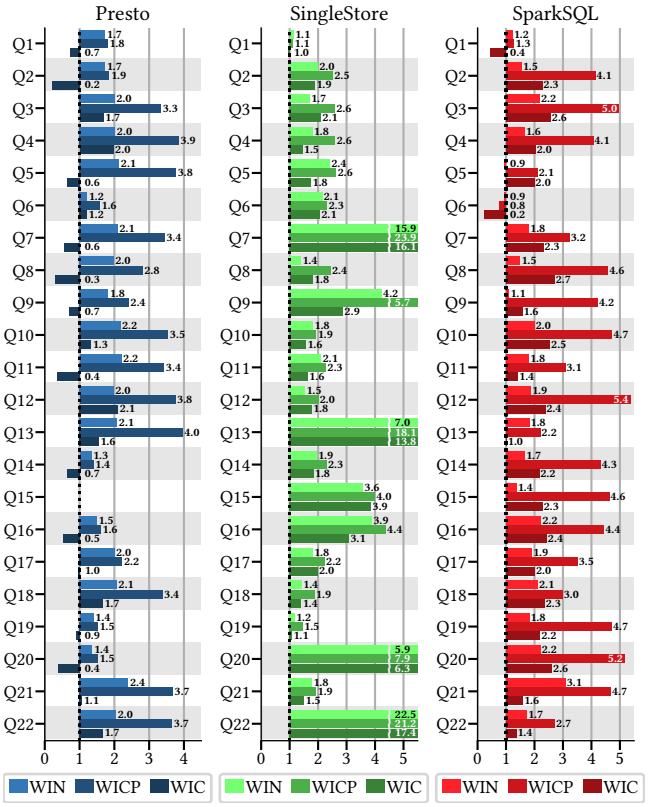


Figure 8: TPC-H speedups on AMD EPYC Milan with scale factor 100

1.71× on AMD, 3.92× on Intel, and 1.27× on ARM. The reason is that the SparkSQL engine unevenly allocates data across the multiple threads with the `WIM` policy. This leads to higher LLC miss rates, a high rate of remote NUMA reads and consequently a congested interconnect hindering efficiency. `WICP` mitigates these issues by evenly spreading data among database worker instances, enhancing data locality and reducing the interconnect pressure.

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, although Intel Sapphire Rapids offers better inter-core communication than AMD EPYC Milan, 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 coordination, is restricted to using just one core, thereby slowing down the overall execution during aggregations.

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

Analysis of TPC-H queries. We also note varied performance across individual queries as shown in Fig. 8. We now focus on a detailed investigation of all queries on each evaluated system.

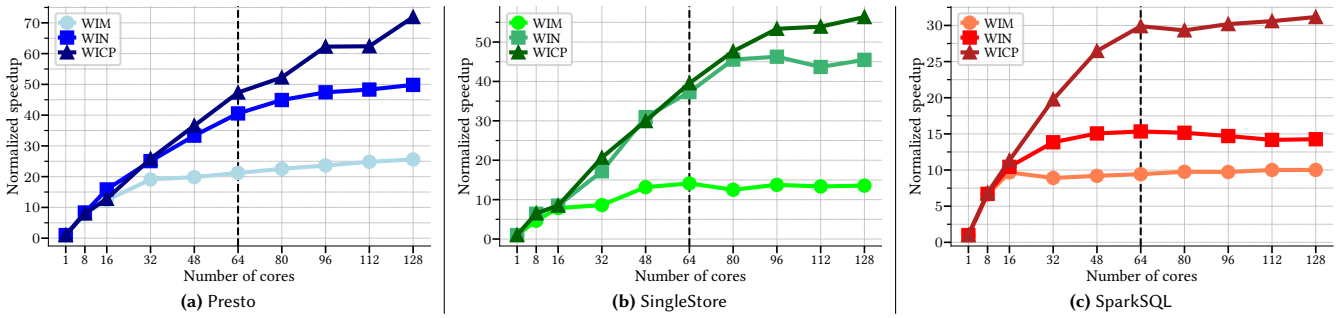
(1) SingleStore's closed-source nature restricts our analysis to query plans, preventing us from examining specific algorithms or runtime-generated code. From the query plans, we observe that SingleStore uses different query plans under the `WICP` policy compared to those generated under the `WIM` policy. The query plans for `WICP` enhance data processing speeds by optimizing data locality for table scans and employing parallelized and distributed query operators (e.g., Bloom filters and hash joins).

Specifically, Q7, Q9, Q13, Q15, Q16, Q20, and Q22 experience the biggest boost in performance, 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 [36]. 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 complete in 0.25 seconds under `WICP` compared to 5.06 seconds under `WIM`. We attribute the speedup to `WICP`'s superior data locality that minimizes data access times. Consequently, Q13, Q20, and Q22 exhibit better performance over Presto and SparkSQL, with speedups of 18×, 8× and 21×, respectively.

SingleStore achieves speedups of 23.9× and 5.7× under the `WICP` policy for Q7 and Q9, respectively. This is attributed to the adoption of 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, because they are easily parallelizable and distributable across worker instances, cutting down the data transfer volume and execution time.

(2) Presto. In contrast to SingleStore, Presto uses a consistent query plan across all policies. However, the number of generated data fragments depends on the number of worker instances, and the computing resources assigned. As the number of workers increases, the number of (smaller) fragments also goes up. The smaller the fragments, the more likely they fit in the L3 cache, boosting data locality. The engine's improved efficiency is particularly evident when executing the ScanFilterTable operator, which dominates Presto's query execution time. By refining this process, Presto significantly reduces the execution times across all benchmark queries. In fact, Q5 and Q18 show the best speedups, as they spend 88% and 96% of their execution time, respectively, on the ScanFilter operation on Orders. Q1, Q2, Q6, Q14, Q16, Q19 and Q20 exhibit less than a 2× speedup, because they do not involve the Orders table.

Interestingly, despite the Lineitem table being larger than the Orders table in the TPC-H benchmark, we do not observe the same



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

Figure 9: Multicore scalability on AMD EPYC Milan

performance gains. This is due to Presto's approach to handling scans and filters. Before executing joins, Presto typically scans and filters tables to reduce the number of rows processed. However, Presto skips this step for the Lineitem table, due to its extensive size and the significant time typically consumed by this phase. As a result, queries involving the Orders table perform better compared to those using the Lineitem table (e.g., Q1, Q6, Q14, Q19, and Q20).

The `WIC` deployment underperforms on Presto compared to SparkSQL and SingleStore because, in our evaluation, Presto's Master instance also acts as a Worker, utilizing one core just like the other Worker instances. In contrast, SparkSQL and SingleStore use a dedicated Master that can use all the resources, allowing more efficient task coordination.

(3) SparkSQL. Unlike SingleStore and Presto, SparkSQL is particularly affected by the negative effects of unbalanced data distribution. Fig. 8 shows that SparkSQL achieves speedups ranging from 3.51× to 5.15× 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 interconnect congestion. Our observation is that the interconnect is congested for an average of 34% of the query execution time. Such congestion particularly impacts the join operations, because SparkSQL relies on SortMerge Join. The `WICP` deployment strategy effectively mitigates these issues by forcing the use of local memory, ensuring a balanced data distribution.

Insights. *Users should deploy the query engine workers on chiplet-based machines using the* `WICP` *policy for improved performance.* `WICP` *enhances data locality, boosting the efficiency of queries that involve extensive data scanning or shuffling. The higher distribution level of the* `WICP` *policy, compared to* `WIM` *and* `WIN`*, produces query plans that better exploit easily parallelizable operators.*

4.3 Multi-core scalability

Next, we examine the scalability of different deployment policies on the AMD EPYC Milan machine. We evaluate the performance of the TPC-H queries using three query engines (i.e., Presto, SingleStore and SparkSQL) and measure their speedups when increasing the CPU core count from 1 to 128. The results are shown in Fig. 9.

`WICP` scales 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. `WICP` improves shared memory access by taking into account the partitioned L3 cache and ensuring an even data distribution, thus reducing resource contention.

`WIN` also exhibits good scalability up to 64 cores, but its performance increases more slowly 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), respectively. This superior scalability of `WICP`, especially for SparkSQL, is attributed to improved cache utilization within chiplets and minimized interconnect congestion. Furthermore, the high inter-core communication variability in a single NUMA domain on the AMD EPYC Milan machine, discussed in §2, reduces `WIN`'s scalability.

As expected, `WIM` shows 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 data access, while fetching data from other chiplets incurs inter-chiplet communication overhead. `WICP_Local` enforces data access on the local L3 cache, 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 reports the aggregated bandwidth when increasing the size of an array from 0.8 MB to 1536 MB. The bandwidth is lower than previously shown in Fig. 5a due to using a smaller number of cores.

We observe that `WICP_Local` achieves higher bandwidth than `WICP_Mixed` until the array size reaches 32 MB, the single chiplet's L3 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 data from main memory. In contrast, `WICP_Mixed` shows more stable bandwidth due to its

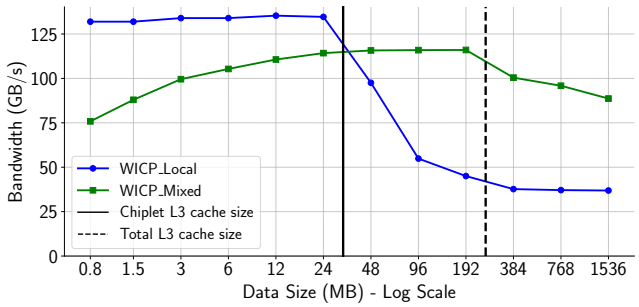


Figure 10: AMD Milan: STREAM benchmark with 8 cores (1 chiplet)

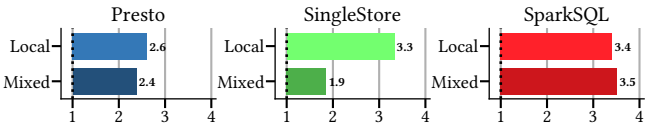


Figure 11: Geomean speedups over `WIM` on AMD Milan

ability to leverage the total capacity of L3 caches from all chiplets, 256 MB. Beyond this, `WICP_Mixed`'s bandwidth also declines.

We further examine `WICP_Local`'s and `WICP_Mixed`'s impact on query performance with Presto, Singlestore, and SparkSQL using the TPC-H benchmark, with a scale factor of 50 GB, on the same AMD EPYC Milan processor. We compare their geomean speedups over the default `WIM` deployment, as shown in Fig. 11.

We observe performance speedups for both `WICP_Local` and `WICP_Mixed`, but different query engines show different impacts. For SingleStore, `WICP_Local` achieves much higher speedups than `WICP_Mixed`, 3.32× vs. 1.85×. In the same vein, Presto has a preference for `WICP_Local` over `WICP_Mixed`, but the gap is smaller, 2.61× vs. 2.38×. However, SparkSQL shows a different trend with `WICP_Mixed` (3.40×) that slightly outperforms `WICP_Local` (3.32×). These different behaviors stem from how these query engines implement data movement during the `shuffle` and `join` operations. Tab. 2 shows the peak memory and bandwidth when executing the most relevant queries for each query engine.

SingleStore stands out as the only system that transfers data within the capacity of an L3 cache (i.e., `WICP_Local` performs much better). Tab. 2 also shows that SingleStore exhibits a more significant difference in maximum bandwidth between `WICP_Local` and `WICP_Mixed`, compared to Presto and SparkSQL. It consistently achieves higher bandwidth in `WICP_Local`. In contrast, both Presto and SparkSQL incur larger data movement beyond the combined L3 cache capacity, resulting in data being fetched 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.

When data exceeds a single L3 cache but fits within the combined L3 caches, the `WICP_Mixed` policy utilizes cache resources more effectively. Therefore, when deciding on a placement policy, the task scheduler should take into account the available cache size, the chiplet's share of L3 cache, and the working set size.

Insights. *Users should use the* `WICP_Local` *deployment policy when the working set is below the capacity of a single chiplet's L3 cache. When the amount of data is larger than the chiplet's share of the L3 cache size but smaller than the size of combined L3 caches, one should use the* `WICP_Mixed` *deployment policy. When accessing a larger amount of data, the performances of* `WICP_Local` *and* `WICP_Mixed` *are similar, and the thread to core placement has less impact.*

Table 2: Peak memory usage and bandwidth of TPC-H queries

Presto	Query ID	Memory (GB)	Local (GB/s)	Mixed (GB/s)
	2	2.16	**251.49**	233,51
	7	9.13	**298.38**	284.99
	17	1.71	236.03	**236.07**
	19	0.51	**221.15**	217.14
	21	32.80	**291.87**	281.21
SingleStore	Query ID	Memory (GB)	Local (GB/s)	Mixed (GB/s)
	2	0.02	**41.38**	37.86
	7	0.32	**210.83**	165.26
	17	0.32	**220.91**	177.40
	19	0.02	**138.95**	99.43
	21	0.30	**182.16**	153.11
SparkSQL	Query ID	Memory (GB)	Local (GB/s)	Mixed (GB/s)
	2	2.10	146.30	**155.28**
	7	3.40	**209.25**	204.86
	17	6.80	**245.37**	225.65
	19	0.43	118.64	**123.98**
	21	12.60	**253.27**	250.12

4.5 Sensitivity analysis

In this section, we explore the sensitivity and real-world applicability of our finding with a diverse set of workloads and configurations.

4.5.1 Number of worker instances. We first explore how the number of worker instances affects the query performance of `WICP` on the Intel Sapphire Rapids. We vary the number of worker instances n, where each worker instance is 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 shows the geometric mean of speedups on TPC-H. Here, Presto (Fig. 12a), SingleStore (Fig. 12b), and SparkSQL (Fig. 12c) exhibit the same trend that the speedup keeps increasing to 8 worker instances, reaching 3.62×, 2.75× and 7.04×, respectively. Their speedups plateau at 8 worker instances and start to deteriorate from 28 worker instances. Note that Intel Sapphire Rapids has 8 chiplets. This means query engines achieve a superior balance of computation and communication granularity when the number of worker instances equals 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. 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 query diversity on performance. All queries exhibit speedups with the `WICP` and `WIN` deployment policies. Q1, Q21, Q22, Q37, Q39, Q81, Q83, Q84 and Q91 show the highest speedups because they spend most of the execution time scanning large tables (e.g., the `inventory` table), which benefit from `WICP` (see §4.2). By examining the execution plans, we find that such scan phases cause expensive remote memory accesses in `WIM`.

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

Figure 12: Varying the number of database worker instances on a dual-socket Intel Sapphire Rapids (geomean speedup with TPC-H SF 100)

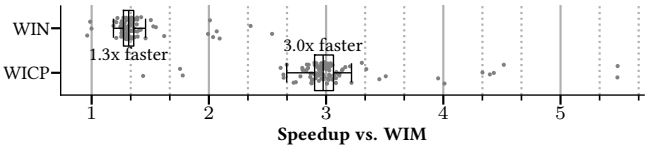


Figure 13: Presto: TPC-DS on AMD EPYC Milan

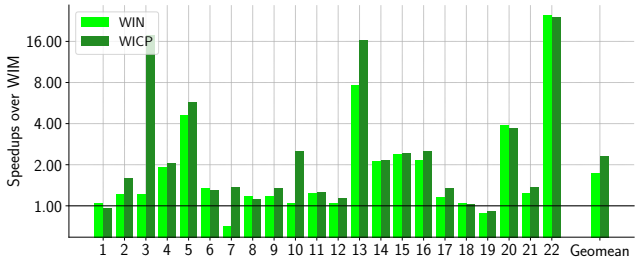


Figure 14: SingleStore: JCC-H on AMD EPYC Milan

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 with complex benchmark suites such as TPC-DS.*

4.5.3 Data skew. Finally, we explore the impact of skewed workloads on the performance of deployment policies. We use SingleStore to evaluate with the JCC-H benchmark [32], which introduces the join-crossing-correlations and injects skew 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 workloads affect the performance of the WICP. The geometric mean speedups for WICP and WIN are 2.31× and 1.72×, respectively, compared to TPC-H's 3.40× and 2.68×, respectively. The reason is that data skew by nature enforces more locality in the default WIM policy and challenges the efficiency of WICP by leading to imbalanced workload distribution. Specifically, data skew 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 resource contention. 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, relying on slower main memory accesses.

To effectively manage data skew in chiplet-based processors, one strategy is to implement skew-aware resource allocation while keeping the processing localized. This involves dynamically adjusting the resources from each chiplet based on the data distribution [26]. For example, in a distributed computing system (e.g., Spark [16]), instances handling larger or more complex data partitions can be allocated specific chiplet's resources that are local to those data partitions. This dynamic allocation can be achieved using real-time monitoring tools that analyze data load on chiplets and performance metrics across instances, similar to those used in distributed database systems (e.g., Apache Cassandra [30]).

Insights. *Designing data systems on chiplet-based processors must account for data skew, which can hinder performance gains with* WICP *due to increased cross-chiplet communication and congestion. Despite this,* WICP *generally outperforms* WIN *for most queries. To effectively manage data skew, guidelines include (i) the redistribution of communication traffic, (ii) the implementation of skew-aware resource allocation, and (iii) the dynamic adjustment of chiplet resources based on data distribution while maintaining localized processing.*

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 it is the flexibility that chiplets offer. Manufacturers can mix and match different components such as CPUs, GPUs, and memory to create configurations that meet specific performance needs for particular applications. The introduction of standards such as the Universal Chiplet Interconnect Express (UCIe) further cements the future of chiplets in the industry [62]. UCIe standardizes the interconnects between different chiplets, facilitating interoperability between components made by various manufacturers, which speeds up the adoption of chiplet technology.

Another exciting development is the integration of High Bandwidth Memory (HBM) into chiplet designs, as used by Intel processors. This integration significantly increases memory bandwidth, making chiplet-based processors especially effective for data-intensive tasks.

NUMA domain configuration. NUMA is an integral part of modern multiprocessing systems. 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 boost performance by improving memory and cache utilization, especially in demanding applications such as large-scale databases and data analytics. However, this is not a complete solution as many systems, such as Presto or SparkSQL, do not effectively exploit multiple NUMA domains. In conclusion, these features can only realize their full potential when paired with software that is equally attuned to the nuances of the hardware.

Inter-machine scalability. When discussing distributed query engines, it is essential to recognize that they are designed for multi-machine environments. We actually exploit this when applying the WICP deployment design and leaving 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, resulting in increased communication overhead during query processing. Thus, one should consider integrating multiple high-performance network cards with the database's worker instances during setup to mitigate NIC-related bottlenecks.

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 example, 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.

Chiplet dollar cost. Chiplet technology is still relatively new, but as it matures and production volume increases, economies of scale will likely bring down the cost of chiplet-based servers [9]. Even today, the list price for Intel's 4th Gen. Xeon Platinum processors is 14% higher compared to 3rd Gen. monolithic processors with the same CPU core count [14, 15]. In addition, chiplets improve manufacturing yield, as smaller chiplets are less likely to have defects [37], and their modular design enables reuse, which amortizes design costs over multiple products.

Dennard scaling. The slowdown of Dennard scaling, which previously enabled power efficiency gains with transistor miniaturization, has limited the scaling of traditional monolithic processors. This slowdown is partly due to increased manufacturing difficulties, which led to higher error rates and lower yields as transistors become smaller. To address these challenges and continue scaling, chiplet technology has emerged as an alternative approach. Breaking down processors into smaller, modular chiplets, maintains higher yields and improves power efficiency compared to a monolithic design that crams everything onto one die [47]. This modular approach allows for independent scaling of different chiplet types, leading to a higher CPU core count than traditional NUMA designs.

6 RELATED WORK

Recent CPU advancements are comprehensively documented in industrial white papers and vendor manuals from major companies such as AMD and Intel [49, 54, 56]. Leading researchers [42] have also brought attention to these developments. For a deeper understanding of physical implementations and control mechanisms, peer-reviewed studies provide valuable insights. For example, Suggs et al. [63] investigate the Zen 2 architecture; Schöne et al. [60] focus on energy efficiency aspects; and Naffziger et al. [55] discuss multi-die chiplet configurations.

Other prior studies have evaluated the latency and bandwidth capabilities of chiplet-based processors: Velten et al. [64] conduct an in-depth experimental evaluation of the memory hierarchy in AMD EPYC Rome and Intel Xeon Cascade Lake SP server processors; Fotouhi et al. [39] examine the energy and performance benefits of a point-to-point silicon-photonic interconnect, highlighting its potential to create a scalable chiplet-based UMA system; and Chirkov et al. [35] also evaluate the interconnect's performance and introduce Meduza, a write-update coherence protocol for chiplet systems.

To date and to the best of our knowledge, there have been no studies on the effect of chiplet processor designs on data intensive applications such as databases and distributed query engines. Prior work has evaluated the performance of distributed databases when running on modern multiprocessor machines [44, 58, 59] but with a particular focus on NUMA. For example, Porobic et al. [58] introduce the concept of hardware islands and focus on determining the right trade-off between shared-everything and shared-nothing deployments in multiprocessor systems, measuring the impact of distributed transactions and skewed requests on different OLTP workloads. Similarly, Salomie et al. [59] proposed to treat a multicore machine as a distributed system and replicate their internal logic to address their inherent scalability limitations.

7 CONCLUSION

Our work focuses on the critical influence of chiplet-based CPU architectures on the performance of multi-core query engines. This is a timely study given the recent adoption of chiplet designs by all leading processor manufacturers. We propose a chiplet-granular deployment policy that optimizes task allocation to exploit the architectural benefits of chiplets. With a wide range of experiments, we show that one can significantly enhance the query performance with careful chiplet-aware placement of tasks and data, even without modifying existing engines, regardless of whether they are used on-premise or deployed in the cloud environment.

Our observations help us derive guidelines on internal optimizations that can be used within engines, customizing their resource allocation policies on novel chiplet-based processors. Overall, we conclude that query engines need us to rethink resource allocation strategies to exploit the underlying hardware resources of chiplet-based processors to a full extent.

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. Amazon Athena. https://docs.aws.amazon.com/whitepapers/latest/big-data-analytics-options/amazon-athena.html. Accessed: 2023-6-19.
[7] 2024. Amazon Redshift provisioned clusters. https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-clusters.html. Accessed: 2023-6-19.
[8] 2024. Analyzing Unconventional Logic Semiconductors – A Shift Away from Semiconductor Manufacturers. https://hacarus.com/ai-lab/03312022-graviton3/. Accessed: 2024-3-1.
[9] 2024. Chiplet Market Update. https://chipletsummit.com/proceeding_files/a0q5f000001WuE0/20230125_PLEN_Hackenberg.PDF. Accessed: 2024-5-12.
[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. Core to Core Latency Data on Large Systems. https://chipsandcheese.com/2023/11/07/core-to-core-latency-data-on-large-systems/. Accessed: 2024-5-12.
[12] 2024. The evolution of single-core bandwidth in multicore processors. https://sites.utexas.edu/jdm4372/2023/04/25/the-evolution-of-single-core-bandwidth-in-multicore-processors/. Accessed: 2024-5-12.
[13] 2024. The evolution of single-core bandwidth in multicore systems — update. https://sites.utexas.edu/jdm4372/2023/12/19/the-evolution-of-single-core-bandwidth-in-multicore-systems-update/. Accessed: 2024-5-12.
[14] 2024. Intel® Xeon® Platinum 8380 Processor. https://ark.intel.com/content/www/us/en/ark/products/212287/intel-xeon-platinum-8380-processor-60m-

cache-2-30-ghz.html. Accessed: 2024-5-12.
[15] 2024. Intel® Xeon® Platinum 8460H Processor. https://ark.intel.com/content/www/us/en/ark/products/231744/intel-xeon-platinum-8460h-processor-105m-cache-2-20-ghz.html. Accessed: 2024-5-12.
[16] 2024. Mastering Dynamic Resource Allocation in Apache Spark. https://www.sparkcodehub.com/spark-dynamic-allocation. Accessed: 2024-5-12.
[17] 2024. MySQL Documentation. https://dev.mysql.com/doc/. Accessed: 2024-3-1.
[18] 2024. NUMA Balancing. https://www.kernel.org/doc/html/latest/admin-guide/sysctl/kernel.html#numa-balancing. Accessed: 2023-6-19.
[19] 2024. numactl(8) — Linux manual page. https://man7.org/linux/man-pages/man8/numactl.8.html. Accessed: 2024-5-12.
[20] 2024. Oracle Exadata Database Machine X8-2. https://www.oracle.com/technetwork/database/exadata/exadata-x8-2-ds-5444350.pdf. Accessed: 2023-6-19.
[21] 2024. Overview of ClickHouse Architecture. https://clickhouse.com/docs/en/development/architecture. Accessed: 2024-3-1.
[22] 2024. Sapphire Rapids: Golden Cove Hits Servers. https://chipsandcheese-com.translate.goog/2023/03/12/a-peek-at-sapphire-rapids/?_x_tr_sl=en&_x_tr_tl=it&_x_tr_hl=it&_x_tr_pto=sc. Accessed: 2024-5-12.
[23] 2024. TPC Benchmark™ DS. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v3.2.0.pdf. Accessed: 2024-3-1.
[24] 2024. TPC Benchmark™ H. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf. Accessed: 2024-3-1.
[25] 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 (2020), 3204–3216.
[26] Hossein Ahmadvand, Tooska Dargahi, Fouzhan Foroutan, Princewill Okorie, and Flavio Esposito. 2021. Big Data Processing at the Edge with Data Skew Aware Resource Allocation. *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* (2021), 81–86.
[27] 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).
[28] 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. *SIGMOD/PODS '22: International Conference on Management of Data* (2022).
[29] Claude Barthels, Ingo Müller, Timo Schneider, Gustavo Alonso, and Torsten Hoefler. 2017. Distributed join algorithms on thousands of cores. *Proceedings VLDB Endowment* 10, 5 (2017), 517–528.
[30] Giuseppe Baruffa, Mauro Femminella, Matteo Pergolesi, and Gianluca Reali. 2020. Comparison of MongoDB and Cassandra Databases for Spectrum Monitoring As-a-Service. *IEEE Transactions on Network and Service Management* 17 (2020), 346–360.
[31] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. 2015. The end of slow networks: It's time for a redesign. (2015).
[32] Peter Boncz, Angelos-Christos Anatiotis, and Steffen Kläbe. 2018. JCC-H: Adding join crossing correlations with skew to TPC-H. (2018), 103–119.
[33] Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K Aguilera. 2017. Black-box Concurrent Data Structures for NUMA Architectures. *ASPLOS '17: Architectural Support for Programming Languages and Operating Systems* (2017).
[34] Jack Chen, Samir Jindel, Robert Walzer, Rajkumar Sen, Nika Jimsheleishvilli, and Michael Andrews. 2016. The MemSQL query optimizer. *Proceedings VLDB Endowment* 9, 13 (2016), 1401–1412.
[35] Grigory Chirkov and David Wentzlaff. 2023. Seizing the Bandwidth Scaling of On-Package Interconnect in a Post-Moore's Law World. *Proceedings of the 37th International Conference on Supercomputing* (2023).
[36] 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.
[37] Yinxiao Feng and Kaisheng Ma. 2022. Chiplet actuary: a quantitative cost model and multi-chiplet architecture exploration. *Proceedings of the 59th ACM/IEEE Design Automation Conference* (2022).
[38] Sérgio Fernandes and Jorge Bernardino. 2014. What is BigQuery? *Proceedings of the 19th International Database Engineering & Applications Symposium* (2014).
[39] Pouya Fotouhi. 2019. Enabling scalable chiplet-based uniform memory architectures with silicon photonics. *Proceedings of the International Symposium on Memory Systems* (2019).
[40] 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.
[41] 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. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015).
[42] John L. Hennessy and David A. Patterson. 2019. A new golden age for computer architecture. *Commun. ACM* 62 (2019), 48 – 60.
[43] 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 (2008), 1496–1499.
[44] T Kiefer, B Schlegel, and W Lehner. 2013. *Experimental Evaluation of NUMA Effects on Database Management Systems.* BTW.
[45] Andi Kleen. 2005. A numa api for linux. *Novel Inc* (2005).
[46] B Lepers, V Quéma, and A Fedorova. 2015. *Thread and Memory Placement on NUMA Systems: Asymmetry Matters." USENIX Annual Technical Conference.*
[47] Mian Liao, Daniel H. Zhou, P. Wang, and Minjie Chen. 2023. Power Systems on Chiplet: Inductor-Linked Multi-Output Switched-Capacitor Multi-Rail Power Delivery on Chiplets. *2023 Fourth International Symposium on 3D Power Electronics Integration and Manufacturing (3D-PEIM)* (2023), 1–7.
[48] Ravi Mahajan, Robert Sankman, N. Patel, Dae woo Kim, Kemal Aygun, Zhiguo Qian, Yidnekachew S. Mekonnen, Islam A. Salama, Sujit Sharan, Deepti Iyengar, and D. Mallik. 2016. Embedded Multi-die Interconnect Bridge (EMIB) – A High Density, High Bandwidth Packaging Interconnect. *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)* (2016), 557–565.
[49] Michael Mattioli. 2021. Rome to Milan, AMD Continues Its Tour of Italy. *IEEE Micro* 41, 4 (2021), 78–83.
[50] John D. McCalpin. 1991-2007. *STREAM: Sustainable Memory Bandwidth in High Performance Computers.* Technical Report. University of Virginia, Charlottesville, Virginia. A continually updated technical report. http://www.cs.virginia.edu/stream/.
[51] John D. McCalpin. 2021. Mapping Core and L3 Slice Numbering to Die Locations in Intel Xeon Scalable Processors. (2021).
[52] John D. McCalpin. 2023. Bandwidth Limits in the Intel Xeon Max (Sapphire Rapids with HBM) Processors. *ISC Workshops* (2023).
[53] 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.
[54] 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.
[55] Samuel D. Naffziger, Kevin M. Lepak, Milam Paraschou, and Mahesh Subramony. 2020. 2.2 AMD Chiplet Architecture for High-Performance Server and Desktop Products. *2020 IEEE International Solid- State Circuits Conference - (ISSCC)* (2020), 44–45.
[56] 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.
[57] 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.
[58] Danica Porobic, Ippokratis Pandis, Miguel Branco, Pınar Tözün, and Anastasia Ailamaki. 2012. OLTP on Hardware Islands. *Proc. VLDB Endow.* 5 (2012), 1447–1458.
[59] Tudor-Ioan Salomie, Ionut Emanuel Subasu, Jana Giceva, and Gustavo Alonso. 2011. Database engines on multicores, why parallelize when you can distribute? *European Conference on Computer Systems* (2011).
[60] Robert Schöne, Thomas Ilsche, Mario Bielert, Markus Velten, Markus Schmidl, and Daniel Hackenberg. 2021. Energy Efficiency Aspects of the AMD Zen 2 Architecture. *2021 IEEE International Conference on Cluster Computing (CLUSTER)* (2021), 562–571.
[61] 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.

[62] Debendra Das Sharma, Gerald Pasdast, Zhiguo Qian, and Kemal Aygun. 2022. Universal Chiplet Interconnect Express (UCIe): An Open Industry Standard for Innovations With Chiplets at Package Level. *IEEE Transactions on Components, Packaging and Manufacturing Technology* 12 (2022), 1423–1431.

[63] David Suggs, Mahesh Subramony, and Dan Bouvier. 2020. The AMD "Zen 2" Processor. *IEEE Micro* 40 (2020), 45–52.

[64] Markus Velten, Robert Schöne, Thomas Ilsche, and Daniel Hackenberg. 2022. Memory Performance of AMD EPYC Rome and Intel Cascade Lake SP Server Processors. *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering* (2022).

[65] 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.