Post-Moore's Law Fusion: High-Bandwidth Memory, Accelerators, and Native Half-Precision Processing for CPU-Local Analytics

Viktor Sanca[1,*], Anastasia Ailamaki[1,2,†]

[1]EPFL, Lausanne, Switzerland
[2]Google, Sunnyvale, USA

Abstract

Modern data management systems aim to provide both cutting-edge functionality and hardware efficiency. With the advent of AI-driven data processing and the post-Moore Law era, traditional memory-bound scale-up data management operations face scalability challenges. On the other hand, using accelerators such as GPUs has long been explored to offload complex analytical patterns while trading-off data movement over an interconnect. GPUs typically provide massive parallelism and high-bandwidth memory, while CPUs are near-data processors and coordinators that are often memory-bound.

In this work, we provide a first look over an architecture that mixes the best of the CPU and GPU world: high-bandwidth memory (HBM), core-local accelerators for matrix multiplications (AMX), and native half-precision data processing inside 4th Generation Intel Xeon Scalable processors known as Sapphire Rapids. We analyze the system, provide an overview of its hierarchical NUMA architecture, focus on individual components, and explore their interplay and how they impact the traditional DRAM bandwidth wall on typical data access patterns and novel AI-DB interactions of vector data processing.

Keywords

Modern Hardware, Data Analytics, CPU, NUMA, HBM, Core-Local Accelerators, AI for DB, Vector Data Processing

1. Introduction: Evolving Scalability

Data management has long focused on bringing both performant and efficient, hardware-conscious solutions and adapting to the ways and the requirements the data has to be analyzed to be useful and extract insights. Moore's law and the improvements in processing technology allowed scaling-down chips and circuits, leading to moving the peripheral interconnect (PCIe) and memory controller (MC) from the Northbridge (1) to being integrated with the CPU (2), depicted in Figure 1. This CPU evolution allowed for avoiding the Front-Side Bus data transfer overheads. Along with the advent of high-capacity main memory and multi-socket, multi-core systems sparked optimizations for shifting the bottleneck from being IO to memory-bound. This was reflected in the emerging database system designs that used vectorization [1] and compilation [2, 3] in contrast to the Volcano-style iteration model to reduce the overheads of previous systems that were appropriate and tuned to the previously available hardware systems and memory hierarchies [4].

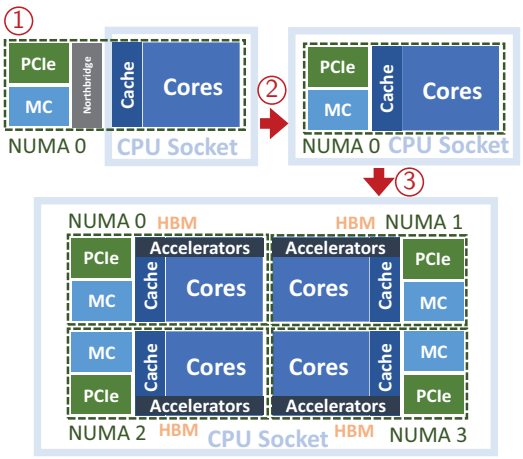

Figure 1: Moore's Law effect: 1) Northbridge scaled to 2) Integrated memory controller scaled to 3) Multi-die chip design. Instead of only being present in multi-socket systems, NUMA evolved to include socket-local granularity.

Similarly, when GPUs became more prevalent and driven by the machine learning community, data manage-

Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23) — Workshop on Accelerating Analytics and Data Management Systems (ADMS'23), August 28 - September 1, 2023, Vancouver, Canada
*Corresponding author.
†Work done entirely at EPFL.
viktor.sanca@epfl.ch (V. Sanca); anastasia.ailamaki@epfl.ch (A. Ailamaki)
https://viktorsanca.com (V. Sanca)
0000-0002-4799-8467 (V. Sanca); 0000-0002-9949-3639 (A. Ailamaki)

ment research focused on using them as data processing accelerators [5, 6, 7], being especially useful for computationally and random-access-heavy operators such as joins [8]. The advent of accelerators has resulted in systems with computational, memory, and interconnect heterogeneity, consequentially resulting in system designs aimed to reduce their complexity for practical use [9, 10]. With memory-bound analytics, the next goal was reducing the effects of new bottlenecks, such as PCIe interconnects that are required to transfer the data from the main memory to the limited capacity, high-bandwidth GPU memory [11], while also studying and utilizing faster available interconnects [12]. Using all the available resources effectively leads to fast and efficient data processing, which requires system designs that adapt to the available hardware and build appropriate abstractions to make it practical.

From another perspective, data management systems not only adapt and drive the hardware trends but are also the backbone of making sense of, producing insights, and deriving value from the increasingly high data volume and various sources. The ways the data is processed change with advancements in fields such as data mining and machine learning. While relational data is the long-established way to store and analyze data in many use cases, with increasingly high amounts of data, machine learning models have become more powerful for various previously human-driven processing, such as object recognition or semantic analysis. With increasingly practical and desirable models, machine learning (and inference) has become part of data analytics, both in terms of machine learning for databases such as learned indexes [13] or reinforcement learning for query optimization [14], as well as optimizing databases for machine learning, for example using tensor computation runtimes [15].

Furthermore, With more recent findings in the domain of ML, such as the Transformer model architectures [16] and data embeddings, a class of multi-modal models has presented state-of-the-art solutions for processing context-rich data. Such novel use cases and findings dictate the next important way to support extracting value and processing data. This is noticeable as machine learning and inference are becoming increasingly available in data management engines such as BigQuery ML, IBM DB2 Data Insights, Azure Data Studio/SQL Server, Oracle Machine Learning, Amazon Aurora Machine Learning, and others. In particular, extracting and learning embeddings from relational tables [17, 18] and using external models to enrich the data [19, 20] are some of the ways of incorporating learning-driven data transformation. This presents novel challenges in building declarative, optimizable, and hardware-conscious hybrid model-relational engines [19] and optimizing them to modern hardware capabilities.

In conjunction, novel applications and the post-Moore's law era are bringing towards horizontal scaling in CPUs, with multi-die chip designs (3) (Figure 1). While this design was already present in Intel Xeon Phi Knights Landing CPUs [21], more recently, `chiplet`-based designs have become mainstream with vendors such as AMD [22, 23], Apple [24], and yet again Intel [25]. In such designs, rather than being a monolithic die, the CPU is composed of multiple individual `chiplets` interconnected in a single package.

Novel hardware brings novel challenges and opportunities for data management system design. For this reason, we study the performance characteristics and present an initial evaluation of Intel Sapphire Rapids (4^{th} Generation Xeon) CPUs and:

- present and describe the novel multi-die chip architecture and the interplay of individual components in Section 2,
- analyze individually the effects of High-Bandwidth Memory (HBM) on access patterns in Section 3, the use of hardware-supported half-precision intrinsics in Section 4, and the novel on-core matrix-accelerators (AMX) in Section 5,
- evaluate the combined effect of individual components on a vector-heavy workload motivated by recent embedding methods in Section 6.

The post-Moore's law world resulted in CPU designs to which data analytics systems must be tailored for effective and efficient use. In a unique fusion of features between high-bandwidth memory to tackle the memory bandwidth wall and half-precision analytics and accelerators for ML workloads in a hierarchical NUMA scaling package, we explore their individual and combined effects.

2. Design: Scaling and Element Fusion

The evolution of the scalability of physical CPU design (Figure 1) exposes bottlenecks or primitives that provide challenges and opportunities for hardware-conscious software system design. While miniaturization allowed fitting more components on a die and allowed the memory and peripheral controller to be integrated (2) rather than separated via Northbridge (1), the same trends have long been known to slow down, as predicted by Moore's law. We consider a NUMA region a unit where processors can uniformly access memory, typically present in a single CPU socket. To scale, adding more cores entails adding chiplets representing individual NUMA regions with memory and peripheral links (3). This creates hierarchical NUMA regions inside the same socket by

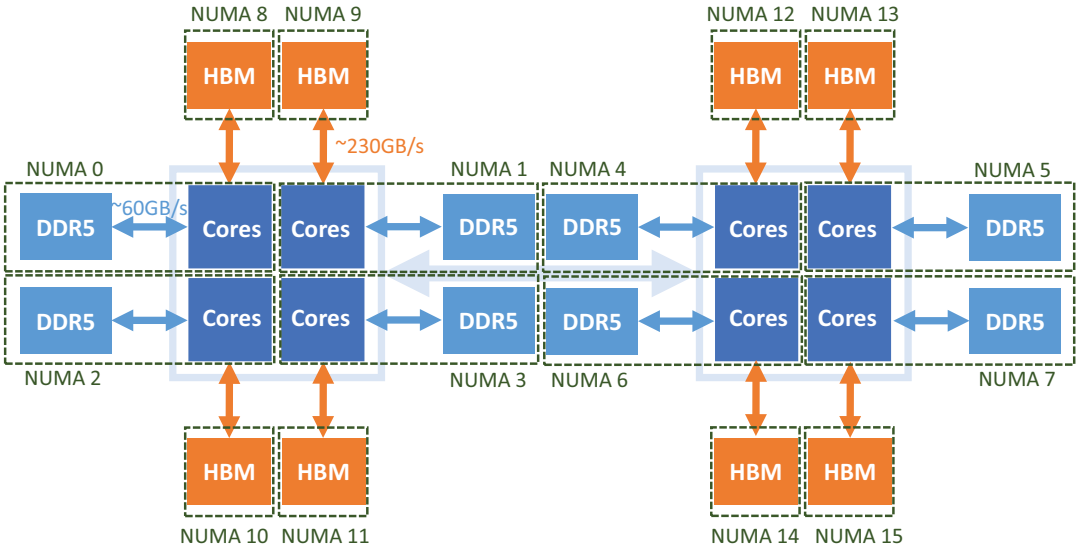

Figure 2: Dual socket setup: Multi-die chip design has 4 NUMA regions in a single socket, each with DDR5 and HBM2. Using Sub-NUMA Clustering 4 (SNC-4), HBM is exposed as a separate NUMA region to the OS, resulting in 8 NUMA regions per socket, 16 in total for a dual-socket setup.

adding more chips instead of further reducing their size. Comparatively, this NUMA phenomenon was seen and studied in Intel Xeon Phi Knights Landing (KNL) processors [26] about 5 years ago. However, the hierarchical NUMA design and chiplets are more critical nowadays for scaling, with major vendors providing their solutions. While AMD chiplet-based CPUs focus on scalability with uniform cores [23] to provide increasingly more cores, memory, and PCIe bandwidth, Intel's approach is to extend the design with accelerators, novel intrinsics, and high-bandwidth memory [25].

Instead of simply adding more NUMA regions and conversely approaching this new hardware design with adaptive NUMA placement strategies [27], Intel Sapphire Rapids (Figure 2) introduces further hardware changes to this design space that is tailored to contemporary use-cases and workloads. In particular, we present the schema of a dual-socket Intel Xeon CPU MAX 9480 [28] with 56 physical cores and 64GB HBM per socket. Every NUMA region has 16GB of HBM and 14 physical cores, and 28 with hyperthreading.

First, High-Bandwidth Memory (HBM) modules are added to each NUMA region, offering lower capacity than main memory (in this case, 16GB per region) but significantly higher bandwidth. For example, in comparison to DDR5 memory that reaches ~ 60 GB/s per region, HBM2 [29] reaches ~ 230 GB/s. In aggregate, HBM per socket reaches nearly 1 TB/s, with DRAM at 240 GB/s. HBM allows addressing the long-standing memory wall problem [30], and the technology is already deployed in high-end GPUs. With memory-bound analytics and with many random-access analytical patterns that are a natural fit for GPUs such as hash joins [8], higher bandwidth combined with many cores can offer a different tradeoff, having the benefit of main memory locality. This has also been demonstrated in the analysis of using the Intel Xeon Phi Knight's Landing CPU over TPC-H queries [4]. In contrast to GPUs and remote accelerators that have interconnects that reduce the transfers to a fraction of the available local memory bandwidth, CPU-integrated HBM effectively allows near-memory processing [31]. Since Intel Xeon KNL had an earlier variant of HBM and chiplet-based architecture that is similar to Figure 2, there is existing applicable research on the topics of optimizing for chiplet-based hierarchical NUMA and HBM [32, 33, 26]. We study this behavior in Section 3.

However, the next important distinction is that half-precision integer (`int8`) and floating point (`fp16`) operations are supported through hardware intrinsics, including a brain-float format (`bf16`), common in machine learning. Beyond being optimized for machine learning workloads or approximate query processing [34], this allows changing the relative throughput and memory footprint for general analytics and designing data structures that are more compact and make use of full hardware processing support. Half-precision types effectively enable twice the throughput of the single-precision types

and, in conjunction with high-bandwidth memory, may change the access-pattern behavior and shift memory- to compute-bound workloads. We explore diverse access patterns and these novel tradeoffs that come with combining HBM and half-precision types in Section 4.

Finally, in addition to the high-bandwidth memory and native half-precision intrinsics, core-local accelerators introduce processing heterogeneity and workload specialization. In conjunction with homogeneous compute cores, Intel Sapphire Rapids CPU comes with specialized components such as tile registers and matrix multiplication units (AMX) aimed at machine learning workloads, data encryption and compression accelerators (QAT), and data streaming accelerator (DSA) to provide hardware acceleration and offload the CPU. In particular, we focus on AMX and matrix operations as support for machine learning and vector-embedding-related analytical tasks. Currently, only a single AMX accelerator is available, being Tile matrix multiply unit (TMUL) that processes matrix data stored in 8x1KB register files called tile registers. As accelerators allow offloading tasks from CPU cores to specialized hardware and achieve better efficiency, we study briefly the impact of AMX in comparison to CPU-only execution in Section 5.

Overall, this CPU architecture introduces a novel design space and opportunities for changing existing tradeoffs by fusing high bandwidth memory, accelerators, and half-precision processing in a single package. This extends the memory hierarchy as it enables fast access to the data in HBM, in addition to the main memory, or NVMe drives via PCIe links that allow comparable aggregate bandwidth [35]. This is especially important when modern analytics start to use embeddings and tensor data formulations [17, 19, 15], where CPU-local accelerators can better support not only machine learning workloads but equally data analytics by having fast and immediate access to the data with better-tailored operations, such as half-precision processing or accelerating matrix multiplications, that is lightweight enough not to necessitate GPU processing and crossing the comparatively slower interconnect.

3. High-Bandwidth Memory (HBM)

We start with evaluating the impact of high-bandwidth memory (HBM) compared to the available DRAM (DDR5). We ensure that all the memory allocations and cores are pinned to desired NUMA nodes using `numactl` utility. We mainly focus on isolating individual NUMA regions, particularly core-local DRAM and HBM (e.g., NUMA 0 and NUMA 8 in Figure 2). As noted, in this setup, HBM is connected to the cores, avoiding interconnects such as PCIe, allowing fast data transfer, which we first evaluate.

NUMA Node	0	1	2	3	4	5	6	7
0	60.04	60.69	60.16	60.25	58.56	58.69	59.24	58.52
1	60.48	59.72	59.98	60.34	58.54	58.56	59.33	58.53
2	60.34	60.39	59.47	60.58	58.61	58.58	59.48	58.55
3	60.31	60.56	60.39	59.88	58.51	58.54	59.50	58.59
4	59.55	58.67	58.27	58.89	59.61	60.32	60.47	60.06
5	59.32	58.56	58.30	58.84	60.37	59.61	60.39	60.20
6	59.27	58.54	58.49	58.76	60.19	60.03	60.08	60.47
7	59.23	58.65	58.54	58.85	60.05	60.13	60.65	59.42

Figure 3: DRAM Bandwidth (GB/s) between NUMA regions.

The server is configured using Sub-NUMA clustering 4 (SNC-4), as in Figure 2. Each socket has 56 physical cores, equally partitioned over four tiles interconnected using Intel's embedded multi-die interconnect bridge (EMIB) technology in a mesh. This yields 14 physical cores in a tile (NUMA region), with direct HBM (separate NUMA region) and DRAM access, exposed in the finest available granularity to the OS.

3.1. Bandwidth and Latency

With data movement becoming increasingly expensive, we evaluate the bandwidth and latency characteristics of DRAM and HBM using the Intel Memory Latency Checker [36].

3.1.1. DRAM

Figure 3 shows the DRAM bandwidth matrix, indicating the available bandwidth from/to a given NUMA node. As each NUMA region has 60GB/s DRAM bandwidth, individually, cross-NUMA transfers can happen at the full bandwidth. Still, we note that while the fine-grained, hierarchical NUMA allows for better control, it can also introduce more complex interference patterns.

While the bandwidth between different fine-grained NUMA regions remains practically unchanged, Figure 4 indicates the expected increase in latency when performing cross-tile inter-socket (EMIB interconnect) and cross-socket (UPI interconnect) transfers. Socket-local regions are 0-3 and 4-7 quadrants (EMIB), while the rest are corresponding UPI transfers.

Numa node	0	1	2	3	4	5	6	7
0	93.9	104.1	111.7	121.6	226.3	233.4	235.7	238
1	103.9	95.4	119.3	113.1	227.4	234.6	236.2	238.3
2	112	120.8	94.8	108.8	231.9	236.8	237.2	238.5
3	118.6	109.1	103	95.6	233.5	237.2	237.4	238.5
4	229.1	233.6	234.2	240.1	94	104.1	116.2	120.4
5	230.3	234.3	235.3	240.6	101.3	93.3	120.7	112
6	232.7	236	237.4	240.2	111.8	119.8	97.3	106.4
7	234	236.1	237.6	240.6	116	107.3	104.8	96.8

Figure 4: DRAM Latency Matrix (ns) between NUMA regions over two sockets. Socket-local regions are 0-3 and 4-7 quadrants (EMIB interconnect), while the rest are UPI transfers.

Local Node	0	1	2	3	Local Node	4	5	6	7
NUMA Node	8	9	10	11	NUMA Node	12	13	14	15
0	220.92	143.62	126.39	122.18	4	221.67	143.93	126.45	122.08
1	144.27	223.97	122.18	126.11	5	144.70	224.34	122.17	125.96
2	126.16	121.85	224.74	141.64	6	126.26	121.95	224.69	141.65
3	122.46	126.79	142.36	224.60	7	122.29	126.82	142.30	226.52

Figure 5: HBM bandwidth matrix (GB/s), the socket-local measurement for each CPU socket with 4 CPU tiles/NUMA regions.

3.1.2. HBM

Next, we evaluate the HBM bandwidth and latency characteristics similarly. Figure 5 indicates the available bandwidth between socket-local NUMA regions. There is almost a 2x performance impact of the locality. Thus judicious data placement and locality maintenance are imperative to achieve the full bandwidth. This contrasts the DRAM bandwidth matrix (Figure 3), where the impact is negligible or nonexistent. Socket-local NUMA regions are 0-3 and 4-7, with corresponding HBM NUMA regions of 8-11 and 12-15 respectively. Figure 6 summarizes the relative bandwidth speedup (in percentages) of HBM over DRAM in the corresponding NUMA regions.

Local Node	0	1	2	3	Local Node	4	5	6	7
Numa node	8	9	10	11	Numa node	12	13	14	15
0	367.95	236.66	210.09	202.79	4	371.84	238.59	209.12	203.27
1	238.55	375.02	203.71	209.00	5	239.69	376.35	202.29	209.23
2	209.06	201.77	377.88	233.80	6	209.77	203.14	374.01	234.23
3	203.05	209.37	235.72	375.08	7	203.65	210.90	234.60	381.25

Figure 6: The bandwidth speedup matrix of HBM over DRAM (%), the socket-local measurement for each CPU socket with 4 CPU tiles/NUMA regions.

Local Node	0	1	2	3	Local Node	4	5	6	7
Numa node	8	9	10	11	Numa node	12	13	14	15
0	120.4	126.1	134.6	146.2	4	120.9	128	133.3	147
1	127.1	122.7	143.7	133.1	5	125.7	121.5	143.3	136
2	133.3	144	120.7	125.1	6	131.8	144.2	120.9	128.7
3	142.6	132	125.4	120.8	7	141.7	132.1	124.6	122.5

Figure 7: HBM latency matrix (ns), the socket-local measurement for each CPU socket with 4 CPU tiles/NUMA regions.

On the other hand, the socket-local HBM module latencies (Figure 7) indicate little to no impact due to the socket local NUMA and HBM mesh enabled by Intel EMIB Packaging Technology [37], which is the socket-local interconnect of all the cores and multi-chip packaging components. Figure 8 summarizes the relative latency slowdown (in percentages) of accessing HBM over DRAM by cores in the indicated NUMA regions.

The available NUMA granularity using SNC-4 allows better resource and workload control at the expense of more complex placement and potential interference between similar memory and HBM bandwidths. Less granular exposure of cores is possible using Quad mode (2S) or HBM using Caching or HBM-only mode.

3.2. Data Access Patterns

With HBM becoming a CPU-local, high-bandwidth, and low-latency part of the memory hierarchy rather than separated by an interconnect, it is a candidate for use in a traditional sense where sequential and random access patterns appear in various workloads.

We abstract out three access patterns that bind the

Local Node	0	1	2	3	4	5	6	7
Numa node	8	9	10	11	12	13	14	15
0	28.22	21.13	20.50	20.23	1.68	0.60	0.68	1.60
1	22.33	28.62	20.45	17.68	1.58	0.77	0.89	1.85
2	19.02	19.21	27.32	14.98	0.34	0.34	1.01	2.31
3	20.24	20.99	21.75	26.36	0.47	0.59	1.26	2.56
4	1.88	0.81	1.67	0.54	28.62	22.96	14.72	22.09
5	1.61	0.90	1.44	0.54	24.09	30.23	18.72	21.43
6	1.16	0.93	0.84	1.08	17.89	20.37	24.25	20.96
7	0.81	1.14	0.97	1.04	22.16	23.11	18.89	26.55

Figure 8: The latency slowdown matrix of HBM over DRAM (%), the socket-local measurement for each CPU socket with 4 CPU tiles/NUMA regions.

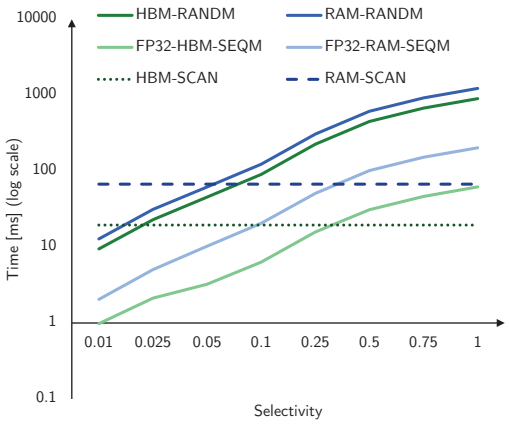

Figure 9: DRAM and HBM access pattern characteristics over FP32 values: full scan (SCAN), random access (RANDM), and sequential scan with indirection (SEQM) using all CPU-tile-local threads (28).

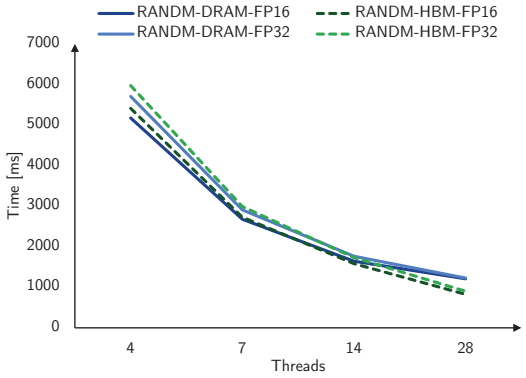

Figure 10: DRAM and HBM access pattern characteristics over FP32 and FP16 values for random access (RANDM), varying threads. HBM continues scaling using hyper-threads beyond the initial higher latency (14 physical to all 28 logical cores).

decision of paying the random access cost versus initiating a full-bandwidth scan [38]. For that reason, we evaluate three access patterns in Figure 9 for both HBM and DRAM to find an appropriate sweet spot between:

1. sequential scan (SCAN), that has the benefit of using the full available bandwidth,
2. random access (RANDM), which we simulate by generating random indexes to probe the original data with,
3. sequential access with probing (SEQM), in which we generate an index column to probe the original data; however, this column is fully sorted, and we pay for the indirection cost as an ideal probing cost, such as in late materialization approaches, with the benefit of partially scanning the data if possible.

We run the experiment using a single FP32 column with 1 billion tuples and generate an integer index column randomly and sequentially for RANDM and SEQM cases, which are of reduced size depending on the indicated selectivity factor. For HBM experiments, we bind the allocations to NUMA node 8, and for DRAM experiments, to NUMA node 0. Computation remains on NUMA node 0, with 14 physical/28 logical cores available.

Despite the higher bandwidth, relative data access characteristics of DRAM versus HBM remain similar but accordingly shifted. Therefore, the tradeoff shifts toward the sequential option when selecting random or sequential access. We also note that the general overheads of random access are also reduced, allowing faster random access pattern processing than DRAM.

We note that the evaluation in Figure 9 uses all the available threads (28 total, 14 physical and 14 hyper-threads). The RAND access patterns are expected to be lower using HBM according to the latency slowdown matrix (Figure 8), and we study this through sensitivity analysis to the number of threads. While the effects of higher latency are visible using fewer cores, there is a crossing point where scalability continues in the case of HBM compared to DRAM at 14 threads (Figure 10).

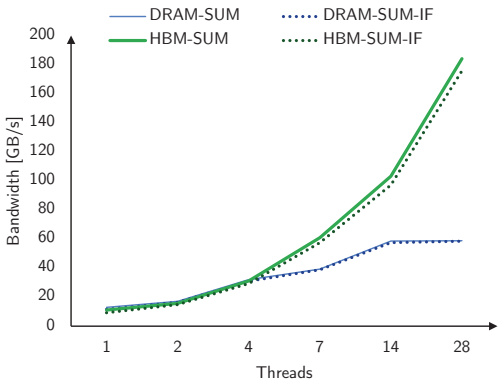

Figure 11: Simple aggregation on DRAM and HBM, sequential (SUM) and branching (SUM-IF).

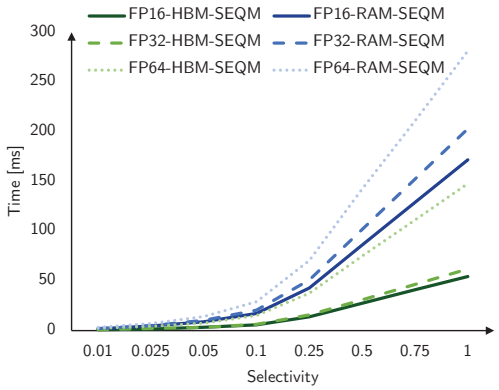

Figure 12: DRAM and HBM access pattern characteristics over FP16, FP32, and FP64 values using sequential scan with indirection (SEQM).

3.3. Value Aggregation

Besides data access and movement, the practical implication behind HBM is that it, for now, breaks the existing DRAM bandwidth wall.

Figure 11 illustrates this phenomenon, where DRAM workload plateaus at 60GB/s (which is the NUMA-local available DDR5 bandwidth), HBM-local data continues scaling and is not bound by memory. Both branching and sequential algorithms were sustained in either DRAM or HBM case. We use two workloads, one with a pure sequential summation of elements (SUM), and one with a predicate that introduces branching (SUM-IF).

4. Half-Precision Computation

While half-precision numerical support is common in GPUs and for machine learning and inference workloads, CPUs have typically not supported this with hardware intrinsics. With support for brain float 16 (BF16), half-precision floating point (FP16), and INT8 data types, higher processing throughput is expected due to more values fitting per cache line and in registers for processing. Such data formats have a lower memory footprint, and higher throughput is expected compared to full (FP32) and double (FP64) precision types. This consequently allows the designing of data structures and algorithms that can more flexibly use available instructions and data layouts.

It is instructive to mention that kernel and compiler support is necessary for using half-precision types. We use Rocky Linux 9.1 (Blue Onyx) with Linux 6.3.1-1 kernel and Clang 16, implement all our prototypes, and experiment using C++. Standard practices of aligned memory allocation have been followed, including a thread pool with pinned cores and controlling the affinities using the numactl utility.

4.1. Throughput and Access Patterns

We briefly revisit the previously explained SEQM pattern, extending it with FP16 and FP64 experiments in Figure 12. While in highly selective queries, the effect of using a wrong (larger than needed) data type is not large, the benefit of using HBM can be significantly diminished when processing most of the data. Hardware intrinsics now allow making even more fine-grained decisions between FP32 and FP16 data types, in contrast to prior support of FP32 and FP64 only.

4.2. BF16 Conversion

An important factor to consider with half-precision data types using current Intel Sapphire Rapids processor intrinsics is that BF16 is meant only as an intermediate data type for computation. Accordingly, there are no load and store instructions, just conversion from other data types and computations. In particular, two BF16 values can be created from a single FP32 value and then processed, for example, using a dot-product intrinsic.

The dot product is a common operation for vector and matrix data processing, for example, in computing the cosine vector similarity. We, therefore, explore the practicality of using BF16 as an intermediate format and evaluate the conversion speed in Figure 13. The task consists of computing a dot product and summing it over two columns of data containing elements of different data types (FP64, FP32, FP16). We include the experiment where starting from FP32 data using DRAM only, intermediate BF16 data is converted, and a specialized

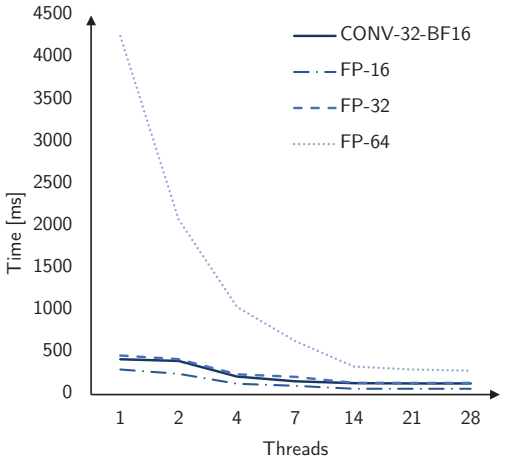

Figure 13: Pairwise dot product and sum between two columns, different data types, including BF16, DRAM only.

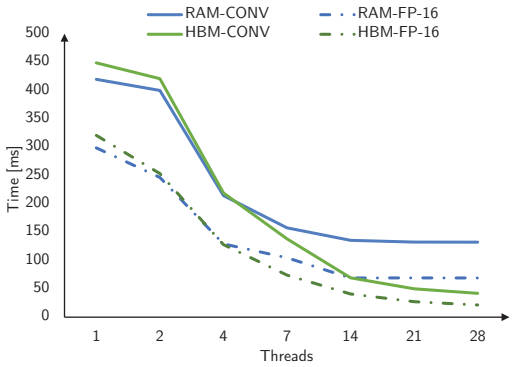

Figure 14: Pairwise dot product and sum between two columns, FP16 and BF16 conversion on HBM and DRAM.

intrinsic is used for dot product computation. We use all the NUMA-local cores (28 in total using hyperthreading).

While the computation consists only of the multiplication and addition of elements, the computational overhead of the FP64 data type is significantly higher due to data movement and computation requirements. Furthermore, we can notice that the BF16 execution time is similar to FP32. This is due to scanning the data in FP32 format and not benefiting from reduced data access, only from faster computations. Still, this overhead is comparably insignificant in more complex computations, such as in computationally intensive machine learning operations.

4.3. HBM + Half-Precision

Still, with the availability of HBM, we are interested in if higher bandwidth can further offset the FP32 to BF16 conversion cost and bring the processing time closer to one of FP16. We keep the setup but run the experiments on local DRAM and HBM separately, using all 28 tile-local threads.

Figure 14 indicates that the best computation time is indeed when FP16 is used, which is allocated as such, benefiting from lower data access and computational cost. This is true for both DRAM and HBM. However, when able to use more than 14 threads and consequently more HBM bandwidth, the FP32 access cost diminishes, and the conversion pays off even compared to FP16 in DRAM. Thus, breaking the bandwidth wall between DRAM and HBM is also apparent at the 4 threads point, briefly demonstrating the importance of both half-precision data types and high bandwidth memory.

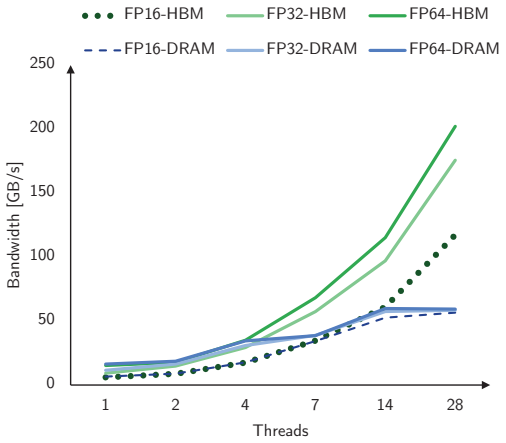

Figure 15: SUM-IF using different precision, DRAM and HBM

This is equally apparent in Figure 15, where we rerun the SUM-IF query using different precision floating points in HBM and DRAM. So long as the computations can sustain the increased bandwidth, it will be useful and break through the existing 60GB/s NUMA-local DRAM bandwidth. Interestingly, corresponding data types have a similar bandwidth until reaching that wall with the number of threads required to transition between being compute and memory bound. Still, to fully benefit from added bandwidth and the impact of data type change, the workload must be able to consume the available bandwidth or initially suffer from data movement bottlenecks. In this case, the relative bandwidth of consuming the same number of tuples between FP16, FP32, and FP64 is not purely memory bound (i.e., the data consumption bandwidth does not scale linearly).

5. On-CPU Accelerators (AMX)

Finally, we analyze the last on-CPU building block that allows efficient data processing through hardware specialization. While other accelerators (QAT, DSA) are specialized for other tasks, we focus on Advanced Matrix Extensions (AMX) as a building block for modern analytical and machine learning workload acceleration over vector data types.

AMX introduces matrix multiplication intrinsics that operate over 8 tile registers of 1KB each, storing half-precision floats and integers and performing dot product operation. In contrast, traditional AVX-512 intrinsics also have dot product instructions specialized for half-precision computation, however, working on fewer data points that fit in standard registers.

5.1. Cores vs Accelerators

The natural question to ask in the presence of specialized hardware along a general-purpose core is where the line between the two lies. In particular, we want to answer how many regular cores one AMX accelerator replaces. We generate 1 million tuples at random, each tuple being a 512-dimensional vector. The given workload is to perform a cosine distance computation over all the tuples against a single 512-dimensional vector. To answer the question, we run AMX on a single thread exclusively while varying the threads for other data types and AVX-512-supported execution.

We first start with DRAM and HBM execution over different precision data types in Figure 16, similar to the experimental setup of Section 4.3.

The experiment is conclusive that care should be taken to use appropriate data type precision, as they significantly impact the execution time and efficiency. While HBM can offset some of that cost, the computation would still dominate.

In the case of DRAM, even in a single-threaded execution, AMX is almost as fast as all-core execution on FP16 and FP32 data types and faster than FP64, indicating high accelerator efficiency for the given task. This allows offloading other tasks to the available CPU cores.

When HBM is used, higher bandwidth allows moving the overhead of repeatedly loading the data to AVX-512 registers closer to AMX, and the crossing point indicates that, in this case, AMX replaces about 6 CPU cores, as shown in Figure 17. This enables more efficient resource utilization in the presence of suitable workloads and reduces the reliance on GPUs and slow data transfers for matrix-specialized operations. Still, data locality should be orchestrated carefully to benefit from the added specialized accelerators, as accessing remote data, even on an adjacent CPU tile (socket-local NUMA), reduces the efficiency to about replacing 3 CPU cores (Figure 18).

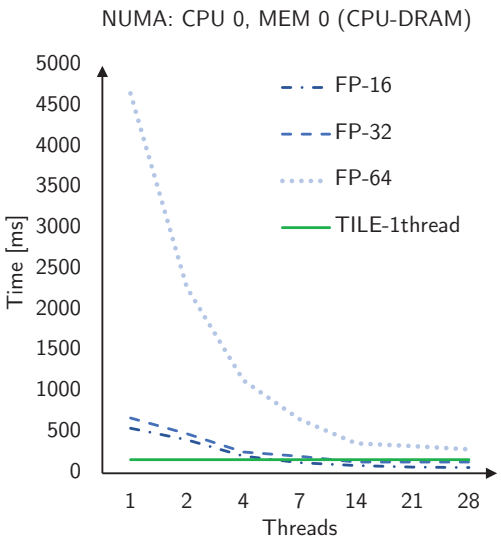

Figure 16: 1 thread AMX vs. AVX-512 and parallel execution on CPU-tile-local cores (up to 28 with hyperthreading) on NUMA node 0 and local DRAM on NUMA node 0.

6. Fusion: HBM + Half-Precision + AMX

At this point, we have individually presented the components, and now we analyze them together: HBM, native half-precision hardware support, and matrix-processing accelerators. For efficient use of hardware resources and with increasingly high heterogeneity, it is essential to evaluate the entire design space and component interactions. The experimental setup is as in Section 5, we generate 1 million tuples randomly, each tuple being a 512-dimensional vector. The given workload is to perform a cosine distance computation over all the tuples against a single 512-dimensional vector. We run AMX on a single thread exclusively while varying the threads for other data types and AVX-512-supported execution.

The most granular NUMA configuration (SNC-4) allows fine-grained data movement and access control over HBM, DRAM, and processor resources on sockets and tiles. However, certain workloads, such as performing matrix multiplication using FP64 datatype (Figure 19), result in workloads natively fully handled better by specialized accelerators, if at all possible to perform the conversion or approximate.

When there is a balance between data movement and computation (Figure 20), an equally balanced use between accelerators and general computational resources is possible, depending on particular optimization goals. The increased bandwidth of HBM helps in the data movement

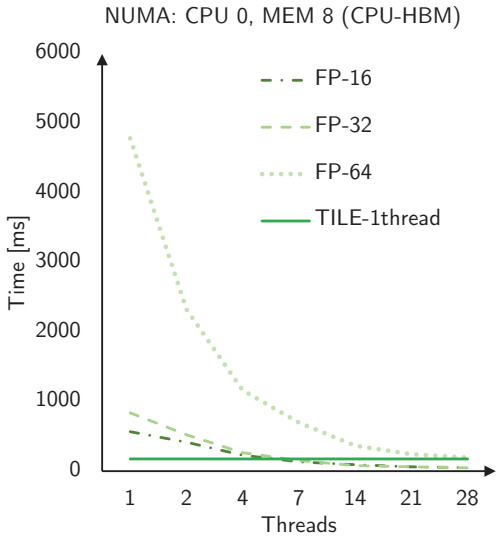

Figure 17: 1 thread AMX vs. AVX-512 and parallel execution on CPU-tile-local cores (up to 28 with hyperthreading) on NUMA node 0 and local HBM on NUMA node 8.

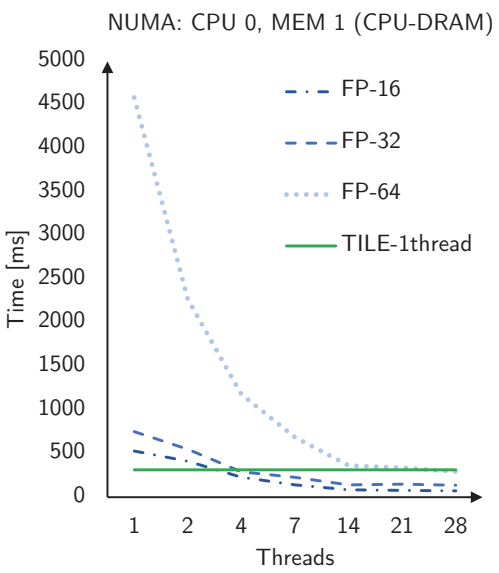

Figure 18: 1 thread AMX vs. AVX-512 and parallel execution on CPU-tile-local cores (up to 28 with hyperthreading) on NUMA node 0 and socket-local but tile-remote DRAM on NUMA node 1.

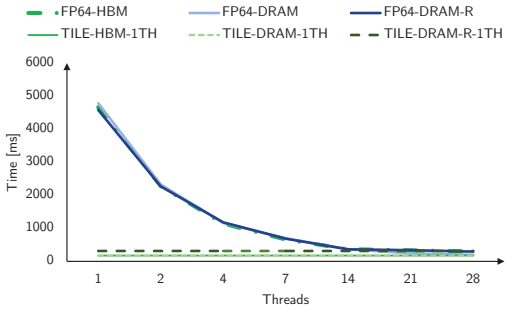

Figure 19: HBM, AMX, and FP64 combined, compared against using CPU-tile-local DRAM and socket-local but tile-remote DRAM (-R).

tasks and further reduces the previously memory-bound workload time.

Finally, using all the hardware characteristics: HBM, FP16, and AMX allows fine-tuning the particular use case to the available resources. When respecting NUMA locality, the traditional CPU cores are utilized the best, and half-precision types reduce the stress on the memory bandwidth and data movement. This allows using remaining resources, such as HBM and available cores for other parallel tasks, and pushes the global efficiency of the whole NUMA node in comparison to using a specialized AMX accelerator, as depicted in Figure 21.

The increased hardware accelerator and memory hierarchy heterogeneity provide opportunities and solutions to address the existing bottlenecks. On the other hand, careful orchestration and tailoring to the workload are necessary to benefit from the fine-grained features and their interactions. Finally, this also spans data placement and memory management problems, as HBM capacity is limited, as well as specializing the computation using correct data types or CPU-specific intrinsics.

7. Conclusion and Opportunities

The looming end of Moore's law has introduced novel CPU architectural solutions. We analyzed 4th Generation Intel Xeon Scalable processors known as Sapphire Rapids, as one proposed answer to continue CPU scalability. In particular, a combination of homogeneous cores, half-precision support, specialized accelerators, and high-bandwidth memory allows a unique fusion of features. Firstly, high-bandwidth memory (HBM) allows scaling the memory wall, accelerators (AMX) allow efficiency and alleviating general-purpose cores, and half-precision types allow tailoring the computation to modern workloads.

To use the underlying hardware efficiently, systems must be tuned to their characteristics. With a new design and tradeoffs in comparison to prior CPUs, our study on Sapphire Rapids is an initial starting point for hardware-

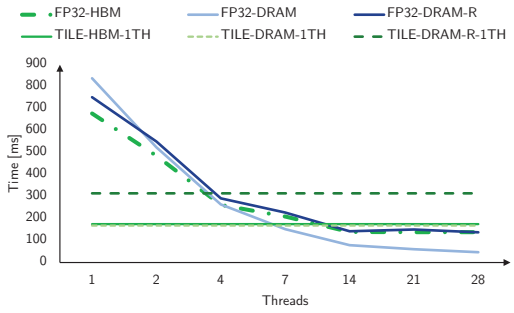

Figure 20: HBM, AMX, and FP32 combined, compared against using CPU-tile-local DRAM and socket-local but tile-remote DRAM (-R).

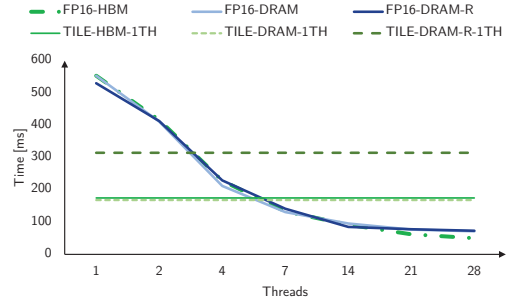

Figure 21: HBM, AMX, and FP16 combined, compared against using CPU-tile-local DRAM and socket-local but tile-remote DRAM (-R).

software codesign of future chiplet-based heterogeneous CPUs and processing units, and how they impact both traditional and novel data management challenges.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and detailed feedback that improved the paper. We also thank the Intel Developer Cloud staff for their support in facilitating access to hardware, especially the team at Intel DCAI Labs in Swindon, who generously provided access to the server equipped with the 4th Generation Intel Xeon Scalable CPU (Sapphire Rapids) for testing and evaluation.

References

[1] M. Zukowski, M. Van de Wiel, P. Boncz, Vectorwise: A vectorized analytical dbms, in: 2012 IEEE 28th International Conference on Data Engineering, IEEE, 2012, pp. 1349–1350.

[2] T. Neumann, Efficiently compiling efficient query plans for modern hardware, Proc. VLDB Endow. 4 (2011) 539–550. URL: http://www.vldb.org/pvldb/vol4/p539-neumann.pdf. doi:10.14778/2002938.2002940.

[3] P. Menon, A. Pavlo, T. C. Mowry, Relaxed operator fusion for in-memory databases: Making compilation, vectorization, and prefetching work together at last, Proc. VLDB Endow. 11 (2017) 1–13. URL: http://www.vldb.org/pvldb/vol11/p1-menon.pdf. doi:10.14778/3151113.3151114.

[4] T. Kersten, V. Leis, A. Kemper, T. Neumann, A. Pavlo, P. Boncz, Everything you always wanted to know about compiled and vectorized queries but were afraid to ask, Proc. VLDB Endow. 11 (2018) 2209–2222. URL: https://doi.org/10.14778/3275366.3284966. doi:10.14778/3275366.3284966.

[5] P. Chrysogelos, P. Sioulas, A. Ailamaki, Hardware-conscious query processing in gpu-accelerated analytical engines, in: 9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings, www.cidrdb.org, 2019. URL: http://cidrdb.org/cidr2019/papers/p127-chrysogelos-cidr19.pdf.

[6] Y. Yuan, R. Lee, X. Zhang, The yin and yang of processing data warehousing queries on gpu devices, Proc. VLDB Endow. 6 (2013) 817–828. URL: https://doi.org/10.14778/2536206.2536210. doi:10.14778/2536206.2536210.

[7] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, P. V. Sander, Relational query coprocessing on graphics processors, ACM Trans. Database Syst. 34 (2009). URL: https://doi.org/10.1145/1620585.1620588. doi:10.1145/1620585.1620588.

[8] P. Sioulas, P. Chrysogelos, M. Karpathiotakis, R. Appuswamy, A. Ailamaki, Hardware-conscious hash-joins on gpus, in: 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019, IEEE, 2019, pp. 698–709. URL: https://doi.org/10.1109/ICDE.2019.00068. doi:10.1109/ICDE.2019.00068.

[9] P. Chrysogelos, Efficient analytical query processing on cpu-gpu hardware platforms (2022) 132. URL: http://infoscience.epfl.ch/record/296204. doi:https://doi.org/10.5075/epfl-thesis-8068.

[10] P. Chrysogelos, M. Karpathiotakis, R. Appuswamy, A. Ailamaki, Hetexchange: Encapsulating heterogeneous CPU-GPU parallelism in JIT compiled engines, Proc. VLDB Endow. 12 (2019) 544–556. URL: http://www.vldb.org/pvldb/vol12/p544-chrysogelos.pdf. doi:10.14778/3303753.3303760.

[11] A. Raza, P. Chrysogelos, P. Sioulas, V. Indjic, A. G. Anadiotis, A. Ailamaki, Gpu-accelerated

data management under the test of time, in: 10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings, www.cidrdb.org, 2020. URL: http://cidrdb.org/cidr2020/papers/p18-raza-cidr20.pdf.

[12] C. Lutz, S. Breß, S. Zeuch, T. Rabl, V. Markl, Pump up the volume: Processing large data on gpus with fast interconnects, in: D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, H. Q. Ngo (Eds.), Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020, ACM, 2020, pp. 1633–1649. URL: https://doi.org/10.1145/3318464.3389705. doi:10.1145/3318464.3389705.

[13] T. Kraska, A. Beutel, E. H. Chi, J. Dean, N. Polyzotis, The case for learned index structures, in: G. Das, C. M. Jermaine, P. A. Bernstein (Eds.), Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018, ACM, 2018, pp. 489–504. URL: https://doi.org/10.1145/3183713.3196909. doi:10.1145/3183713.3196909.

[14] P. Sioulas, A. Ailamaki, Scalable multi-query execution using reinforcement learning, in: G. Li, Z. Li, S. Idreos, D. Srivastava (Eds.), SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, ACM, 2021, pp. 1651–1663. URL: https://doi.org/10.1145/3448016.3452799. doi:10.1145/3448016.3452799.

[15] D. He, S. C. Nakandala, D. Banda, R. Sen, K. Saur, K. Park, C. Curino, J. Camacho-Rodríguez, K. Karanasos, M. Interlandi, Query processing on tensor computation runtimes, Proc. VLDB Endow. 15 (2022) 2811–2825. URL: https://www.vldb.org/pvldb/vol15/p2811-he.pdf.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems 30 (2017).

[17] R. Bordawekar, O. Shmueli, Using word embedding to enable semantic queries in relational databases, in: Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning, DEEM'17, Association for Computing Machinery, New York, NY, USA, 2017. URL: https://doi.org/10.1145/3076246.3076251. doi:10.1145/3076246.3076251.

[18] R. Bordawekar, O. Shmueli, Enabling cognitive intelligence queries in relational databases using low-dimensional word embeddings, arXiv preprint arXiv:1603.07185 (2016).

[19] V. Sanca, A. Ailamaki, Analytical engines with context-rich processing: Towards efficient next-generation analytics, in: 2023 IEEE 39th International Conference on Data Engineering (ICDE), 2023, pp. 3699–3707. doi:10.1109/ICDE55515.2023.00298.

[20] K. Park, K. Saur, D. Banda, R. Sen, M. Interlandi, K. Karanasos, End-to-end optimization of machine learning prediction queries, in: Z. G. Ives, A. Bonifati, A. E. Abbadi (Eds.), SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022, ACM, 2022, pp. 587–601. URL: https://doi.org/10.1145/3514221.3526141. doi:10.1145/3514221.3526141.

[21] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, Y.-C. Liu, Knights landing: Second-generation intel xeon phi product, IEEE Micro 36 (2016) 34–46. doi:10.1109/MM.2016.25.

[22] S. Naffziger, N. Beck, T. Burd, K. Lepak, G. H. Loh, M. Subramony, S. White, Pioneering chiplet technology and design for the amd epyc™ and ryzen™ processor families: Industrial product, in: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), IEEE, 2021, pp. 57–70.

[23] S. Naffziger, K. Lepak, M. Paraschou, M. Subramony, 2.2 amd chiplet architecture for high-performance server and desktop products, in: 2020 IEEE International Solid-State Circuits Conference-(ISSCC), IEEE, 2020, pp. 44–45.

[24] C. Kenyon, C. Capano, Apple silicon performance in scientific computing, in: 2022 IEEE High Performance Extreme Computing Conference (HPEC), IEEE, 2022, pp. 1–10.

[25] N. Nassif, A. O. Munch, C. L. Molnar, G. Pasdast, S. V. Lyer, Z. Yang, O. Mendoza, M. Huddart, S. Venkataraman, S. Kandula, R. Marom, A. M. Kern, B. Bowhill, D. R. Mulvihill, S. Nimmagadda, V. Kalidindi, J. Krause, M. M. Haq, R. Sharma, K. Duda, Sapphire rapids: The next-generation intel xeon scalable processor, in: 2022 IEEE International Solid- State Circuits Conference (ISSCC), volume 65, 2022, pp. 44–46. doi:10.1109/ISSCC42614.2022.9731107.

[26] S. Williams, L. Ionkov, M. Lang, Numa distance for heterogeneous memory, in: Proceedings of the Workshop on Memory Centric Programming for HPC, MCHPC'17, Association for Computing Machinery, New York, NY, USA, 2017, p. 30–34. URL: https://doi.org/10.1145/3145617.3145620. doi:10.1145/3145617.3145620.

[27] I. Psaroudakis, T. Scheuer, N. May, A. Sellami, A. Ailamaki, Adaptive numa-aware data placement and task scheduling for analytical workloads in main-memory column-stores, Proc. VLDB Endow. 10 (2016) 37–48. URL: https://doi.org/

10.14778/3015274.3015275. doi:10.14778/3015274.3015275.

[28] Intel, 2023. URL: https://www.intel.com/content/www/us/en/products/sku/232592/intel-xeon-cpu-max-9480-processor-112-5m-cache-1-90-ghz/specifications.html.

[29] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, K. Kim, Hbm (high bandwidth memory) dram technology and architecture, in: 2017 IEEE International Memory Workshop (IMW), 2017, pp. 1–4. doi:10.1109/IMW.2017.7939084.

[30] W. A. Wulf, S. A. McKee, Hitting the memory wall: Implications of the obvious, ACM SIGARCH computer architecture news 23 (1995) 20–24.

[31] W. Cui, Q. Zhang, S. Blanas, J. Camacho-Rodríguez, B. Haynes, Y. Li, R. Ramamurthy, P. Cheng, R. Sen, M. Interlandi, Query processing on gaming consoles, in: Proceedings of the 19th International Workshop on Data Management on New Hardware, DaMoN '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 86–88. URL: https://doi.org/10.1145/3592980.3595313. doi:10.1145/3592980.3595313.

[32] S. Ramos, T. Hoefler, Capability models for manycore memory systems: A case-study with xeon phi knl, in: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2017, pp. 297–306. doi:10.1109/IPDPS.2017.30.

[33] S. Jha, B. He, M. Lu, X. Cheng, H. P. Huynh, Improving main memory hash joins on intel xeon phi processors: An experimental approach, Proc. VLDB Endow. 8 (2015) 642–653. URL: https://doi.org/10.14778/2735703.2735704. doi:10.14778/2735703.2735704.

[34] V. Sanca, A. Ailamaki, Sampling-based AQP in modern analytical engines, in: S. Blanas, N. May (Eds.), International Conference on Management of Data, DaMoN 2022, Philadelphia, PA, USA, 13 June 2022, ACM, 2022, pp. 4:1–4:8. URL: https://doi.org/10.1145/3533737.3535095. doi:10.1145/3533737.3535095.

[35] H. Nicholson, A. Raza, P. Chrysogelos, A. Ailamaki, Hetcache: Synergising nvme storage and GPU acceleration for memory-efficient analytics, in: 13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023, www.cidrdb.org, 2023. URL: https://www.cidrdb.org/cidr2023/papers/p84-nicholson.pdf.

[36] K. Viswanathan, Intel® memory latency checker v3.9a, 2023. URL: https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html.

[37] R. Mahajan, R. Sankman, N. Patel, D.-W. Kim, K. Aygun, Z. Qian, Y. Mekonnen, I. Salama, S. Sharan, D. Iyengar, et al., Embedded multi-die interconnect bridge (emib)–a high density, high bandwidth packaging interconnect, in: 2016 IEEE 66th Electronic Components and Technology Conference (ECTC), IEEE, 2016, pp. 557–565.

[38] M. S. Kester, M. Athanassoulis, S. Idreos, Access path selection in main-memory optimized data systems: Should i scan or should i probe?, in: Proceedings of the 2017 ACM International Conference on Management of Data, 2017, pp. 715–730.