Maia 200: A Software Defined Dataflow System for Large-scale AI Acceleration

Sherry Xu, Marco Heddes, Jackson Peng, Tom Savell, Monica Tang, Prashant Ranjan,
Jesse Benson, Ofer Dekel, Saurabh Dighe, Anupama Kurpad, Artour Levin, Matthew Mattina,
George Petre, Cheng Tang, Yuan Yu, Li Zhang, Torsten Hoefler
Microsoft Corporation

***Abstract*—We introduce Maia 200, an advanced AI accelerator delivering high performance—10 145 Tflop/s FP4 and 5072 Tflop/s FP8 within a 750W TDP and 7 TB/s HBM bandwidth. Maia exemplifies a new class of Software Defined Locally Accessed Dataflow Architectures (SDLA), which explicitly program dataflow engines to orchestrate highly specialized memories and data movement engines. This approach shifts the focus from today's thread-centric to data-movement-centric architecture, improving efficiency and scalability. Our taxonomy of data management, inspired by Flynn's classification, highlights how SDLA addresses challenges in modern AI computing. Maia 200 achieves significant cost and energy savings while supporting massive parallelism for AI inference workloads, making it a compelling solution for next-generation high-performance computing systems.**

I. Introduction

The advances in modern generative AI techniques have taken the high-performance computing community by storm. AI workloads, especially inference of Large Language Models (LLMs), are now consuming most of the compute cycles world-wide and building efficient high-performance systems to support those workloads is crucial. We are in the middle of one of the biggest infrastructure buildouts in human history – while training systems are deployed in large datacenter supercomputers with hundreds of thousands of accelerators [19], [53], inference systems even outnumber their computational capacity. Based on public statements, we conservatively estimate that each day, at least 1.2 trillion tokens are generated by such systems world-wide. If we assume a relatively small 8 billion parameter (8B) LLM, this would equate to a sustained compute of 6.85 exaflop/s at any given moment; for a larger 400B model, this would be 0.3 zettaflop/s. In addition, the largest supercomputers deployed perform full-scale AI training. Soon, the required total computation will be in the zettaflop/s range and developing TCO- and CO_2e-efficient systems is most important.

Here, we describe Maia 200, a top-of-the-line AI accelerator system, specialized to AI computing workloads delivering best-in-class performance with 10 145 Tflop/s FP4 / 5072 Tflop/s FP8 performance per chip within a 750W TDP (13.3/6.7 Tflop/W) and 7 TiB/s HBM bandwidth. A distributed Maia 200 system integrating 6144 chips offers up to 62 exaflop/s FP4 throughput, 43 PiB/s memory, and 8.6 PiB/s Ethernet network bandwidth. Internal data suggests that Maia 200 saves 30% cost (TCO) and 15% energy compared to any other AI accelerator in Microsoft's fleet due to an aggressive co-design strategy of AI workloads and architecture concepts as well as specific microarchitecture implementation, software stack, and workload implementation without over-specializing to specific problems.

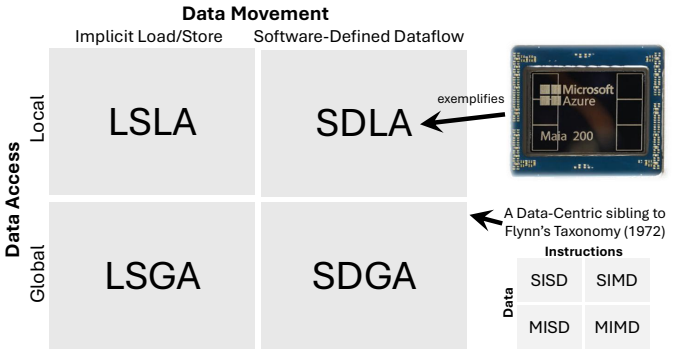

Fig. 1. A Modern Data Management Classification of Processors Inspired by Flynn's Original Instruction-focused Taxonomy from 1972.

Maia implements a new type of accelerator, which defines a family of emerging architectures that we call Software Defined Locally Accessed Dataflow Architectures (SDLA). Specifically, Software-defined Dataflow uses an abstract machine model that makes data path programming explicit. It is a similar architectural step to what Single Instruction Multiple Threads (SIMT) was to define the warp-based thread scheduling mechanism establishing classical GPU architectures. It is very different in philosophy as SDLA uses parallel and independent control and data instruction streams to efficiently orchestrate highly specialized memories and data movement engines, **taking the spotlight away from threads to focus on a data-movement centric view with localized data access.**

We first describe the fundamentals of Software-defined Dataflow and its variant SDLA and then dive into details on the Maia 200 workload requirements, chip and system architectures.

II. Accelerating Data Management with Software-Defined Dataflow

We describe a new family of computer architectures that is aimed specifically at accelerating AI workloads at scale. The main optimization goals are to optimize TCO and lower CO_2e emissions by improving silicon area and overall system and energy efficiency. We recognize that data management, which comprises storing, moving, and converting data, is the most challenging problem in modern computer architecture [33], [60], [76]. Furthermore, AI workloads typically process large amounts of data with massive parallelism in relatively regular coarse-grained blocks with predictable control- and dataflow, which we will later define more formally using the concept of data obliviousness.

To describe the principles in more detail, we define a data-management taxonomy that is similar to Flynn's taxonomy of instruction parallelism [20]. Our taxonomy is shown in Fig. 1 and distinguishes two dimensions of how compute elements access data and how data movement is defined.

The vertical data access dimension is similar to Flynn's data dimension in that it defines whether the compute units can access only local data (logically distributed memories) or all data is accessed globally (logically a single memory). We call those *locally and globally accessed*. Our horizontal dimension, the data movement dimension is similar to Flynn's instruction dimension in that it defines how data movement is orchestrated – either as part of the computation instruction stream (Load/Store) or in a separate control instruction stream (which we call *Software Defined dataflow*).

All four quadrants of our taxonomy have example systems today: LSGA is used in standard multicore CPUs forming the most established shared memory abstraction [21]. GPUs or Cerebras' WSEs [42] use LSLA with distributed scratch pad memories. Some CPUs extend the standard load/store model to SDGA by adding programmable DMA units such as Intel's I/O Acceleration Technology [72] or Data Streaming Accelerator [9], AMD's DirectPath I/O [1], or ARM's AMBA DMA [3]; all those use global data access in shared memory.

In this work, we emphasize Software-Defined Locally Addressed (SDLA) Dataflow architectures. Specifically, SDLA separates dataflow execution from control to allow the programmer to orchestrate details of data management in a fine-grained, parallel, and asynchronous manner to enable ideal overlapping of computation and data movement. Furthermore, SDLA's distributed memories provide significant opportunities for specialization and acceleration such as right-sized memories collocated with execution units and optimized data movement and type conversion acceleration units. Some existing architectures such as AMD's Xilinx-based Distributed Network Architecture (XDNA) [58] or Samba Nova's Reconfigurable Dataflow Architecture [54] would fall roughly into the SDLA category and industry adoption is growing. In this work, we focus on our Maia 200 architecture, the second-generation SDLA from Microsoft.

SDLA defines a new abstract machine model as a contract between programmers and computer architects. It can be seen as one step beyond Single Instruction Multiple Threads (SIMT) architectures that form the basis of classical GPU acceleration [43]. SDLA differs in that it does not focus on sharing instruction streams between threads but on the efficiency of data management. While SDLA enables instruction stream sharing between processing elements, it does not enforce full warp-style synchronization at the program level, enabling independent programming of all units. SDLA fosters efficient programming of spatial architectures where data locality and management are first-class citizens [23], [48], [73].

Fundamentally, SDLA is based on three foundational principles from a programming perspective:

1) **SDLA exposes parallel low-level control programming of accelerated data movement, conversion, and synchronization engines to the programmer, compiler, and software stack.** This enables highest performance and highest power efficiency through explicit control of data management.
2) **SDLA allows programmers to manage an efficient system of distributed and specialized (on-chip and in-pod) memories.** This enables architects to optimize memories and accelerate data movement with special engines and programmers to minimize overheads.
3) **SDLA enables architects to build systems offering mostly deterministic performance to enable optimized scheduling.** This is powerful for implementing kernels such as transformer attention, where much effort is spent to adopt code to idiosyncrasies of current accelerators [62].

These principles enable compiler engineers to build emulation layers and scheduling libraries that behave similarly to current firmware in existing accelerators and provide a simplified programming interface. Yet it empowers expert programmers to either use high-level programming for non-critical pieces or dive into every detail of memory management and scheduling for performance-critical code pieces. Current classical accelerators like GPUs only offer one level of programming such as CUDA while scheduling (e.g., warps) and fine-grained DMA control are implemented in hardware or firmware and thus out of the reach of typical programmers.

These principles also enable computer architects to co-design execution engines for their tasks such as data movement, synchronization, or computation and thus optimize silicon design. Current expert GPU programmers use "warp specialization" as a technique to pipeline memory accesses with computations. Yet, this software technique uses standard warp threads for both purposes. SDLA is one step ahead in that execution engines are specialized to their purpose and thus more efficient when co-designed in hardware.

In summary, **SDLA redefines the contract between architects and programmers to provide lowest level control but at the same time enable an advanced software stack and compiler to hide the complexity.** Similar to how classical GPUs hide details of the instruction sets (ISAs) through

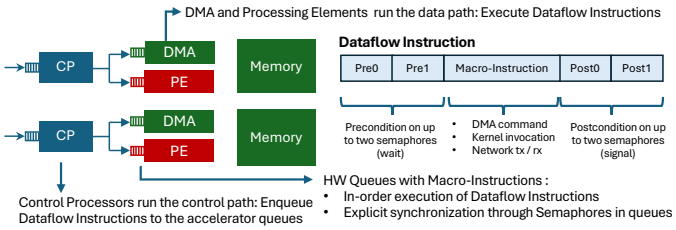

Fig. 2. Abstract Machine Model for our Dataflow ISA.

LLVM-like PTX abstractions [50], SDLA enables two layers of programming where the programmer can choose to write assembly code (cf. Streaming Assembly, SASS [38]) or higher-level CUDA-like code. The **control path** (processors) is programmed in a standard low-level systems language such as C or C++ while the **programmable data path uses a dataflow instruction set architecture** to execute complex dataflow dependencies. The software stack can then build higher-level functionality such as Python/PyTorch or Triton [70] interfaces on top of the control and programmable data paths.

At a more abstract level, SDLA extends the classical GPU's SIMD to SIMT transition with a focus on data management. It provides access to the programmable data path architecture details for explicit accelerated data movement between specialized memories on-chip and across chips. Software-defined means that the application logic is supported by **control programs that orchestrate the data path**. This makes SDLA a statement about control (synchronization and data movement) and not about application logic and it thus supports both SPMD (e.g., OpenMP [14], MPI [47]) and MPMD (e.g., separating different coupled components such as prefill and generation phases in LLM inference) programming styles.

We show an example of our generic SDLA dataflow architecture in Fig. 2. It uses control programs written in a programmer-accessible high-level systems language such as C or C++. Those programs run in the control path and orchestrate the dataflow computation in the data path through queued instructions. The data path itself comprises different units for data management and processing. Those units support a dataflow instruction set architecture (Dataflow ISA) that defines synchronization pre-conditions, a macro-instruction, and synchronization post-conditions. The management of pre- and post-conditions can be fully implemented in hardware and macro-instructions can either be complex hardware-instructions such as a tensor-core matrix multiplication or software functions invoked on programmable processors such as vector units.

III. The Maia 200 System

Microsoft's AI Architecture (Maia) 200 is Microsoft's second- generation of AI accelerators implementing an SDLA dataflow architecture. It is highly optimized for Microsoft's inference workloads to enable extreme-scale transformer-based LLMs at competitive cost and energy consumption throughout the worldwide fleet. It's designed to run a very defined workload through extreme co-design of hardware, software, deployment, and operational aspects of the system. To achieve this, it embraces a new programming philosophy that delivers lower TCO and energy consumption. Yet it is not overspecialized and supports future workload shifts.

Maia 200's architecture takes advantage of the SDLA programming view to improve silicon and energy efficiency and thus cost. Specifically:

- Maia 200's architecture uses specialized small memories attached to specific functional units to improve silicon and energy efficiency.
- Maia 200 organizes those memories and compute units hierarchically to take advantage of locality in workloads and programming.
- Maia 200 extends seamlessly into the network. The network interfaces are driven as part of the programmable data path, e.g., data moves seamlessly from local SRAM or HBM into remote SRAM or HBM.

Maia 200 is fabricated in TSMC's 3nm process with more than 140 billion transistors on a near reticle-sized monolithic die (26x33mm). It uses CoWoS-S packaging technology to co-locate HBM on a silicon interposer in a 75mm x 75mm package with a total of 750 W SoC TDP distributed via 19 metal layers. It is a full system including tray, rack, and network architecture scalable to thousands of accelerators in a single cluster and it is in production in the fleet today. It is mainly optimized for delivering highest efficiency for massive inference workloads using trillion-parameter frontier models as this is the main demand today.

A. *Inference workload challenges at scale*

Maia 200 is specifically designed for inference workloads of large LLMs including Mixture of Expert layers [63]. Much of the workload has soft real-time requirements in that a user is waiting for the output relying on relatively strict SLAs for example 300 – 4000 ms to first token and 20-30 ms per token. Inference generally can be split in two components: prefill and decode. Prefill computes the first output token from the concatenated system and user prompts, which can be tens of thousands of tokens while decode generates token-by-token based on the contextual history in the KV cache [75]. Thus, prefill usually uses the system architecture in a balanced manner while decode is often memory-bandwidth limited due to the KV cache accesses.

Batching can change the balance by re-using weights across elements of a batch but decode still requires loading a separate KV cache for each batch entry. Maximal batch sizes are limited by the SLA as well as the accelerator's memory size and parallelization [8]. Inference systems can utilize lower-precision datatypes such as FP4 or FP8 if used with fine-grained scaling factors [49], [52]. Prefill and decode phases can be disaggregated to bandwidth-optimized and compute-optimized accelerator configurations. Typically, batching results in a relatively large number of compute and communication operations, often involving several kilobytes or megabytes of data.

In general, inference of large LLMs requires a cluster of petaflop/s accelerators to perform the task for multiple requests simultaneously [41], [79]. The Maia 200 system architecture designs such a supercomputer cluster system with 62 exaflop/s FP4 performance that enables specialization to the different phases and 8.6 PiB/s network connectivity to enable efficient inference.

Alongside those common challenges in large-scale LLM inference, we explicitly model new trends that will shape future workloads:

1) Massively growing context windows are required by reasoning models [10] that implement chain, tree, or graph of thoughts, multimodal models including images, voice, and even videos, and retrieval augmented generation (RAG). Those require hundreds of thousands to millions of tokens and extreme bandwidths in wide data paths (Section III-B3).
2) Growing models require more compute power and memory bandwidth per token; diffusion models need even more compute power for many iterations (Section III-B4).
3) Coarse-grained sparsity and dynamic computations generate more data movement that can only be scheduled at runtime. The prime examples are Mixture of Experts (MoE) [63] systems that often require all-to-all communication at extremely high bandwidths. Thus, the control path must be fast to implement data-dependent control (Section III-B5).
4) Agentic systems and tool use agents cause more complex workflows [67], potentially involving multiple accelerators with different models that need to be tightly coupled (Section III-C).

In general, the extremely fast-growing demand makes reducing energy consumption and cost even more important in the near future.

B. The Maia 200 System on Chip

While Maia's foundational innovation lies in adopting the SDLA principles outlined above, the Maia 200 SoC requires many staple features for modern AI accelerators, such as:

Specialized (scaled block) narrow datatypes to enable hardware-efficient inference and training without significant accuracy loss: Maia 200 emphasizes block-scaled FP4 and FP8 tensor compute.

A capable Network on Chip (NoC) to connect the reticle-sized massive monolithic chip: Maia 200's NoC is physically split into a logical high-bandwidth data NoC and a specialized control NoC to keep data- and control traffic separate. The data NoC also supports QoS to prioritize different data traffic (e.g., coming from NIC or HBM).

High HBM bandwidth, connected to the NoC to deliver highest bandwidth to all units: Maia 200 uses six HBM3e stacks.

Tensor cores with complex operations: Maia 200's flexible Tile Tensor Units support matrix multiplications as well as convolutions natively, a feature only announced for future GPUs.

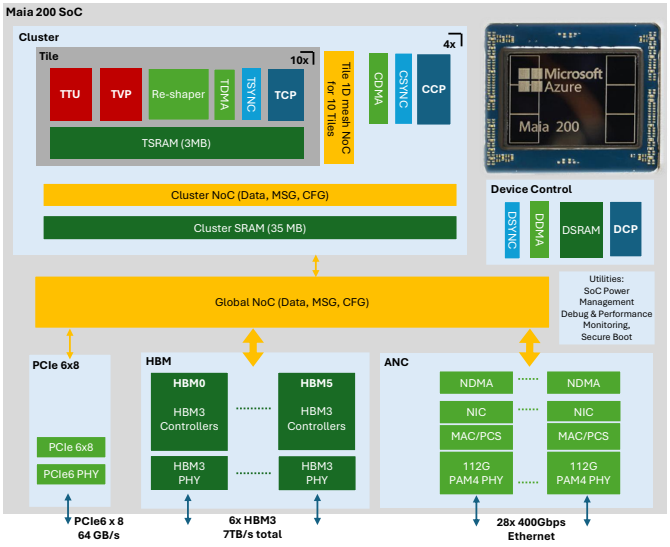

Fig. 3. Maia 200 Abstract System on Chip Architecture. Processing blocks are shown in red, the data path is green with data movement accelerators in light green and memories in darker green, the NoC is in yellow, and the control path is in blue with synchronization engines in light blue and control processors in darker blue.

Advanced power management features (DVFS): Maia 200 offers power steering through separate clock domains for compute, the NoC, and the HBM subsystem. This allows us to specialize chips at deployment in different configurations, e.g., optimized for prefill or generation.

In addition to those staple features that are a must-have for any modern AI accelerator, Maia 200 innovates on multiple fronts using SDLA principles:

- Control and data management are implemented separately, allowing us to optimize highly-specialized units such as DMA, synchronization (hardware semaphore), and processing elements.
- The NoC offers multicast capability to efficiently distribute data to all clusters at full bandwidth.
- PEs are equipped with highly specialized memories, for example, the Tile Tensor Unit uses highly specialized memories to access inputs and outputs.
- Additionally, an on-chip hierarchy of specialized memories supports efficient near- (within an on-chip cluster) and far (remote on-chip cluster) communications.
- The system offers several specialized programmable data-movement and data management accelerators supporting: Strided DMA operations for 1D, 2D, 3D, 4D tensors including scatter/gather support and line-rate datatype casting and sparsity support in the Tile DMA engines.
- Extreme network bandwidth while keeping the network below 20% of the system cost, key are the 28 integrated co-designed NICs with a simplified transport protocol and an innovative network topology using mostly fixed links saving switches and cables.

Fig. 3 shows the overall logical architecture of a Maia 200 SoC. The central compute complex is formed by four

Clusters on the chip die. Each of the four clusters contains nine or ten Tiles. Each Tile contains one Tile Tensor Unit (TTU) and one Tile Vector Processor (TVP), a specialized tile memory, DMA engines, Synchronization (Sync) engines, and a Tile Control Processor (TCP). The tiles are connected by a special Tile 1D mesh NoC to enable faster communication among each other. Each of the four Clusters has a separate Cluster SRAM (CSRAM), a Cluster Control Processor (CCP), Synchronization blocks, and DMA engines. The global NoC (GNoC) connects all clusters, the HBM, the host system via PCIe, and the Integrated NIC block (AI Network Controller, ANC). The chip is coordinated by a Device Control block including a Device Control Processor (DCP) as well as SRAM and specialized DMA and Sync engines. The SoC also contains utility functions for power management, performance monitoring, debugging, and secure boot. All-in-all, Maia 200 is a capable microarchitecture implementing the SDLA ideas. We will elaborate on some of the design decisions below.

1) Explicit Scratchpad versus Caches for SoC SRAM: There is a subtle tradeoff between caches and explicitly programmed scratchpad memories. Hiding fast cache memories between the main memory and registers is simplest for entry-level programmers. However, for extreme performance-conscious ninja programming, the discussion becomes more complex. Ninja programmers usually consider the details of the cache to optimize their data access patterns. This can be quite complex and non-portable depending on the complexity and documentation of the cache itself. For example, cache replacement strategies, associativity, and cache-line sizes must be considered when writing the highest-performing code [4], [18], [30], [59], [65], [66]. Yet not all code pieces are performance critical and getting reasonable performance at lowest initial investment and simpler programming remains an important goal.

First, we look at AI workloads: An important property of algorithms is whether they are data oblivious. An algorithm is data oblivious (or oblivious for short) if all memory accesses and dataflow can be computed based on a small number of parameters that are available at compilation time. Most AI workloads are a static composition of fixed kernels (MMM, SoftMax, non-linearities, etc.) that are all oblivious, making the overall dataflow oblivious and thus plannable [25]. There are some small exceptions such as continual/dynamic batching, Mixture of Experts, other forms of sparsity, and early termination [26]. All those can either be handled in a parametric way that takes advantage of their near obliviousness, or they are at the very coarse control level and thus not critical for performance. In summary, the **mostly oblivious and thus plannable AI workloads enable ninja programmers and compilers to statically plan all data movement and placement in an overall optimized (if not optimal) schedule using distributed memories and accelerated data paths efficiently** [5], [40], [48], [80].

Caches are considered vital for many applications and enable extremely quick time to first implementation. In fact, one can show that, for an LRU replacement cache with unit-sized cache-lines that is twice as large as a scratchpad memory using the optimal (offline) memory management scheme, the cache will at most have twice as many misses [37], [57]. Thus, LRU caches are only a small constant factor of about four away from optimal. This minor overhead is often tolerable, so CPUs and most general-purpose GPUs use caches to make programming easier. Thus, we carefully study the benefits programmers as well as computer architects could gain from replacing caches.

Alongside the previously mentioned energy and cost savings—up to four times—ninja programmers are also able to orchestrate highly efficient data transfers for AI tasks. Similarly, for a set of HPC workloads, including many non-oblivious codes, Marinelli et al. [46] showed a mean geometric speedup of 13% when manually porting those from caches to scratchpad memories. Specifically, ninja programmers benefit from scratchpad memories in that (1) Data can be placed and packed at word-granularity in all memories. Cache line size and cache line sharing play no role, which simplifies the handling of non-contiguous fine-grained data significantly. (2) Programmers do not need to worry about replacement schemes and keeping the right data in cache or issuing special uncached load/store instructions for streaming accesses (such as weights) that may block the pipeline. They simply keep the data that they want in scratchpad and overwrite it explicitly. (3) Programmers do not need to account for hidden prefetch streams. In SDLA, programmers orchestrate all details of asynchronous pipelining through accelerated data movement engines. And finally, (4) specific extensions such as 1D/2D/3D tensor loads and/or datatype conversions can save a significant number of CPU instructions [39], [61].

Hardware implementations of caches are also more expensive than software: in addition to the SRAM cells, they require tag memory arrays and address remapping logic. The area and energy overhead for this logic is around 30-35% [6], [46]. Furthermore, the access latency is increased by 10-15% due to address decoding and mapping [6], [46]. Overall, this causes an area-time and dynamic energy overhead of up to 43% in practice [46].

Our explicitly programmed SDLA memory architecture enables Maia 200 to spend less than 20% of its chip space on memory, which is far less than many CPUs with smaller cache sizes. Furthermore, all SRAMs and the HBM are ECC protected and can be scrubbed on demand to protect the large die from bit flip corruption. The specialized DMA engines to orchestrate the data movement occupy only negligible die area.

To unify the best of hardware design and plannable software, Maia 200 aims to use compilers to utilize software-managed scratchpad memories at low programmer complexity and reasonable performance [2], [68], while ninja programmers will have the choice to orchestrate the data movement at the lowest levels. Compilers offer basic support for Python/PyTorch and Triton programming at the top level while ninja programmers can dive into writing C/C++ control programs for some kernels to use the architecture at highest

efficiency. In this work, we focus on both the hardware and system architecture and thus will not describe the software stack.

2) Dataflow Programming and Instructions: Maia 200 combines several types of computational and data management (memory and data movement) units to implement an SDLA microarchitecture. It combines two main types of compute units paired in each Tile: A Tile Tensor Unit (TTU) and a Tile Vector Processor (TVP). TVPs can execute complex programs written in C/C++ while the TTU supports a large set of fixed dataflow instructions (matrix multiply and convolutions). DMA engines support a fixed instruction set for moving data between memories. Each Tile DMA engine includes a Reshaper engine that can perform data conversions between different formats and data layout changes such as transposition at line-rate.

Each of those engines supports a specific set of powerful **macro-instructions**, which can be full programs (e.g., in TVPs) or fixed functions (e.g., in TTUs or DMA engines). Those macro-instructions are part of a **dataflow instruction set architecture (DISA), which defines a set of dataflow preconditions, a macro-instruction to invoke, and a set of dataflow postconditions** as shown in Fig. 2. Each basic (dataflow and compute) execution engine supports DISA instructions that are used to orchestrate the overall dataflow program. Dataflow conditions are managed using sync engines that offer logical semaphore objects that are assigned by the control program. Each semaphore can be configured to fire at a specific threshold. Each dataflow instruction can be configured to wait for up to two semaphores before executing (precondition). After the execution finishes, it can signal up to two semaphores (postconditions), which in turn could trigger another dataflow macro-instruction. All wait and signal conditions have a programmable decrement or increment value and commands can be chained to enable more complex synchronization conditions.

Maia 200 includes a hierarchy of different types of control processors that orchestrate the dataflow programs using the execution units and sync engines to manage the overall data flow. The control program orchestrates the setup of the dataflow instructions and semaphores ahead of time and the overall dataflow program executes asynchronously with respect to the control program (which usually runs ahead). This scheme can implement optimal dataflow for oblivious programs without any delays due to the parallelism of control and dataflow in hardware. We offer a simple C++-based low level programming language called NEsted PArallel Language (NEPAL) to program the control path.

3) Connecting Memories and Compute Engines: Maia 200 uses memories that are directly attached to compute engines for highest efficiency. They are sized according to Little's Law [44] to satisfy the dataflow throughput of the engines for pipelined and overlapped loading and computing as well as to maximize reuse of data and minimize data movement and addressing overheads.

Each tile has 3 MiB of specialized SRAM to hold the operands for the TTU. Each TTU can perform 65 536 Multiply Accumulate (MAC) FP4 operations per cycle. It reads two input matrices of size 32x64 and outputs, or accumulates into a 32x32 FP32 or BF16/FP16 matrix. For FP8 and BF16, it performs half or 1/8th of the MAC operations and supports 32x32 or 32x8 input matrices, respectively. This means that the input dataflow bandwidth at 2 GHz is 2024 GiB/s per operand and the output bandwidth is up to 4096 GiB/s (with an additional 4096 GiB/s read bandwidth for accumulation). Thus, the memories can hold work for the TTU for more than 700 cycles, which enables full-utilization pipelining. Each TVP works on TSMEM such that it can perform post-processing of the matrix outputs and TTUs can also read one input operand from TSMEM to enable efficient chains of operations. The memories are orchestrated through Tile DMA engine dataflow commands, which support optional reshaping. The Tile DMA can use the Tile NoC to communicate with neighboring tiles or the Cluster NoC to access the Cluster SRAM.

The overall system comprises many dozens of different DMA engines, some move data between on-chip SRAMs and/or HBM, some move data into the network to remote SRAMs or HBM. NoC QoS priorities can be assigned on a per-command basis. The utility fabric fully connects all Sync blocks, which can each notify any other Sync block of the SoC at lowest latency.

4) Datatypes and Conversion Acceleration: Maia 200 supports a flurry of different datatypes and formats. In line with its dataflow principles, it separates between *storage data types and compute data types*. Storage data types are used in the data management path to reduce the number of bits transported and stored while maintaining better accuracy than lower bit formats. Sparse tensor storage is the main storage datatype. The Reshaper unit can convert between wider and narrower data types using stochastic rounding or round to nearest. It can also load from 8:16 or 4:16 sparse tensors to/from dense tensors at full bandwidth.

Each TTU supports BF16, FP8, FP6, and FP4, with 8192, 32 768, 32 768, and 65 536 MACs, respectively. FP6 achieves the same performance as FP8 but at significant energy savings through gating. TTUs read matrices of size 32xKx32 in blocked FP4 and FP8 formats (e.g., K=32 for FP8 and K=64 for FP4) and output FP32 or BF16 at 262.14 Tflop/s for FP4 multiplications at 2 GHz. Each TVP supports a richer set of data formats for vector operations, which are shown in the adjacent table. TVPs implement 256 lanes for types up to 16-bit width at 3.07 Tflop/s and 128 lanes for FP32 at 1.54 Tflop/s at 2 GHz.

Datatype	Lanes
INT8/UINT8	512
INT16/UINT16	256
INT32/UINT32	128
BFP16	256
FP16	256
FP4 (E2M1, E1M2)	256
FP6 (E3M2, E2M3)	256
FP8 (E4M3, E5M2)	256
FP32	128

The FP8 format supports E4M3 and E5M2 layouts, the FP6 format supports E2M3 and E3M2 layouts and the FP4 format supports the E1M2, E2M1 layouts. Maia 200 allows

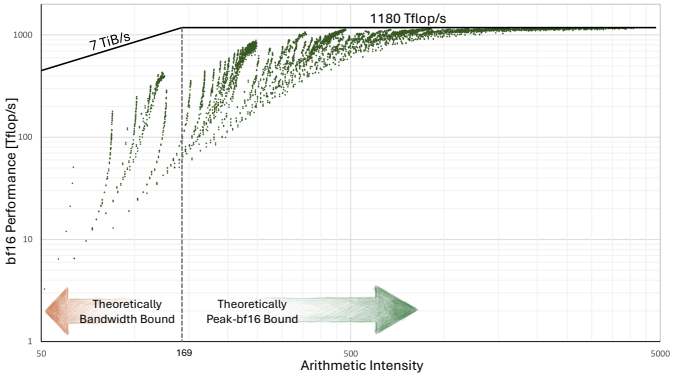

Fig. 4. BF16 Matrix Multiply Performance for 6143 Relevant Matrix Sizes.

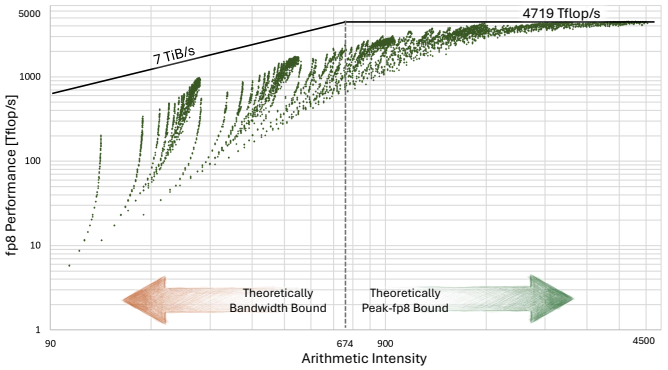

Fig. 5. FP8 Matrix Multiply Performance for 6143 Relevant Matrix Sizes.

programmers to compose those formats into OCP compliant MXFP data types with E8M0 scaling factors and group sizes of 32 [52].

5) Software-Defined Dataflow Control: The control path is the last missing piece to a full SDLA microarchitecture. Maia 200 combines three hierarchy levels of control: the SoC, Cluster, and Tile levels. Each of those levels has several control processors. Those processors are running C/C++ programs of the user to orchestrate the data path using the Dataflow ISA defined above. Those processors usually run slightly ahead of the dataflow to configure the engines before they are used. Slightly larger staging memories (CSRAM, TSRAM) support this asynchronicity and allow the control processors to stash enough work such that units are never waiting for work. Data-dependent data-to-control dependencies require a quick connection from the data path to the control path.

Maia's hierarchical architecture benefits such data-dependent control in that they can happen in the same tile and the data-to-control remains between adjacent units. For example, for a mixture of experts operation in which a routing function decides where to send activations to, the TVP evaluates the output of the TTU routing matrix multiplication in TSRAM and hands off the result to the TCP in a small number of cycles to kick-off preconfigured data movement operations to each destination. The control for this data-dependent flow (from TTU result to the TVP to the TCP) can be set up in advance through semaphores and DMA operations that get their scatter-gather operations from the TVP. This enables fastest local control for quickly changing data-dependencies. Similar methods can be used to support local implementations for sparse processing [26].

6) Power, Energy, and Management: Maia 200 is a complex system, and we skipped over most of the management functionality. For example, the details of how each control processor is configured and how kernels are launched are complex but less scientifically interesting. We now highlight some of those aspects that are most interesting to performance-conscious users and engineers.

Maia 200 offers a flexible Dynamic Voltage and Frequency Scaling (DVFS) system where each of the four clusters as well as the Global NoC are in different frequency domains. This allows us to deploy configurations optimized for prefill (high compute and high memory clocks) or token generation (lower compute and high memory clocks) to optimize for specific use-cases.

Maia 200 also offers a fully secure boot system and it is seamlessly integrated into Azure's compute philosophy and management.

7) Roofline Matrix Multiply Performance: We continue by investigating the most important workload in deep learning inference: low precision matrix multiplication. Here, we focus on the most relevant formats today: multiplying two bf16 matrices and multiplying two fp8 matrices. Both of those operations output bf16 matrices. We benchmark a Maia 200 system with 9 Tiles per cluster enabled running at 2 GHz (unthrottled). Thus, the peak bf16 performance is $262.14 \cdot 4 \cdot 9 = 1180$ Tflop/s and the peak fp8 performance is 4785 Tflop/s. The roofline's ridge point [74] lies at an arithmetic intensity of 674 for fp8 and 169 for bf16, respectively.

Fig. 4 shows a scatterplot of 6143 relevant matrix multiplications of different sizes that are relevant for AI inference workloads, spanning arithmetic intensities from 85 to 5000. We measured each matrix size performance by repeatedly loading the input matrices from HBM and writing back into HBM. The measurement was repeated until the time exceeded 1ms to amortize the timer accuracy. The variation across measurements is minimal due to the explicit SDLA programming and the absence of caches. We achieve remarkably high **bf16 roofline efficiency up to 99.69% of peak** in the compute bounded regime. The near-100% utilization is due to the asynchronous nature of data movement and SDLA programming. All instructions and data loading can overlap and all units are busy, synchronized by semaphores. We exceed 90% of the peak flop performance for multiplications using more than 58 Tflop of compute. In the memory bound regime, we achieve up to 51.4% of the peak bandwidth due to the smaller size of the matrices (larger ones quickly become compute bound). We achieved higher than 50% bandwidth for combined input and output operands bigger than 113.5 MiB. In both regimes, we outperform comparable GPUs and CPUs but consciously refrain from a direct comparison as this study aims to show the architectural efficiency of SDLA and Maia 200 using real

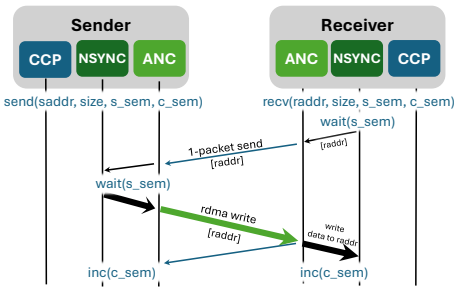

Fig. 6. ATL's Receiver-driven Messaging Scheme.

benchmarks.

Fig. 5 shows similar data and performance for fp8 matrix multiplications in a very similar setup with the same matrix sizes and experimental methodology. Here, we span arithmetic intensities from 95 to 4600 and achieve up to 96% of the peak performance in the compute-bound regime and up to 56% in the memory-bound regime.

C. Connecting Many SoCs into a Distributed Compute Cluster

Maia 200 uses 28 integrated 400 Gbps Ethernet-based AI Network Controllers (ANC) for a total bandwidth of 1.4 TB/s full duplex. The ANCs are heavily optimized for minimal chip space and energy consumption. They are fully integrated into the SDLA microarchitecture to which they offer programmable DMA engines, synchronization, and seamless data movement to remote SRAM and HBM memories. Over the lossless (PFC) Ethernet L2 network they run Microsoft's in-house AI Transport Layer version 2 (ATLv2) protocol [24] to implement network transport that later influenced the standardization of Ultra Ethernet [29], [71].

For packet transmission, the ATLv2 protocol uses standard L2 Ethernet frames with L3 IP routing headers and full end-to-end AES-GCM-256 encryption and a simple window-based congestion control scheme. It supports ECMP and entropy vectors encoded in the position of the UDP source port for per-packet load balancing and it uses a protocol similar to REPS [11], [36] for recycling entropies of known-to-be-good paths. It supports selective retransmit to overcome well-known limits of Go-Back-N retransmission [28]. Furthermore, it supports QoS not only in the network but also in the GNoC towards HBM to make sure messages are delivered without causing network back-up.

The SDLA functionality to access remote memories are exposed through send/receive where the sender specifies a send buffer and the receiver specifies a receive buffer (or multiple in the case of broadcasting) for the message to be copied into. We implement those semantics using Remote Direct Memory Access (RDMA) in the ANC. We use a receiver-driven messaging scheme that is optimized for AI communication in Collective Communication Libraries (*CCLs) to minimize message matching state. This scheme inspired the AI Base profile that was later standardized in Ultra Ethernet [29], [71] and is shown in Fig. 6.

In Fig. 6, the Sender CCP posts a send operation that will start after the start semaphore (s_sem) fires and it will increment the completion semaphore (c_sem). At the receiver, the CCP posts a receive with similar start and completion semaphores. The receiver waits for its s_sem to fire and then sends a control message containing the receive address to the sender. The sender's ANC will await a message from the receiver and once it receives it wait for the s_sem to fire. If both conditions are true, it will issue an RDMA write message to the receiver's address. Upon reception of the write command, the receiver increments its c_sem and once the sender receives the acknowledgement of the successful write operation, it in turn triggers its local c_sem. Note that this protocol does not require any buffering in the ANC as it can always retransmit the data via DMA from the source memories. The ANC itself splits each message into multiple packets that are sent in any order and both sender and receiver use bitmaps to track message completion. The system is optimized for messages larger than 4 kiB, which is common in AI workloads [24].

The system supports packet spraying across multiple ANCs along different network paths. ANCs also support remote semaphore increment together with data transmission as well as remote data broadcast to clusters on the GNoC. Messages can be written into remote CSRAM or HMB. This makes remote nodes seamless peers in the overall SDLA programming model.

The system design combines fixed connections on the same blade with switched connections. The topology is a special case of a 2x2 1D Hamming Mesh [27] where cross links are added on each board establishing full connectivity in the group collocated on a physical tray. Of the 28 ANCs per SoC, 20 are connected with fixed links as shown in Fig. 7, and eight are connected to a switched network. The switched network consists of four identical planes and each SoC connects with two 400G ANCs to each plane. While the switched ANC links can be connected into any topology with hundreds of thousands of endpoints, the current design point is aimed at a two-tier network with a maximum of 6144 SoCs for inference deployments. In this configuration, each of the 51.2T (128 400G ports) Tier-0 (T0) switches is connected to 48 Maia 200 SoCs in 12 trays with two links each. The remaining 32 ports are connected to up to 32 Tier-1 (T1) switches with a 1:3 oversubscription ratio. Thus, the overall number of SoCs is $48 \cdot 128 = 6144$. Smaller subset configurations are possible and deployed in the field. The chips and trays themselves can also be used to build larger-scale systems by adding more tiers of switches or changing the topology.

1) Collective Communication Performance: Collective communications are critical for deep learning performance. We now discuss how basic algorithms are implemented in Maia 200. We focus on initial implementations of two algorithms, direct-connect and ring [69], while we fully acknowledge that the performance can be improved with more elaborate algorithms (e.g., Bine Trees [17]). Yet, our purpose here is to demonstrate the efficacy and performance of the SLDA

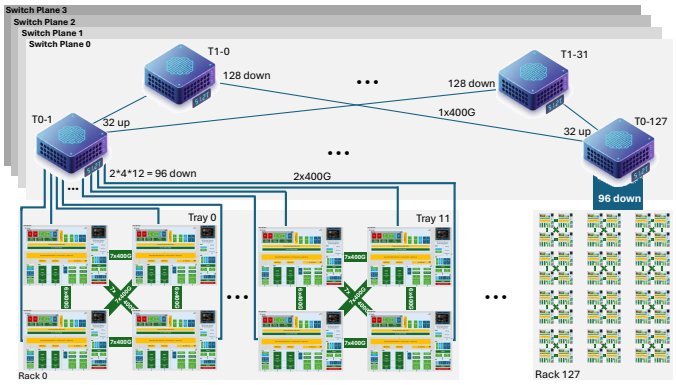

Fig. 7. Maia 200's Two-Tier Network Topology.

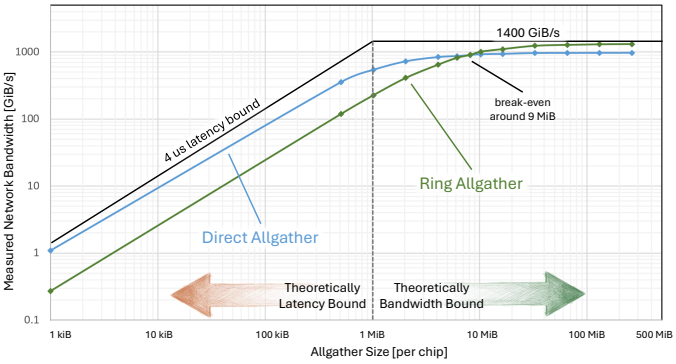

Fig. 8. Allgather Benchmark Results on 8 Maia 200 Chips in Two Trays

architecture and specifically Maia 200's system design running production workloads.

The direct connect algorithm simply sends the right pieces of the data directly to the target endpoints. The ring algorithm established a logical one-dimensional ring connecting all chips and sends chunks of data in a pipelined manner along the ring such that all accelerators send and receive at all times. Details on those standard algorithms can be found in Thakur et al. [69]. The key difference between them is that the direct connect algorithm has a message depth (longest path of depending message synchronizations) of one: no message depends on another message to be received. Thus, direct connect has the lowest latency. Yet, it may not utilize the bandwidth most efficiently. Ring on the other hand has the longest message depth of P-1 for communicators with P chips but is well-known to be bandwidth efficient for large messages. Thus, we expect the direct algorithm to perform best for small messages and the ring algorithm for large.

Furthermore, we establish the *speed of light (SoL)* bounds that either algorithm cannot exceed using a simple assessment based on latency and bandwidth. We estimate the maximum small-message latency of the network to be around 4us when traversing through the switched part. Fig. 7 shows the bandwidth configuration. While the tray design may seem asymmetric, i.e., north-south links can carry only 300 GB/s while east-west and diagonal can carry 350 GB/s, we note that the switched part of the network (400 GB/s) can flexibly

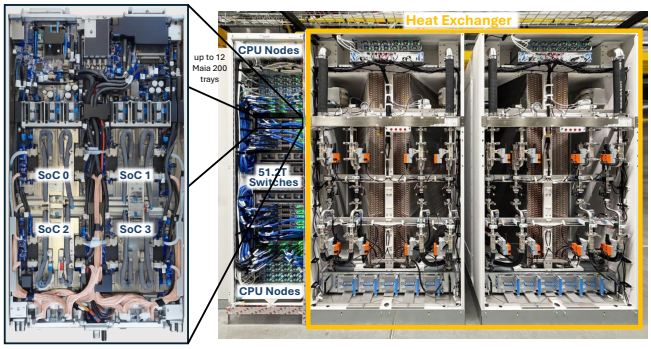

Fig. 9. Maia 200 Tray (left) and Rack for Deployment in Air-Cooled Environments (right)

be used to balance the bandwidth by using 50 GB/s bandwidth to strengthen the north-south links. This would lead to a fully balanced system of 350 GB/s for all four directions, leading to a total of 1.4 TB/s balanced network bandwidth. This switched design makes configurations more flexible and allows programmers to adjust mappings to different traffic requirements [27].

We do not intend to provide a fully exhaustive benchmarking study but rather demonstrate the efficacy of SDLA for real-world workloads. Thus, we focus our analysis on one of the most important collective operations for LLM inference: Allgather is a central component in Fully Sharded Data Parallel LLM inference [64], [78] and 3D parallelism using narrow datatypes [56]. In Allgather, all chips collect data from all other chips (e.g., weights or activations). Specifically, for a communicator of size P and a communication volume of N Bytes per process, each chip receives $R = N(P-1)$ Bytes. The speed of light for this operation would thus be $SoL = \max\left(4us, \frac{R}{1.4\,TiB/s}\right)$. Fig. 8 shows the speed of light as upper bound to the bandwidth (in black) and measured benchmark results on 8 chips for relevant data sizes in our production workloads. We achieve 78% of the latency bound as well as 94% of the bandwidth bound defined by the architectural limits, meeting or exceeding similar architectures. In practice, AI workloads run a multi-algorithm through a collective communication library similar to NCCL [31] or MCCL [32] that chooses the fastest implementation based on the input parameters.

2) Azure integration: Maia 200 is fully integrated into Microsoft's Azure datacenters. The package is liquid cooled and can be connected to liquid cooling datacenter infrastructure. A tray is shown in the left part of Fig. 9. Since liquid cooling is not standard in most datacenters yet, Maia 200 can be deployed in air-cooled datacenters by using an integrated heat exchanger as shown in the right part of Fig. 9. Maia 200 uses standard Ethernet cabling and switches for all networking. This deep integration into existing Azure infrastructure contributes to the total TCO savings.

TABLE I
QWEN 2.5 7B MATRIX MULTIPLICATION AND LOAD SIZES

Operation	Shape [N x K x M]	Count	Size [MiB]
k_/v_projection	S x 3584 x 512	56	3.67
q_/o_projection	S x 3584 x 3584	56	25.69
up_/gate_projection	S x 3584 x 18 944	56	135.79
down_projection	S x 18 944 x 3584	28	135.79
logits	S x 3584 x 151 936	1	1089.08

IV. END-TO-END GENERATIVE INFERENCE IN A REAL-WORLD SETTING

In closing, we demonstrate a complete end-to-end example for inference running a public model, Qwen 2.5 7B [55] on a Maia 200 chip using all SDLA features described before. The model has 28 layers with an inner dimension of size 3584 and a feed-forward intermediate size of 18 944. It has 28 attention heads using grouped query attention with 4 KV heads. We write a matrix multiplication that consumes two input matrices of size N x K and K x M, respectively as shape N x K x M. Thus, the key and value projection matrix multiplications for a sequence with S entries are shaped S x 3584 x 512 and the query and output projection matrix multiplications are shaped S x 3584 x 3584. The feed-forward layer's up projection uses a SwiGLU activation function, which combines two matrix multiplications shaped S x 3584 x 18 944 with a down projection of shape S x 18 944 x 3584. The final logits head projects to the vocabulary size with a matrix multiplication of shape S x 3584 x 151 936. Table I shows the matrix multiplications and their shapes, equaling a total size of 14.14 GiB for all weights.

For our analysis, we focus on the most important and most challenging memory-bound token generation phase. The arithmetic intensity in this case, where S=1, makes the workload fully memory bound and loading 112 small matrices of less than 26 MB and 84 medium-sized matrices of 136 MB and one large matrix of about 1 GB. The table also shows the expected load bandwidth according to our roofline plot in Fig. 4. We analyze inference for the case of a long generation of 16 384 previous tokens to generate the 16 385th token autoregressively. The size of the KV cache in this case is 2 (K, V) x 28 (layers) x 4 (heads/layer) x 128 (dim per head) x 2 (Bytes / dim) x 16 384 = 939.52 MiB. Our straight-forward implementation orchestrated with PyTorch calling standard kernels without additional fusing achieves 2434 tokens/s, which is more than 70% of the estimated maximum performance. We also validated our implementation for correctness with respect to existing GPU inference solutions.

This demonstrates that Maia 200's implementation of the software-defined locally accessed (SDLA) dataflow architecture principle performs well not only close to peak for single matrix multiplication or collective operations but also for complex end-to-end practical inference workloads. This first feasibility study and implementation open the door for many additional optimizations to inline operators minimizing data movement. While we achieve good baseline performance, SDLA enables ninja programmers to explicitly manage every memory allocation and movement and orchestrate asynchronous control codes to achieve near-100% utilization of all execution units even for complex workloads similar to what we demonstrated for matrix multiplications.

V. RELATED WORK

Many new AI accelerators are entering the market to serve the quickly growing demand of compute cycles. In addition to many independent solutions [12], [42], several datacenter providers have their own designs: AWS' Trainium and Inferentia [22], Google's TPUs [34], [35], and Meta's MTIA [13]. Established GPU accelerators follow suit and specifically optimize for AI and LLM workloads. Our work follows and extends this lineage of existing accelerators towards software-defined dataflow using localized memories.

One could argue that modern GPUs are moving in the same direction: NVIDIA's Tensor Memory Accelerator (TMA), introduced in Hopper GPUs [45], can be seen as comprising specialized memory and data movement engines to be hand-orchestrated by programmers. Yet, TMA interacts in non-trivial ways with the GPUs SIMT/warp system leading to a complete redesign of Hopper's warp-group MMA (WGMMA) to Unified (UMMA) in Blackwell, breaking backwards compatibility [51]. In SDLA, we made explicit and localized memory programming a foundational principle that fits seamlessly from day one leading to improved energy efficiency.

Furthermore, one can argue that performance-aware AI programmers are moving towards more explicit, manually-scheduled approaches where they work around hidden architecture features. One of the prime examples is the series of four Flash Attention papers that carefully adjust the kernels to each new GPU architecture in a scientific paper each [15], [16], [62], [77]. Such manually-scheduled ninja programming is becoming the new norm and frameworks like Triton [70] or DaCe [7] attempt to fill the resulting productivity gap. SDLA explicitly enables lowest-level control to simplify this transition.

VI. DISCUSSION AND CONCLUSIONS

We discussed Software-Defined Locally Addressed Dataflow as a new architectural model to co-design and build efficient AI and HPC accelerator systems. SDLA focuses on data movement and its orchestration as leading design principles. We establish a taxonomy, similar to Flynn's taxonomy that illustrates how SDLA combines two foundational principles: (1) **software-defined dataflow** that explicitly orchestrates the data path, allowing programmers to achieve more than 99% utilization of tensor units by offloading the control and (2) **local data access** using optimized on-chip and system-wide memory placement that allows designers to build and co-design machines to specific dataflow requirements of the application. We also describe and demonstrate Maia 200, a real-world implementation of SDLA deployed in production in Microsoft's fleet today. We show how Maia combines specialized data movement through distributed memories and networks and specialized control

through explicit DMA, Synchronization, and Control units into a highly efficient architecture.

ACKNOWLEDGMENTS

No text in this document was produced by an AI or LLM. Arrows in Figs. 4+5 were generated using a diffusion model.

REFERENCES

[1] Advanced Micro Devices, Inc., *AMD I/O Virtualization Technology (IOMMU) Specification*, AMD, 2023, revision 3.08.

[2] L. Alvarez, L. Vilanova, M. Moreto, M. Casas, M. Gonzàlez, X. Martorell, N. Navarro, E. Ayguadé, and M. Valero, "Coherence protocol for transparent management of scratchpad memories in shared memory manycore architectures," *SIGARCH Comput. Archit. News*, vol. 43, no. 3S, p. 720–732, Jun. 2015. [Online]. Available: https://doi.org/10.1145/2872887.2750411

[3] Arm Limited, *Arm CoreLink DMA-350 Controller Technical Reference Manual*, Arm Limited, 2023, document ID: 102482_0000_04_en. [Online]. Available: https://developer.arm.com/documentation/102482/latest/

[4] D. H. Bailey, "Unfavorable strides in cache memory systems (rnr technical report rnr-92-015)," *Sci. Program.*, vol. 4, no. 2, p. 53–58, Apr. 1995. [Online]. Available: https://doi.org/10.1155/1995/937016

[5] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in numerical linear algebra," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 3, pp. 866–901, 2011. [Online]. Available: https://doi.org/10.1137/090769156

[6] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: a design alternative for cache on-chip memory in embedded systems," in *Proceedings of the Tenth International Symposium on Hardware/Software Codesign. CODES 2002 (IEEE Cat. No.02TH8627)*, 2002, pp. 73–78. [Online]. Available: https://doi.org/10.1145/774789.774805

[7] T. Ben-Nun, J. de Fine Licht, A. N. Ziogas, T. Schneider, and T. Hoefler, "Stateful dataflow multigraphs: a data-centric model for performance portability on heterogeneous architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356173

[8] T. Ben-Nun and T. Hoefler, "Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 65:1–65:43, Aug. 2019. [Online]. Available: https://doi.org/10.1145/3320060

[9] A. Berthold, C. Fürst, A. Obersteiner, and H. Schirmeier, "A quantitative analysis and guidelines of data streaming accelerator in modern intel xeon scalable processors," *IEEE Computer Architecture Letters*, 2025. [Online]. Available: https://doi.org/10.1109/LCA.2024.341204

[10] M. Besta, J. Barth, E. Schreiber, A. Kubicek, A. Catarino, R. Gerstenberger, P. Nyczyk, P. Iff, Y. Li, S. Houliston, T. Sternal, M. Copik, G. Kwaśniewski, J. Müller, Łukasz Flis, H. Eberhard, Z. Chen, H. Niewiadomski, and T. Hoefler, "Reasoning language models: A blueprint," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.11223

[11] T. Bonato, A. Kabbani, A. Ghalayini, M. Papamichael, M. Dohadwala, L. Gianinazzi, M. Khalilov, E. Achermann, D. D. Sensi, and T. Hoefler, "Reps: Recycled entropy packet spraying for adaptive load balancing and failure mitigation," 2026. [Online]. Available: https://doi.org/10.1145/3767295.3769320

[12] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, p. 269–284, Feb. 2014. [Online]. Available: https://doi.org/10.1145/2654822.2541967

[13] J. Coburn, C. Tang, S. A. Asal, N. Agrawal, R. Chinta, H. Dixit, B. Dodds, S. Dwarakapuram, A. Firoozshahian, C. Gao, K. Gondkar, T. Graf, J. Hu, J. Huang, S. Hughes, A. Hutchin, B. Jakka, G. J. Chen, I. Kalyanaraman, A. Kamath, P. Kansal, E. Kazi, R. Levenstein, M. Maddury, A. Mastro, S. Medaiyese, P. Modi, J. Montgomery, S. Nadathur, A. Nagpal, A. Narasimha, M. Naumov, E. Ozer, J. Park, P. Ramani, H. Reddy, D. Reiss, D. Roy, S. Sekar, A. Sharma, P. Shetty, A. Sukumaran-Rajam, E. Tal, M. Tsai, S. Varshini, R. Wareing, O. Wu, X. Xie, J. Yang, H. Yu, T. Zargar, Z. Zeng, F. Zhang, A. Matthews, X. Jiao, J. Zhang, E. Menage, T. E. Stokke, and M. Sourouri, "Meta's second generation ai chip: Model-chip co-design and productionization experiences," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1689–1702. [Online]. Available: https://doi.org/10.1145/3695053.3731409

[14] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998. [Online]. Available: https://doi.org/10.1109/99.660313

[15] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.08691

[16] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2205.14135

[17] D. De Sensi, S. Pasqualoni, L. Piarulli, T. Bonato, S. Ba, M. Turisini, J. Domke, and T. Hoefler, "Bine trees: Enhancing collective operations by optimizing communication locality," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1901–1916. [Online]. Available: https://doi.org/10.1145/3712285.3759835

[18] U. Drepper, "What every programmer should know about memory," Red Hat, Inc., Tech. Rep., 2007. [Online]. Available: https://people.freebsd.org/ lstewart/articles/cpumemory.pdf

[19] A. Dubey, A. Jauhri, A. Pandey *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.21783

[20] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Transactions on Computers*, vol. C-21, no. 9, pp. 948–960, 1972. [Online]. Available: https://doi.org/10.1109/TC.1972.5009071

[21] S. Fortune and J. Wyllie, "Parallelism in random access machines," in *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, ser. STOC '78. New York, NY, USA: Association for Computing Machinery, 1978, p. 114–118. [Online]. Available: https://doi.org/10.1145/800133.804339

[22] X. Fu, Z. Zhang, H. Fan, G. Huang, M. El-Shabani, R. Huang, R. Solanki, F. Wu, R. Diamant, and Y. Wang, "Distributed training of large language models on aws trainium," in *Proceedings of the 2024 ACM Symposium on Cloud Computing*, ser. SoCC '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 961–976. [Online]. Available: https://doi.org/10.1145/3698038.3698535

[23] L. Gianinazzi, T. Ben-Nun, and T. Hoefler, "Spada: A spatial dataflow architecture programming language," *arXiv preprint arXiv:2511.09447*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2511.09447

[24] D. Goel, M. C. Heddes, T. Hoefler, and X. Xu, "Message communication between integrated computing devices," U.S. Patent US11 886 938B2, January 30, 2024.

[25] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, no. 3, p. 431–473, May 1996. [Online]. Available: https://doi.org/10.1145/233551.233553

[26] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks," *Journal of Machine Learning Research*, vol. 22, no. 241, pp. 1–124, Sep. 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2102.00554

[27] T. Hoefler, T. Bonato, D. D. Sensi, S. D. Girolamo, S. Li, M. Heddes, J. Belk, D. Goel, and S. S. Miguel Castro, "HammingMesh: A Network Topology for Large-Scale Deep Learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'22)*, Nov. 2022. [Online]. Available: https://doi.org/10.1109/SC41404.2022.00013

[28] T. Hoefler, D. Roweth, K. Underwood, R. Alverson, M. Griswold, V. Tabatabaee, M. Kalkunte, S. Anubolu, S. Shen, M. McLaren, A. Kabbani, and S. Scott, "Data Center Ethernet and Remote Direct Memory Access: Issues at Hyperscale," *IEEE Computer*, vol. 56, no. 7, pp. 67–77, Jul. 2023. [Online]. Available: https://doi.org/10.1109/MC.2023.3267568

[29] T. Hoefler, K. Schramm, E. Spada, K. Underwood, C. Alexander, B. Alverson, P. Bottorff, A. Caulfield, M. Handley, C. Huang, C. Raiciu, A. Kabbani, E. Opsasnick, R. Pan, A. Ran, and R. Sohan, "Ultra ethernet's design principles and architectural innovations," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2508.08906

[30] C. Hong, W. Bao, A. Cohen, S. Krishnamoorthy, L.-N. Pouchet, F. Rastello, J. Ramanujam, and P. Sadayappan, "Effective padding of multidimensional arrays to avoid cache conflict misses," *SIGPLAN Not.*, vol. 51, no. 6, p. 129–144, Jun. 2016. [Online]. Available: https://doi.org/10.1145/2980983.2908123

[31] Z. Hu, S. Shen, T. Bonato, S. Jeaugey, C. Alexander, E. Spada, J. Dinan, J. Hammond, and T. Hoefler, "Demystifying nccl: An in-depth analysis of gpu communication protocols and algorithms," 2026. [Online]. Available: https://doi.org/10.48550/arXiv.2507.04786

[32] C. Hwang, P. Cheng, R. Dathathri, A. Jangda, S. Maleki, M. Musuvathi, O. Saarikivi, A. Shah, Z. Yang, B. Li, C. Rocha, Q. Zhou, M. Ghazimirsaeed, S. Anantharamu, and J. Jose, "Msccl++: Rethinking gpu communication abstractions for ai inference," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ACM, Mar. 2026, p. 1201–1215. [Online]. Available: https://doi.org/10.1145/3779212.3790188

[33] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, "Data Movement Is All You Need: A Case Study on Optimizing Transformers," in *Proceedings of Machine Learning and Systems 3 (MLSys 2021)*, 04 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2007.00072

[34] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. A. Patterson, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589350

[35] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–12. [Online]. Available: https://doi.org/10.1145/3079856.3080246

[36] A. Kabbani and T. Hoefler, "Recycled entropies packet spraying," U.S. Patent US 12,255,824, March 18, 2025.

[37] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, "Competitive snoopy caching," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 1986, pp. 244–254. [Online]. Available: https://doi.org/10.1109/SFCS.1986.14

[38] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, *Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling*, 2020. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00047

[39] R. Komuravelli, M. D. Sinclair, J. Alsop, M. Huzaifa, M. Kotsifakou, P. Srivastava, S. V. Adve, and V. S. Adve, "Stash: Have your scratchpad and cache it too," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 707–719. [Online]. Available: https://doi.org/10.1145/2749469.2750374

[40] G. Kwasniewski, M. Kabić, M. Besta, J. VandeVondele, R. Solcà, and T. Hoefler, "Red-Blue Pebbling Revisited: Near Optimal Parallel Matrix-Matrix Multiplication," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC19)*, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356201

[41] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 611–626. [Online]. Available: https://doi.org/10.1145/3600006.3613165

[42] S. Lie, "Cerebras architecture deep dive: First look inside the hw/sw co-design for deep learning," *IEEE Micro*, vol. 43, no. 3, pp. 30–41, 2023. [Online]. Available: https://doi.org/10.1109/MM.2023.3262524

[43] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, p. 39–55, Mar. 2008. [Online]. Available: https://doi.org/10.1109/MM.2008.31

[44] J. D. Little, "A proof for the queuing formula: $l = \lambda w$," *Operations Research*, vol. 9, no. 3, pp. 383–387, 1961. [Online]. Available: https://doi.org/10.1287/opre.9.3.383

[45] W. Luo, R. Fan, Z. Li, D. Du, Q. Wang, and X. Chu, " Benchmarking and Dissecting the Nvidia Hopper GPU Architecture ," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2024, pp. 656–667. [Online]. Available: https://doi.org/10.1109/IPDPS57955.2024.00064

[46] T. Marinelli, J. I. Gómez Pérez, C. Tenllado, and F. Catthoor, "Compad: A heterogeneous cache-scratchpad cpu architecture with data layout compaction for embedded loop-dominated applications," *J. Syst. Archit.*, vol. 145, no. C, Dec. 2023. [Online]. Available: https://doi.org/10.1016/j.sysarc.2023.103022

[47] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard, Version 4.1*, November 2023. [Online]. Available: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf

[48] F. Nina Paravecino, M. E. Davies, A. D. Kulkarni, M. A. Raihan, A. More, A. Ankit, T. Hoefler, and D. C. Burger, "Assigning workloads to physical resources in spatial architectures," U.S. Patent Application US 2024/0 303 117 A1, September 12, 2024.

[49] NVIDIA Corporation, *NVIDIA Blackwell Architecture Technical Overview: Unlocking the Next Era of Generative AI*, March 2024, introduces the NVFP4 format and 5th-generation Tensor Cores with microscopic scaling. [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-b200/

[50] NVIDIA Corporation, *Parallel Thread Execution (PTX) ISA Version 9.2*, NVIDIA Corporation, March 2026. [Online]. Available: https://docs.nvidia.com/cuda/parallel-thread-execution/index.html

[51] NVIDIA Corporation, *Parallel Thread Execution (PTX) ISA, Version 9.2*, NVIDIA Corporation, March 2026, available at https://docs.nvidia.com/cuda/parallel-thread-execution/index.html.

[52] Open Compute Project Foundation, "Ocp microscaling formats (mx) specification v1.0," OCP MX Alliance, Tech. Rep., October 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.10537

[53] C. Payne, J. Smith, and M. Hernandez, "Colossus at one gigawatt: Inside the new ai infrastructure race for power, compute, and control," ResearchGate, White Paper, 2026.

[54] R. Prabhakar, R. Sivaramakrishnan, D. Gandhi, Y. Du, M. Wang, X. Song, K. Zhang, T. Gao, A. Wang, X. Li, Y. Sheng, J. Brot, D. Sokolov, A. Vivek, C. Leung, A. Sabnis, J. Bai, T. Zhao, M. Gottscho, D. Jackson, M. Luttrell, M. K. Shah, Z. Chen, K. Liang, S. Jain, U. Thakker, D. Huang, S. Jairath, K. J. Brown, and K. Olukotun, "Sambanova sn40l: Scaling the ai memory wall with dataflow and composition of experts," in *Proceedings of the 2024 57th IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '24. IEEE Press, 2024, p. 1353–1366. [Online]. Available: https://doi.org/10.1109/MICRO61859.2024.00100

[55] Qwen, :, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, "Qwen2.5 technical report," 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2412.15115

[56] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: memory optimizations toward training trillion parameter models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020. [Online]. Available: https://doi.org/10.1109/SC41405.2020.00024

[57] P. Rajhavan and M. Snir, "Memory versus randomization in on-line algorithms," *IBM Journal of Research and Development*, vol. 38, no. 6, pp. 683–707, 1994. [Online]. Available: https://doi.org/10.1147/rd.386.0683

[58] A. Rico, S. Pareek, J. Cabezas, D. Clarke, B. Ozgul, F. Barat, Y. Fu, S. Münz, D. Stuart, P. Schlangen, P. Duarte, S. Date, I. Paul, J. Weng, S. Santan, V. Kathail, A. Sirasao, and J. Noguera, "Amd xdna npu in ryzen ai processors," *IEEE Micro*, vol. 44, no. 6, p. 73–82, Nov. 2024. [Online]. Available: https://doi.org/10.1109/MM.2024.3423692

[59] G. Rivera and C.-W. Tseng, "Data transformations for eliminating conflict misses," in *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation*, ser. PLDI '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 38–49. [Online]. Available: https://doi.org/10.1145/277650.277661

[60] K. Roy, A. Kosta, T. Sharma, S. Negi, D. Sharma, U. Saxena, S. Roy, A. Raghunathan, Z. Wan, S. Spetalnick, C.-K. Liu, and A. Raychowdhury, "Breaking the memory wall: next-generation artificial intelligence hardware," *Frontiers in Science*, vol. Volume 3 - 2025, 2025. [Online]. Available: https://doi.org/10.3389/fsci.2025.1611658

[61] P. Scheffler, F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Sparse Stream Semantic Registers: A Lightweight ISA Extension Accelerating General Sparse Linear Algebra," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 12, pp. 3147–3161, Oct. 2023. [Online]. Available: https://doi.org/10.1109/TPDS.2023.3323497

[62] J. Shah, G. Bikshandi, Y. Zhang, V. Thakkar, P. Ramani, and T. Dao, "Flashattention-3: fast and accurate attention with asynchrony and low-precision," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, ser. NIPS '24. Red Hook, NY, USA: Curran Associates Inc., 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.08608

[63] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer." in *ICLR (Poster)*. OpenReview.net, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1701.06538

[64] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," 2020. [Online]. Available: https://doi.org/10.48550/arXiv.1909.08053

[65] A. J. Smith, "Line (block) size choice for cpu cache memories," *IEEE Trans. Comput.*, vol. 36, no. 9, p. 1063–1076, Sep. 1987. [Online]. Available: https://doi.org/10.1109/TC.1987.5009537

[66] H. Stengel, J. Treibig, G. Hager, and G. Wellein, "Quantifying performance bottlenecks of stencil computations using the execution-cache-memory model," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ser. ICS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 207–216. [Online]. Available: https://doi.org/10.1145/2751205.2751240

[67] T. R. Sumers, S. Yao, K. Narasimhan, and T. L. Griffiths, "Cognitive architectures for language agents," 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2309.02427

[68] X. Tao, J. Pang, J. Xu, and Y. Zhu, "Compiler-directed scratchpad memory data transfer optimization for multithreaded applications on a heterogeneous many-core architecture," *The Journal of Supercomputing*, vol. 77, pp. 14 502 – 14 524, 2021. [Online]. Available: https://doi.org/10.1007/s11227-021-03861-1

[69] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, p. 49–66, Feb. 2005. [Online]. Available: https://doi.org/10.1177/1094342005051521

[70] P. Tillet, H. T. Kung, and D. Cox, "Triton: an intermediate language and compiler for tiled neural network computations," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, ser. MAPL 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 10–19. [Online]. Available: https://doi.org/10.1145/3315508.3329973

[71] Ultra Ethernet Consortium, *Ultra Ethernet Site Architecture Specification v1.0*, Linux Foundation, September 2024, defines the UEC Transport (UET) layer, Physical layer, and Software stack for AI/HPC. [Online]. Available: https://ultraethernet.org/specifications/

[72] K. Vaidyanathan, W. Huang, S. Narravula, and D. K. Panda, "Benefits of i/o acceleration technology (i/oat) in clusters," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2007, pp. 220–229. [Online]. Available: https://doi.org/10.1109/ISPASS.2007.363752

[73] D. Van Essendelft, P. Wingo, T. Jordan, and R. Smith, "A system level compiler for massively-parallel, spatial, dataflow architectures," *arXiv preprint arXiv:2506.15875*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2506.15875

[74] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, p. 65–76, Apr. 2009. [Online]. Available: https://doi.org/10.1145/1498765.1498785

[75] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Q. Liu and D. Schlangen, Eds. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://doi.org/10.18653/v1/2020.emnlp-demos.6

[76] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, p. 20–24, Mar. 1995. [Online]. Available: https://doi.org/10.1145/216585.216588

[77] T. Zadouri, M. Hoehnerbach, J. Shah, T. Liu, V. Thakkar, and T. Dao, "Flashattention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling," 2026. [Online]. Available: https://doi.org/10.48550/arXiv.2603.05451

[78] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li, "Pytorch fsdp: Experiences on scaling fully sharded data parallel," *Proc. VLDB Endow.*, vol. 16, no. 12, p. 3848–3860, Aug. 2023. [Online]. Available: https://doi.org/10.14778/3611540.3611569

[79] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng, "Sglang: efficient execution of structured language model programs," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, ser. NIPS '24. Red Hook, NY, USA: Curran Associates Inc., 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2312.07104

[80] A. N. Ziogas, G. Kwasniewski, T. Ben-Nun, T. Schneider, and T. Hoefler, "Deinsum: Practically I/O Optimal Multilinear Algebra," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'22)*, Nov. 2022. [Online]. Available: https://doi.org/10.1109/SC41404.2022.00018