Cerebras Architecture Deep Dive: First Look Inside the Hardware/Software Co-Design for Deep Learning

Sean Lie, *Cerebras Systems, Sunnyvale, CA, 94085, USA*

The compute and memory demands for deep learning and machine learning (ML) have increased by several orders of magnitude in just the last couple of years, and there is no end in sight. Traditional improvements in processor performance alone struggle to keep up with the exponential demand. A new chip architecture co-designed with the ML algorithms can be better equipped to satisfy this unprecedented demand and enable the ML workloads of the future. This article describes the Cerebras architecture and how it is designed specifically with this purpose, from the ground up, as a wafer-sized chip to enable emerging extreme-scale ML models. It uses fine-grained data flow compute cores to accelerate unstructured sparsity, distributed static random-access memory for full memory bandwidth to the data paths, and a specially designed on-chip and off-chip interconnect for ML training. With these techniques, the Cerebras architecture provides unique capabilities beyond traditional designs.

The field of modern machine learning (ML) is relatively new, and we are already reaching a pace that is often limited by traditional approaches to training and inference. In 2018, state-of-the-art neural networks such as BERT[1] had 100 million parameters. Two years later, the famous GPT-3[2] had 175 billion parameters. There is no end in sight; next, the ML community is striving to run models with trillions of parameters. That represents more than 1000× growth in compute demand in just two years, as shown in Figure 1.

This is the grand ML demand challenge in front of the industry. At Cerebras, we believe that we can meet this unprecedented demand. We cannot do it by relying on just a single solution. It must be addressed by making substantial improvements—by an order of magnitude or more—across a broad spectrum of multiple different components. To meet this unprecedented demand, we target order-of-magnitude improvements in three key areas:

- *Core architecture*: to enable performance scaling beyond raw floating-point operations per second (flops) alone.
- *Scale-up*: to accelerate process improvements beyond Moore's law.
- *Scale-out*: to improve and simplify distributed clustering.

We believe all of these are required to keep up with this unprecedented demand. To achieve improvements in all three areas, our architecture is co-designed from the ground up specifically for neural networks. In the following sections, we describe each area in detail.

0272-1732 © 2023 IEEE
Digital Object Identifier 10.1109/MM.2023.3256384
Date of current version 15 May 2023.

OVERVIEW

The foundation of the Cerebras architecture is the first-ever wafer-sized processor chip, called the wafer-scale engine (WSE), which was released in 2019. The second-generation WSE-2 was released in 2021 and is focus of this article. It is the largest processor ever built, at 46,000 mm^2 in size and containing 2.6 trillion transistors. Surrounding each WSE-2 is a custom-built system called the CS-2. Within each CS-2 and WSE-2, a wafer-scale on-chip fabric connects 850,000 cores that are specifically designed for ML workloads. In contrast to traditional processors, the Cerebras core uses fine-grained data flow to accelerate unstructured sparsity, which makes it particularly well suited for neural networks, which often have natural sparsity or can be sparsified. Memory is entirely distributed static

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

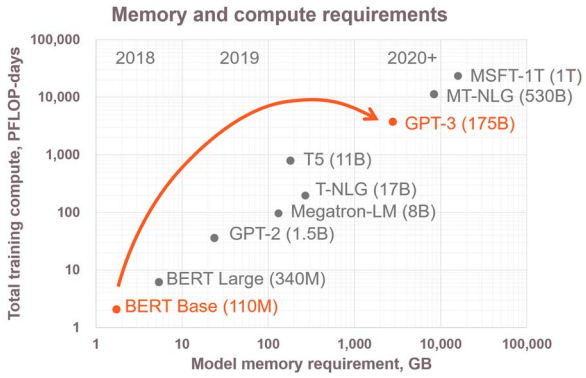


FIGURE 1. Memory and compute requirements for various state-of-the-art neural networks on a log–log scale. PFLOP: peta-floating-point operations per second.

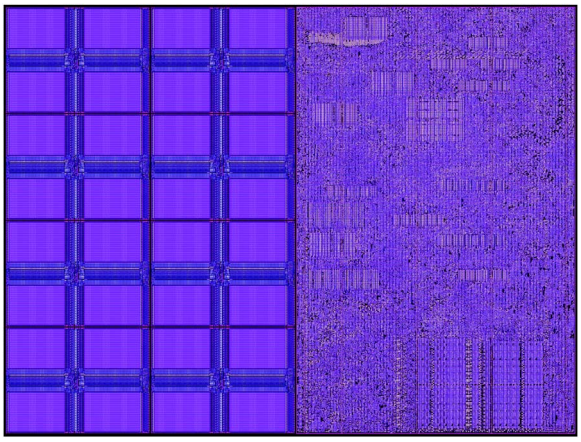


FIGURE 2. The Cerebras core physical design: 50% of the area is static random-access memory (SRAM) and 50% of the area is logic.

random-access memory (SRAM) instead of dynamic RAM (DRAM) to provide full memory bandwidth to the every core's data path. With so many cores and sparsity acceleration, many large ML models can be solved with a single WSE-2 alone. However, to address emerging extreme-scale ML models, the Cerebras architecture uses an off-chip fabric to extend to a cluster of CS-2 systems. The cluster architecture leverages the scale of each CS-2 to provide simple data parallel-only scaling to the larger models and larger clusters.

CORE ARCHITECTURE

The core is specially designed for the fine-grained dynamic sparsity in neural networks. The core physical design is shown in Figure 2. It is a small-core design, at approximately 38,000 μm^2 of silicon area in the Taiwan Semiconductor Manufacturing Company (TSMC) 7 μm process. Half of the silicon area is used by 48 kB of memory. The other half is logic, made up of 110,000 standard cells. The entire core runs at 1.1-GHz clock frequency and consumes 30 mW of peak power.

Memory

Traditional memory architectures modern processors, such as AMD Milan[3] or Nvidia Hopper,[4] use shared central DRAM, but DRAM is both relatively slow and far away when compared to the compute performance. Even with advanced techniques like high-bandwidth memory (HBM3),[5] the relative bandwidth from memory is significantly lower than the core data path bandwidth. For example, it is common for GPU compute data paths to have 100 times more bandwidth than DRAM memory bandwidth. This means each operand from a memory must be used at least 100 times in the data path to keep utilization high. The traditional way to handle this is by using data reuse through local caching or local registers and accumulators.

Instead of the traditional approach, the Cerebras architecture provides full memory bandwidth to all data paths at full performance, removing the need for data reuse. This is accomplished by fully distributing the memory right next to where it is used, which enables memory bandwidth that is equal to the operand bandwidth of the core data path. To make this possible, we take advantage of the physical properties of shorter distance communication. Driving the bits only tens of microns from local memory to the data path, all on silicon, is much easier than through a package to an external device.

Each small core has 48 kB of local SRAM dedicated to the core. That memory is designed to optimize density while providing full performance to the data path: 192-bit access per cycle from two 64-bit reads and one 64-bit write. This is achieved by organizing the memory into eight single-ported banks that are each 32 bits wide, as shown in Figure 3. With that organization, there is 256-bit access per cycle providing more raw memory bandwidth than the data path. The 32-bit bank size was chosen to achieve high density while minimizing bank conflicts in typical ML workloads. It is important to note that all of the memory is independently addressed per core. There is no shared memory in the traditional sense. To enable truly scalable memory, all of the sharing between cores is done explicitly through the fabric. In addition to the high-performance SRAM, there is also a small 256-B software-managed cache that is used for frequently accessed data structures, such as accumulators. This cache is designed to be

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

physically very compact and close to the data path to minimize power for these frequent accesses.

The SRAM uses 50% of the die area, which is in line with the area used for traditional caches, register files, and memory controllers in CPU or GPU designs. Per unit area, our SRAM memory design is higher density than traditional caches because it is not a cache; it does not have overhead such as multiporting, tagging, and cache management logic. Additionally, by not using DRAM, our design also does not incur the die area and power of double data rate or HBM memory controllers, which can be large, especially for high-bandwidth designs. The most significant tradeoff of using SRAM instead of DRAM is in capacity because SRAM density is lower than that of DRAM. The wafer-scale integration is used to make up the capacity to achieve DRAM equivalent capacity per chip.

With this distributed memory architecture and DRAM equivalent capacity, the design can be considered an in-memory compute architecture that achieves significantly higher aggregate memory bandwidth than traditional shared DRAM memory architectures. Normalizing to the GPU area, the Cerebras memory bandwidth is approximately 200 times higher within the same silicon area,[a] directly to the data paths.

Full Performance at All Basic Linear Algebra Subroutine (BLAS) Levels

With such high memory bandwidth, the design is capable of running matrix operations out of memory at full performance across all BLAS levels, as shown in Figure 4. Traditional CPU and GPU architectures with limited off-chip memory bandwidth are limited to running only general matrix–matrix multiply (GEMM) operations at full performance. Any BLAS level below full GEMM requires a large jump in memory bandwidth because there is less data reuse. This becomes challenging with traditional architectures, but with enough memory bandwidth, we can enable full performance all the way down to vector–scalar multiply (AXPY). Within neural network computation, this capability is important because it enables fully unstructured sparsity acceleration. A sparse GEMM is simply a collection of AXPY operations, with one operation for every nonzero element, making this level of memory bandwidth a prerequisite for unstructured sparsity acceleration.

[a]GPU normalized comparison is the WSE-2 memory bandwidth (400 TB/s) in equivalent area against that of the Nvidia A100 (2 TB/s).

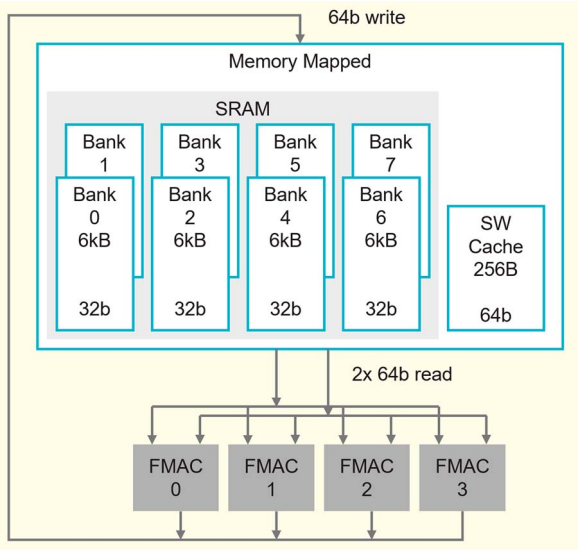


FIGURE 3. The Cerebras core memory design. Memory is organized into 8× 6-kB SRAM banks with an additional 256 B of software-managed cache. SW: software; FMAC: fused multiply-accumulate.

Programmability

The foundation of the Cerebras core is a fully general processor that can be programmed to adapt to the changing field of ML. Like any general-purpose processor, it supports a full set of general-purpose instructions that include arithmetic, logical, load/store, compare, and branch instructions. These instructions are fully local to each core, stored in the same 48 kB of local memory as the data. This is important because it means that every core is independent, which enables very fine-grained, dynamic computation globally across the entire chip. These general-purpose instructions operate on 16 general-purpose registers, and they run in a compact six-stage pipeline. For ML workloads, all instructions are statically loaded upfront using a configuration interface, and they do not change during execution. Given that the same instructions are reused repeatedly (which could be millions of times in neural network training), the time to load the instructions does not appreciably impact overall performance.

On top of this general-purpose foundation, there is hardware support for tensor instructions that are intended for all data processing. These tensor operations execute on the underlying 64-bit data path, which is made up of four 16-bit fused-multiply-accumulate (FMAC) units supporting both floating point (FP)16 and brain floating point (BF)16 data types. The FMAC accumulator optionally supports either the native 16-bit

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

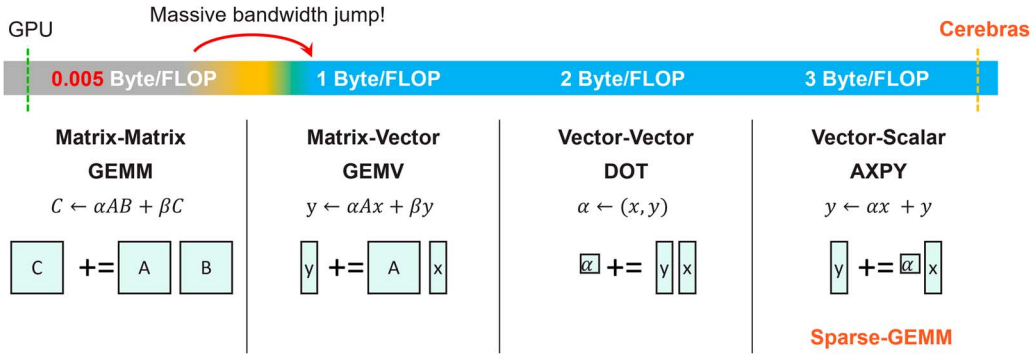


FIGURE 4. BLAS levels of linear algebra computation and memory bandwidth requirements for FP16 flops (1 MAC = 2 flops). Note that Sparse-GEMM is comprised of one AXPY per nonzero matrix element. AXPY: vector–scalar multiply; BLAS: basic linear algebra subroutine; GEMM: general matrix–matrix; MAC: multiply–accumulate; FP16: floating point 16 bit; GEMM: general matrix-matrix multiply; GEMV: general matrix-vector multiply; DOT: dot product.

```
fmach [fpsum] = [fpsum], [fwd_wgt], r_in
        ↑         ↑          ↑        ↑
        3D        3D         2D     scalar
```

FIGURE 5. Example of FMAC instruction with tensors as first-class operands.

format or FP32 data type. To optimize for performance and flexibility, the instruction set architecture has tensors as first-class operands, just like general-purpose registers or memory. Figure 5 shows an example of an FMAC instruction operating on 3-D and 2-D tensors as operands directly.

The core uses data structure registers (DSRs) as operands to the instructions. Our core has 44 of these DSRs, each of which contains a descriptor with a pointer to the tensor and other key information, such as length, shape, and size of that tensor. With these DSRs, the hardware architecture is flexible enough to natively support up to 4-D tensors that are in memory; fabric-streaming tensors; first-in, first-out buffers; or circular buffers. Behind the scenes, there are hardware state machines that use the DSRs and sequence through the full tensor at full performance on the data path.

Instructions for each individual core are generated by the Cerebras software stack and compiler,[b] which lowers ML programs from frameworks, such as TensorFlow or PyTorch.

[b]Cerebras software resources can be found at https://www.cerebras.net/product-software/.

Fine-Grained Data Flow Scheduling

On top of these tensor operations, the core uses fine-grained data flow scheduling. All computation is triggered by the data, as shown in Figure 6. The fabric transports both the data and the associative control directly in the hardware. Once the cores receive that data, the hardware triggers a lookup of instructions to run. That lookup is entirely based on what is received in the fabric. With this data flow mechanism in the cores, the entire compute fabric is a data flow engine. This enables native sparsity acceleration because it only performs work on nonzero data. We filter out all of the zero data at the sender, so the receiver does not even see it. Only nonzero data are sent, and that is what triggers all of the computation. Not only do we save power by not performing the wasted compute, but we get acceleration by skipping it and moving on to the next useful compute. Since the operations are triggered by single data elements, this accelerates fine-grained fully unstructured sparsity. The level of power saving and acceleration is not capped since this capability enables all forms of sparse ML techniques and gives the ML user control of the performance through algorithmic changes.

To complement the dynamic nature of data flow, the core also supports eight simultaneous tensor operations, which we call microthreads. These are independent tensor contexts that the hardware can switch between on a cycle-by-cycle basis. The scheduler is continuously monitoring the input and the output availability for all of the tensors that are being processed, and it picks an eligible microthread with all of its resources available. Each microthread is also assigned a priority, so, if multiple microthreads are eligible to run, the scheduler will pick the higher priority

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

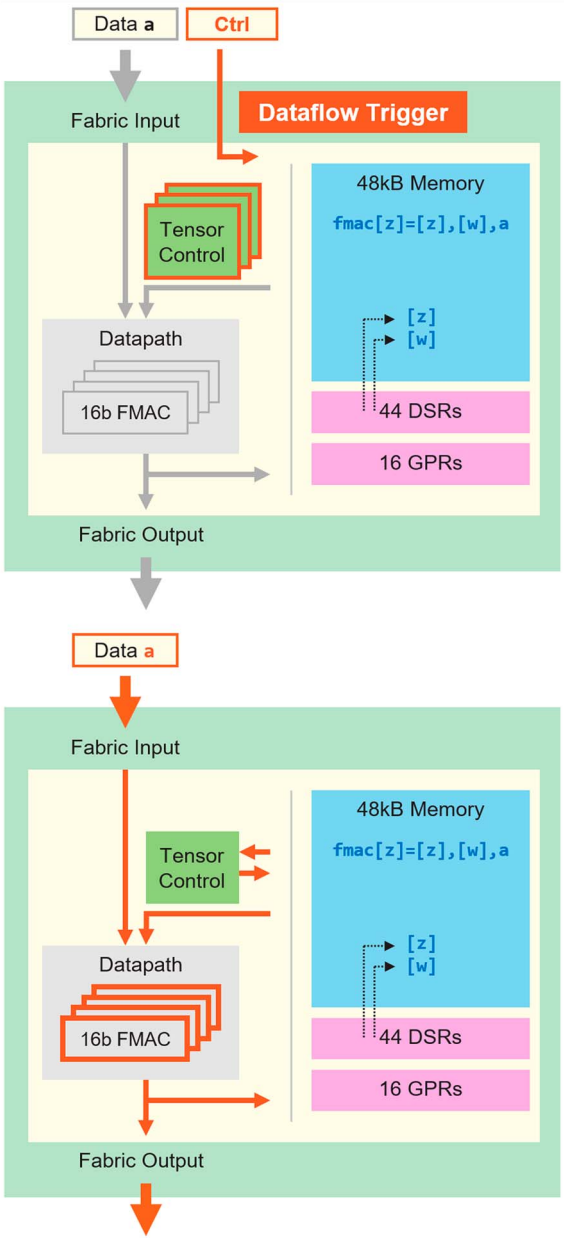


FIGURE 6. Core data path and data flow scheduling to enable acceleration of fine-grained dynamic unstructured sparsity. (Top) Control information from the fabric triggers the core to look up the operation and associated tensor information. (Bottom) The data from the fabric is used directly by the data path to perform the computation, as directed by the control. Ctrl: control; DSR: data structure register; GPR: general purpose register.

microthread. Each microthread has full access to all resources, including registers and memory in the core, without any static partitioning. As with instructions, each core has its own independent microthreads, so context switching is entirely independent per core, enabling it to drive up utilization during dynamic behavior by switching to other tasks independently when there would otherwise be bubbles in the pipeline.

With this fine-grained, dynamic, small-core architecture, we can achieve high compute performance—as high as 10× more utilization than GPUs on unstructured, sparse compute[c] or, potentially, even more with higher sparsity. Coming back to the grand challenge, this sparse utilization advantage is how we target an order-of-magnitude improvement from the core architecture.

SCALE-UP: AMPLIFYING MOORE'S LAW

Traditionally, scaling up within a chip has been the domain of semiconductor foundries. Moore's law[6] has carried the industry for decades, enabling denser and larger chips. Today, Moore's law continues to improve chip scale but only by relatively incremental gains, on the order of 2× improvement per process generation, which, alone, is not enough to satisfy the ML demand. Within the Cerebras architecture, our design goal is to amplify Moore's law and get improvement by an order of magnitude or more.

The traditional way to amplify Moore's law is to make larger chips. We did that with wafer-scale integration, and the result is the WSE-2 fabricated on the TSMC N7 process. The WSE-2 is the largest chip ever built, at 56× larger than the largest GPU today. It is more than 46,000 mm^2 in size, with 2.6 trillion transistors on a single chip, and it contains 850,000 cores. With all of those cores integrated on a single piece of silicon, the single chip has 20 PB/s of memory bandwidth and 220 Pb/s of fabric bandwidth. The WSE-2 improves on the 16-nm design of the first-generation WSE-1, which had fewer than half the number of cores (400,000) as well as associated memory and fabric bandwidth.

We built a specially designed system around the WSE-2, called the CS-2. The system was co-designed around the WSE-2, enabling the wafer-scale chip to be used in a standard data center environment.

To build up to wafer from the small cores, first we create a traditional die with 10,000 cores each. Instead of cutting up the die to make traditional small chips, we keep it intact, but we carve out the largest square within the round 300-mm wafer. The final chip contains a total of 84 dies, with 850,000 cores, all on a single chip, as shown in Figure 7.

[c]GPU utilization comparison is estimated against 10× unstructured sparse compute on the Nvidia A100.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

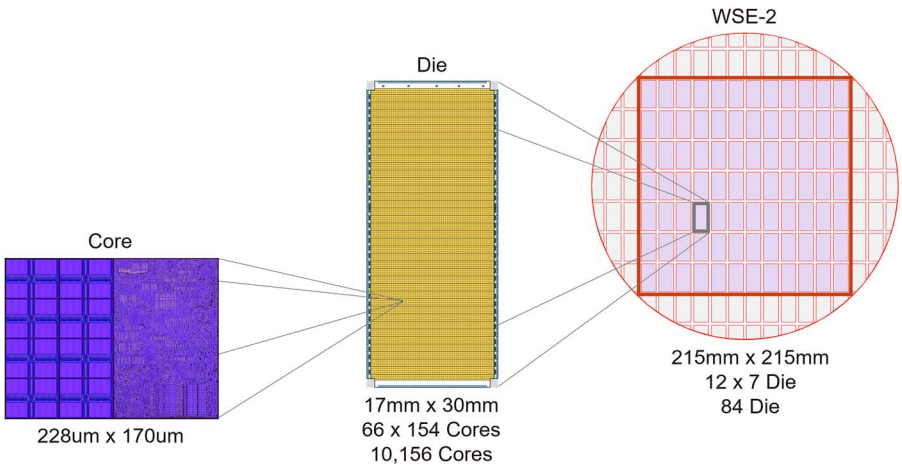


FIGURE 7. Scaling from small core to entire wafer. WSE: wafer-scale engine.

High-Bandwidth, Low-Latency Fabric

The wafer-sized chip is only possible if the underlying architecture can scale to that extreme size. The fundamental enabler is the fabric. It needs to enable efficient and high-performance communication across the entire wafer. Our fabric does this by using a 2-D mesh topology, as shown in Figure 8, that is well suited to scale on silicon with extremely low overhead. This fabric ties together all of the cores, where each core has a fabric router within the mesh topology. The fabric routers have a simple five-port design, with 32-bit bidirectional interfaces in each of the four cardinal directions and one port facing the core itself. Each router uses lossless flow control with low buffering. This design enables single clock cycle latency between nodes while remaining relatively low cost in terms of silicon area and power.

The fundamental data packet is a single 16-bit data element (FP16 or BF16) optimized for neural network training. Along with those 16-bit data, there are 16 bits of control information, making up a 32-bit fine-grained packet. To optimize the fabric design further, it uses

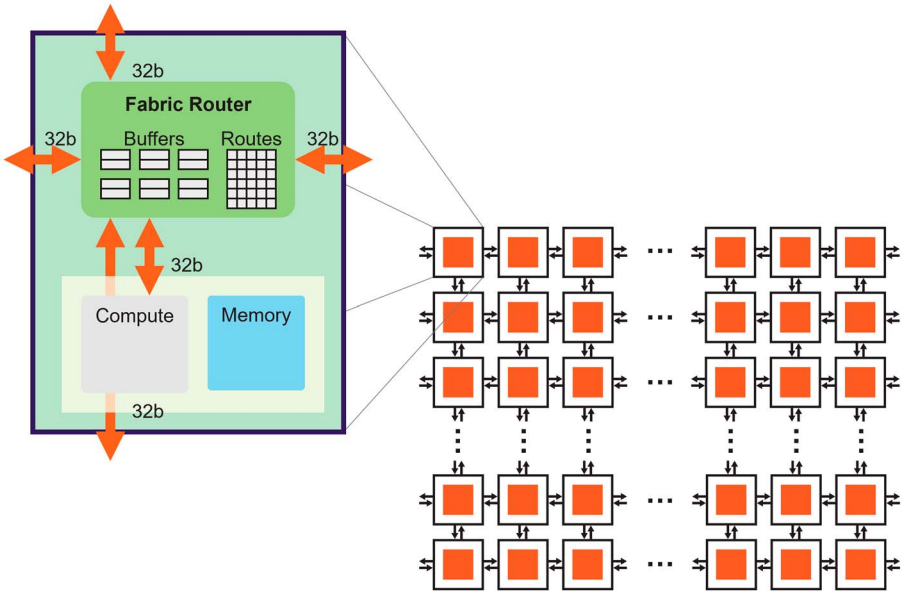


FIGURE 8. High-bandwidth, low-latency fabric. There is a five-port fabric router in each core that has dedicated buffers for each statically configured route (color).

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

	Area	Bandwidth		Power	
	Mm^2	TB/s	$GB/s/mm^2$	pJ/bit	W
GPU Estimate	826	0.6	0.7	10	60
WSE-2 Sub-fabric	826	4.3	5.2	0.15	6
Ratio		7x	7x	66x	10x

FIGURE 9. Comparison of on-wafer fabric interconnect to traditional interconnect in terms of raw bandwidth and power.

entirely static routing, which is efficient and has low overhead while matching to the requirements of neural network static connections. To enable multiple routes on the same physical link, each core has 24 local static routes that can be configured, called colors. All of the colors are nonblocking between one another, and they are all time-multiplexed onto the same physical links. Globally, these local independent colors are combined to support a large number of total routes throughout the entire core array. Additionally, each color is used for a variety of different data and control communications, so the fixed 24 colors enable far more different messages or tasks in total. Finally, neural network communication inherently has a high degree of fan-out, so our fabric is designed with native broadcast and multicast capabilities within each fabric router.

To scale this fabric, we simply extend it out physically in both the x- and y-dimensions. Scaling within a single die is simple. To scale beyond the die, we extend the fabric across die boundaries. The fabric crosses less than a millimeter of scribe line using high-level metal layers within the TSMC process. This extends the 2-D mesh compute fabric to a fully homogeneous array of cores across the entire wafer. The die-to-die interface is an efficient, source-synchronous, parallel interface. Each interface has only a few short wires in high-level metal, making it highly resilient to defects. However, at the wafer scale, all interfaces add up to more than 1 million wires, so defects are unavoidable, and redundancy must be built directly into the underlying protocol. We do that with auto-correction state machines that automatically use redundant links when errors are discovered by training and error detection. Therefore, even with inevitable localize defects in the fabrication process, most die-to-die interfaces remain functional.

These simple short wires are the key enabler of efficient scale-up because they span less than a millimeter of distance on silicon. When compared to traditional serializer/deserializer (SERDES) communication, the difference is significant. Just like the memory, the physical properties mean that driving bits less than a millimeter on the chip is much easier than across package connectors, printed circuit boards, and sometimes even cables. The result is an orders-of-magnitude improvement compared to traditional input–output (IO), as shown in Figure 9. The on-wafer fabric achieves about an order of magnitude more bandwidth per unit area and almost two orders of magnitude better power efficiency per bit. All of these results translates into on-chip-level fabric performance across the entire wafer: 7× more bandwidth than GPU die-to-die bandwidth in the same GPU area at only 6 W of power.[d] This level of global fabric performance is what enables the wafer to operate as a single chip.

JUST LIKE THE MEMORY, THE PHYSICAL PROPERTIES MEAN THAT DRIVING BITS LESS THAN A MILLIMETER ON THE CHIP IS MUCH EASIER THAN ACROSS PACKAGE CONNECTORS, PRINTED CIRCUIT BOARDS, AND SOMETIMES EVEN CABLES.

Weight Streaming Enables the Largest Models

The fabric enables the execution of extremely large neural networks all on a single chip. The WSE-2 has enough performance and capacity to run even the largest state-of-the-art models without partitioning or complex model parallel distribution. This is done by disaggregating the neural network model, the weight memory, from the compute, as shown in Figure 10. We store all of the model weights externally in a device called MemoryX, and we stream all of the weights onto the CS-2 system as they are needed to compute each layer of the network, one layer at a time. Weights are stored in DRAM and flash memory in MemoryX and streamed into the CS-2 at its full IO bandwidth of 1.2 Tb/s. The weights are never stored on the CS-2 system, not even temporarily, because the CS-2 performs the computation using the underlying data flow mechanisms in the cores.

Using data flow scheduling, each individual weight triggers the computation as an individual AXPY operation on a batch of activations resident on the wafer

[d]GPU estimate uses a 7-nm peripheral component interconnect express (PCIe) 5.0 SERDES with the Nvidia A100 NVLink bandwidth. WSE-2 subfabric uses a subset of the wafer area equivalent to that of the Nvidia A100.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

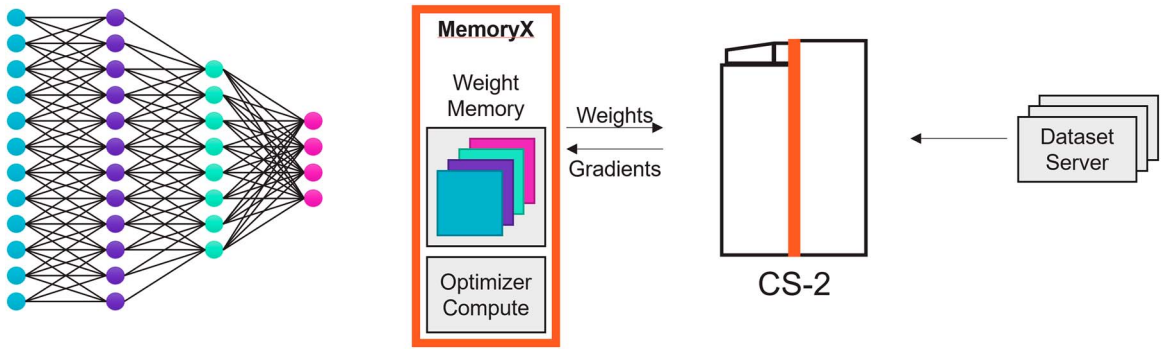


FIGURE 10. Weight streaming disaggregates model weights from compute to enable all model sizes on a single chip. The model (on the left) weights are stored in the external MemoryX unit (shown as colors) and streamed to the CS-2 system.

memory. Once each weight is complete, it is discarded, and the hardware moves on to the next weight without ever storing the weight on the chip. Since the weights are never stored, the size of the model is not constrained by the memory capacity on chip. To amortize the cost of streaming each weight from the external MemoryX, each weight is natively reused multiple times on the CS-2 by using it for a batch of activations. On the backward pass, the gradients are streamed out in the reverse direction back to the MemoryX unit, where the weight updates happen.

Wafer Matrix-Multiply (MatMul) Array

Neural network layers are composed of matrix multiplication. Because of the scale of the WSE-2, we can use all 850,000 cores of the wafer as a single giant MatMul array. For transformer models like GPT, for example, activation tensors have three logical dimensions: batch, sequence, and hidden dimension. We split these tensor dimensions over the 2-D grid of cores on the wafer. The hidden dimension is split over the fabric in the x-direction, and the batch and sequence dimensions are split over the fabric's y-direction, as shown in Figure 11. This arrangement allows for efficient weight broadcast and reductions over the sequence and hidden dimensions.

With activations stored on the cores where the work will be performed, the next step is to trigger the computation on those activations. This is done by using the on-chip broadcast fabric to send the weights, the data, and the commands to each column. Using the hardware data flow mechanisms, the weights then trigger the FMAC operations directly. Since the broadcast occurs over columns, all cores containing the same subset of hidden dimension features receive the same weights. Additionally, we send commands to trigger other computations, such as reductions or nonlinear operations.

More specifically, the MatMul operation starts by broadcasting the first row of weights across the wafer,

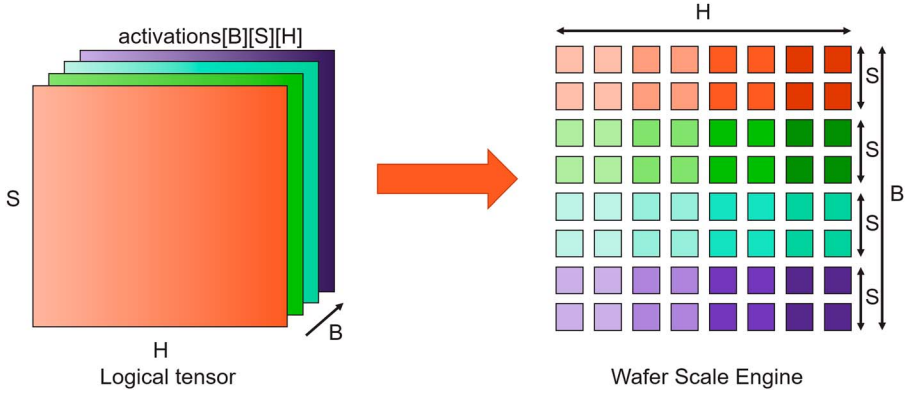


FIGURE 11. Logical tensor distribution onto the physical wafer matrix-multiply (MatMul) array. The hidden (H) dimension is split over the fabric in the x-direction, and the batch (B) and sequence (S) dimensions are split over the fabric's y-direction.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

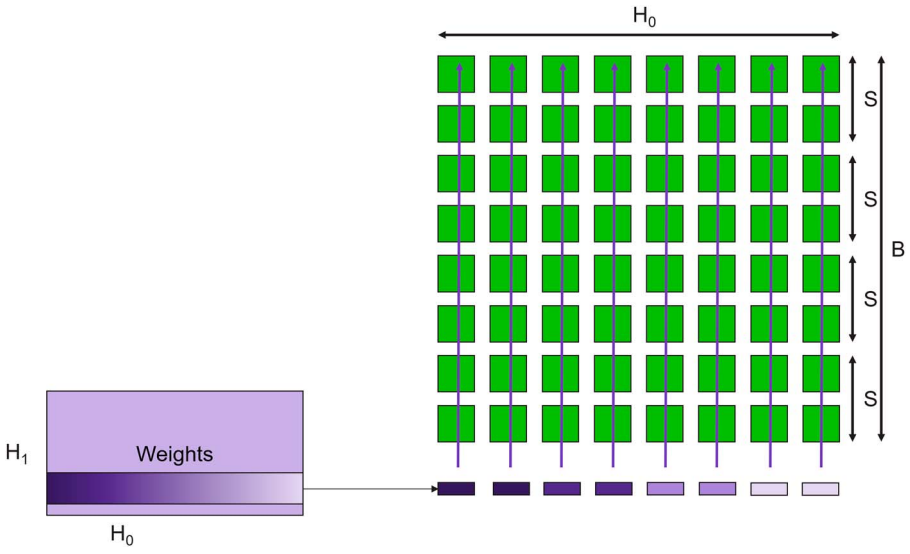


FIGURE 12. Weight broadcast across the wafer MatMul array.

as shown in Figure 12. Within that row, there are multiple weights that map onto a single column. When there is sparsity, only those nonzero weights are broadcasted to the column, triggering FMAC computations only for dense weights. We skip all the zero weights, and we stream in the next nonzero weight. This technique is what produces a sparsity acceleration.

This operation uses several key properties of the core architecture. When a weight arrives, using the hardware data flow scheduling of the core, it triggers an FMAC operation on the data path, as shown in Figure 13. The weight value is multiplied with each of the activations already in SRAM memory and added into a local accumulator in the software-managed cache. The series of multiple FMACs is performed using a single tensor instruction with the activations as a tensor operand. All of these operations are done without additional overhead on the core. Additionally, there is no memory capacity overhead for the weights since, once the compute done, the core moves on to the next weight, never storing any of the weights. Once all weights for the row have been received, each core contains a partial sum that needs to be reduced across the row of cores.

That reduction is then triggered by a command packet broadcasted to all the cores of each column. Again, using the data flow scheduling of the core, once the command packet is received, it triggers the partial sum reduction, also shown in Figure 13. The actual reduction compute is also performed using a single tensor instruction, this time with fabric tensor operands. All of the columns receive a partial sum command (PSUM), except one column receives a special final sum command (FSUM). The FSUM command indicates to the core that it should store the final sum, ensuring that the output features are stored using the same distribution as was used for the input features, setting up for the next layer to proceed the same way. Once the core receives the commands, it communicates using a ring pattern across the entire wafer, which is set up using the static routing colors of the on-chip fabric. Using microthreads in the core, all of the reduction is overlapped with the FMAC compute for the next weight row, which starts in parallel. Once all rows of weights are processed, the full MatMul operation is complete, and all of our activations are in place for the next layer.

This mapping of the MatMul onto the wafer enables neural networks of practically all sizes to run with high performance on the single chip. The unique core memory and fabric architecture enable extremely large matrices to be supported without blocking or model parallel partitioning. Even the largest models with up to 100,000 × 100,000 MatMul layers can run without splitting up the matrix. Across all cores of the WSE-2 chip, the result is 75 Pflops of FP16 sparse peak performance[e] (and, potentially, even more with higher sparsity) or 7.5 Pflops of FP16 dense peak performance, all on a single chip. The WSE-2 has the same peak performance for BF16. Now, bringing this back to the grand ML

[e]Sparse performance with 10× sparsity acceleration.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

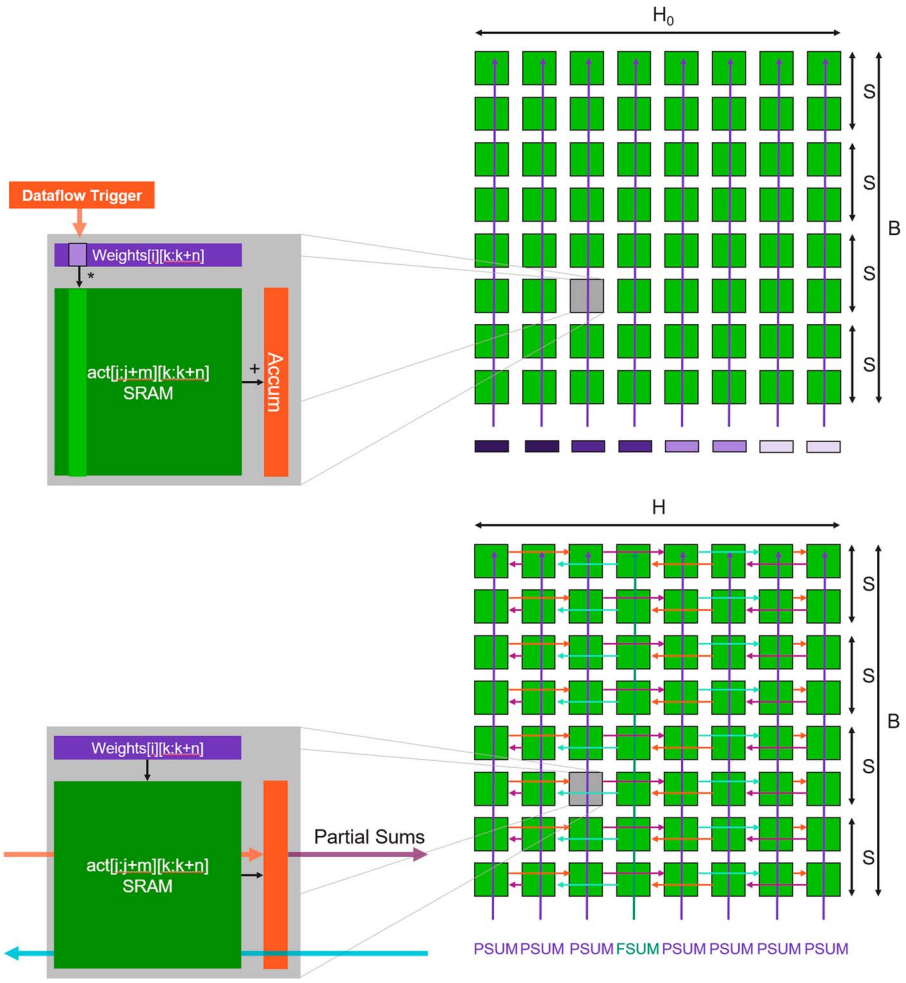


FIGURE 13. Sparse-GEMM computation on the wafer MatMul array. (Top) The weights trigger the AXPY operations using the data flow scheduling mechanism. (Bottom) The partial sums are accumulated in a ring and distributed across the fabric.

demand challenge, leveraging wafer-scale co-design in this way is how we achieve an order-of-magnitude-level improvement in scaling up.

SCALE-OUT: WHY IS IT SO HARD TODAY?

ML clustering solutions already exist today, but it is still so hard to scale out. To understand why, we will look at existing ML scale-out techniques in distributed frameworks, such as DeepSpeed,[7] as shown in Figure 14. The most common is *data parallel*. Data parallel is the simplest approach because it just replicates the model in each device, but it does not work well for large models because the entire model needs to fit in each device. To solve that problem, a common approach is *pipelined model parallel*. This approach splits the model and runs different layers on different devices as a pipeline. However, as the pipeline grows, the activation memory increases quadratically to keep the pipeline full, which is prohibitive for large models. To avoid that, another common approach is to run a form of model parallel by splitting layers across devices, called *tensor model parallel*. This approach has significant communication overhead, and splitting individual layers is complicated. Ultimately, because of all of these constraints, there is no single one-size-fits-all way to scale out today. In fact, in most cases, training large models requires a hybrid approach with all three forms of parallelism.

Therefore, although scale-out solutions exist, they have many limitations. The fundamental reason is because, in traditional scale-out, memory and compute are tied to each other. Trying to run a single model on

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

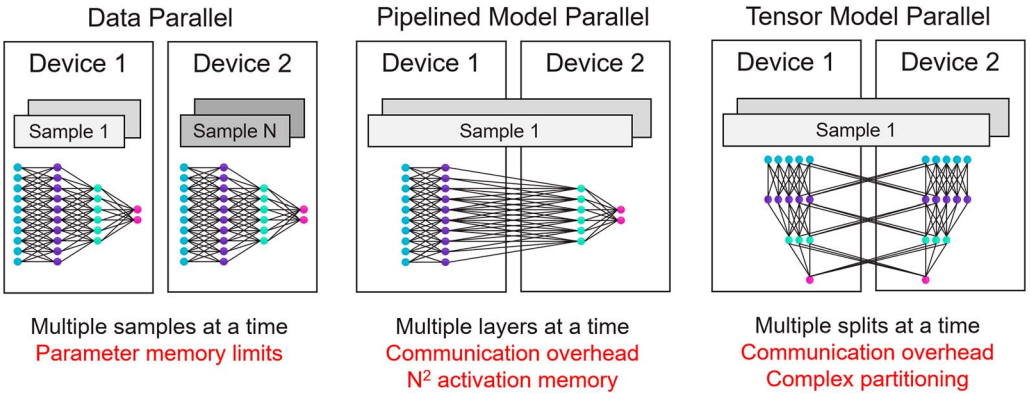


FIGURE 14. Traditional scale-out parallelism techniques and their challenges.

thousands of devices turns the scaling of both memory and compute into distributed constraint problems that are interdependent.

The result of this complexity is shown in Figure 15, which shows the largest models trained on GPUs over the last few years and the different types of parallelism used. We derived the parallelism breakdown from the publications of MegatronV1,[8] T-NLG,[9] GPT-NeoX,[10] GPT-3,[2] PanGu-α,[11] MegatronV2,[12] and MT-NLG.[13] As the models get larger, the more types of parallelism are needed, and this results in a tremendous amount of complexity. For example, the level of tensor model parallelism is always limited to 8× because that is the number of GPUs that are typically in a single server with high internal interconnect bandwidth. As a result, most parallelism for large models is pipelined model parallelism, but that has quadratic memory tradeoffs. Training these models on GPU clusters requires navigating all these bespoke distributed system problems. This complexity results in longer development times and, often, suboptimal scaling.

Data Parallel-Only Scaling

On the other hand, because the Cerebras architecture enables running even the largest models on a single chip without partitioning, scaling simply with *data*

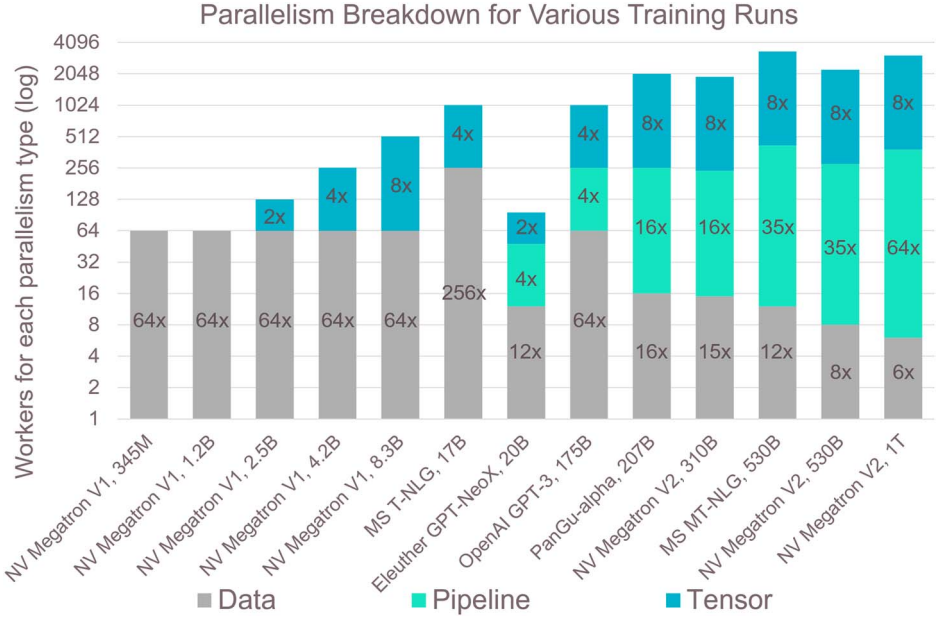


FIGURE 15. Parallelism breakdown for various state-of-the-art models. Large models use all three forms of parallelism: data parallel, pipeline model parallel, and tensor model parallel.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

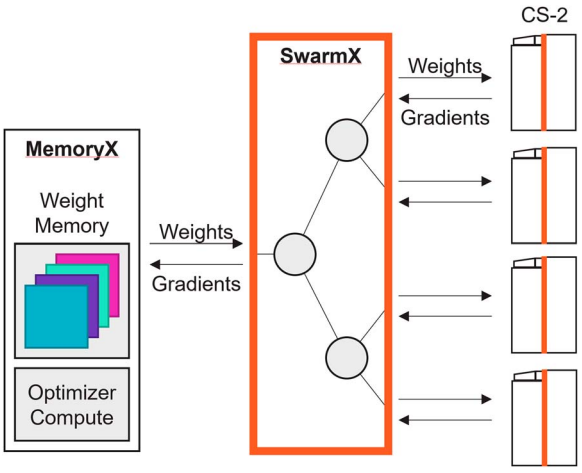


FIGURE 16. SwarmX fabric to scale out with data parallel only. SwarmX broadcasts weights and reduces gradients, enabling the same flow to run on a single CS-2 or multiple CS-2s.

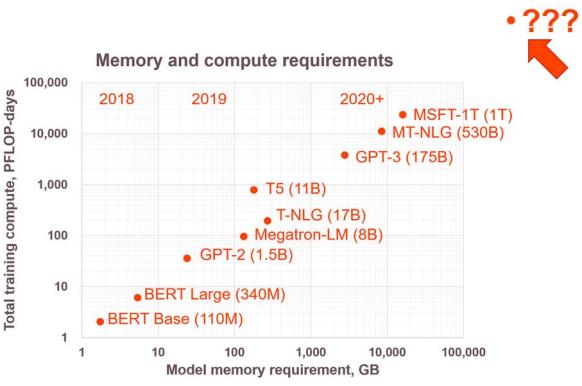


FIGURE 17. Future memory and compute requirements for various state-of-the-art neural networks.

parallel only is possible, without any of the complexity of any form of model parallel partitioning.

Data parallel scaling is done with a specially designed interconnect called SwarmX, as shown in Figure 16. SwarmX sits between the MemoryX units that hold the weights and the CS-2 systems for compute, but it is completely independent from both. SwarmX broadcasts weights to all CS-2 systems, and it reduces gradients from all CS-2s, making it an active component in the training process, purpose-built for data parallel scale-out. Internally, SwarmX uses a tree topology to enable modular and low-overhead scaling. Because it is modular and disaggregated, it can scale to any number of CS-2 systems with the same execution model as a single system. Scaling to more compute is as simple as adding more nodes to the SwarmX topology and adding more CS-2 systems. Using this scalable cluster design is how the Cerebras architecture addresses the last component of the grand ML demand challenge to improve and drastically simplify scale-out.

CONCLUSION

In the past couple of years, we have seen demand of more than three orders of magnitude greater from ML workloads. There is no sign of slowing down, so the next couple of years could see the ML demand increasing by another several orders of magnitude, as shown in Figure 17. The Cerebras architecture is designed from the ground up to address this challenge.

The architecture principal is to break from traditional techniques and improve by an order of magnitude in three key areas: 1) by improving the core architecture by an order of magnitude with unstructured sparsity acceleration, 2) by scaling up by an order of magnitude with wafer-scale chips, 3) by improving cluster scale-out by an order of magnitude with truly scalable clustering. With all of these techniques together, we believe this future is achievable.

Neural network models are continuing to grow exponentially. Few companies today have access to them, and that list is only getting smaller. With the Cerebras architecture, by enabling the largest models to run on a single device, data parallel-only scale-out, and native unstructured sparsity acceleration, our goal is to make these large models available to everyone.

REFERENCES

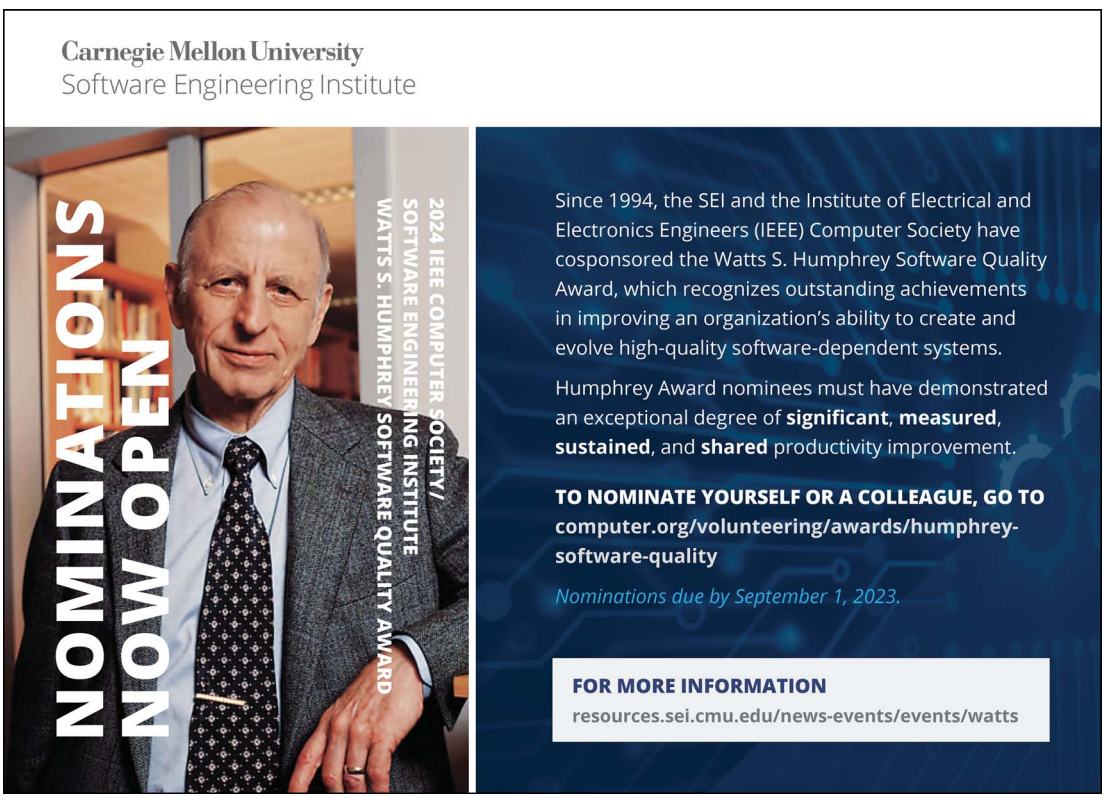
1. J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
2. T. Brown et al., "Language models are few-shot learners," 2020, *arXiv:2005.14165*.
3. *4th Gen AMD EPYCTM Processor Architecture*. (2022). Advanced Micro Devices. [Online]. Available: https://www.amd.com/system/files/documents/4th-gen-epyc-processor-architecture-white-paper.pdf
4. "NVIDIA H100 tensor core GPU architecture." Nvidia. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper
5. "High bandwidth memory (HBM3) DRAM," JEDEC, Arlington, VA, USA, JESD238, 2022. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd238
6. G. E. Moore, "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.," *IEEE*

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.

Solid-State Circuits Soc. Newslett., vol. 11, no. 3, pp. 33–35, Sep. 2006, doi: 10.1109/N-SSC.2006.4785860.

7. R. Majumder and J. Wang. *DeepSpeed: Extreme-Scale Model Training* for *Everyone*. (2020). Microsoft. [Online]. Available: https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/
8. M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv:1909.08053*.
9. C. Rosset. *Turing-NLG: A 17-Billion-Parameter Language Model by Microsoft*. (2020). Microsoft. [Online]. Available: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/
10. S. Black et al., "GPT-NeoX-20B: An open-source autoregressive language model," 2022, *arXiv:2204.06745*.
11. W. Zeng et al., "PanGu-α: Large-scale autoregressive pretrained Chinese language models with auto-parallel computation," 2021, *arXiv:2104.12369*.
12. D. Narayanan et al. "Scaling language model training to a trillion parameters using Megatron." Nvidia. Accessed: 2022. [Online]. Available: https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/
13. S. Smith et al., "Using DeepSpeed and Megatron to train Megatron-Turing NLG 530B, a large-scale generative language model," 2022, *arXiv:2201.11990*.

SEAN LIE is the chief hardware architect and cofounder of Cerebras Systems, Sunnyvale, CA, 94085, USA. His research interests include hardware/software co-design and machine learning, transactional memory, high-performance CPUs, networking, storage, and large-scale distributed clusters. Lie received his B.S. and M.Eng. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology. Contact him at sean@cerebras.net.



Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on March 22,2025 at 01:21:12 UTC from IEEE Xplore. Restrictions apply.