GSI TECHNOLOGY

Compute-in-Memory Computational Devices

www.gsitechnology.com

Contents

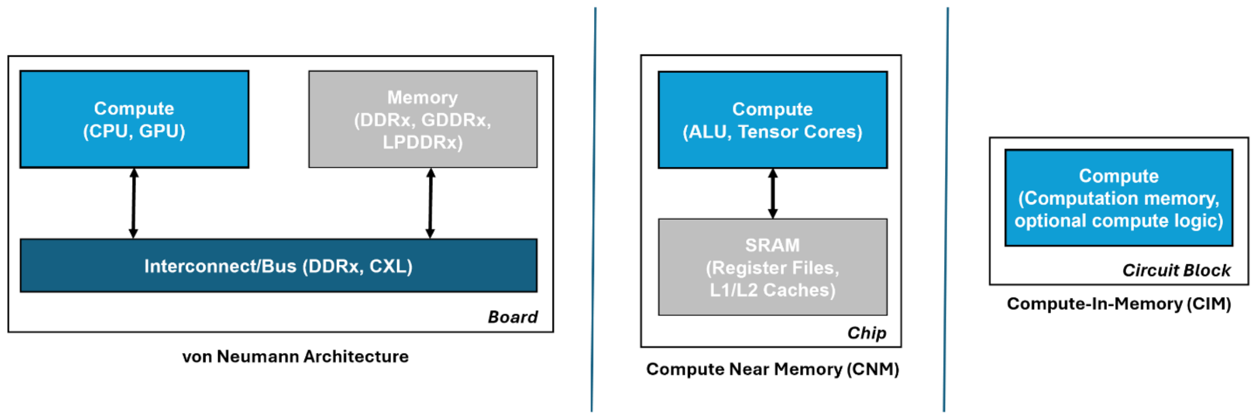
Compute-In-Memory (CIM) Definition

Compute-in-memory (CIM) conceptually means doing compute directly **in** the memory. As will be discussed in detail in subsequent sections, the memory itself is capable of performing efficient computing, which incurs much less data transfers resulting in higher performance and much lower power. These characteristics make it very beneficial for AI and high-performance computing workloads. Sometimes In-memory compute (IMC) is used for this definition also, but as will be explained in detail care must be observed to distinguish some of the uses of this term with a more appropriate definition of compute-near-memory which has different characteristics.

In contrast, near-memory computing architectures have the compute and memory blocks, even within the same chip or die, physically separate as standalone circuit blocks. CPUs, GPUs, or in general any modern AI ASICs (e.g., TPU, Inferentia, etc.) integrate various heterogeneous blocks and larger memory blocks—commonly referred to as caches and register files. These modern processors, SoCs, and AI accelerators are good examples of near-memory computing architectures. These follow what’s known as a von Neumann architecture, which requires the movement of data and instructions from memory to these heterogeneous computational units via direction from a sequencer and pipelining mechanism. The minimum block that computations can be measured from are a computation unit and the local register memory that provides the inputs for it to operate on.

The traditional von Neumann architecture extends to the near memory concept at a system level, with separate compute and memory chips in a system. In traditional von Neumann, including near-memory architectures, data movement between compute and memory at board level or chip level becomes costly in terms of time and energy. In recent data-centric applications, such as Artificial Intelligence/Machine Learning (AI/ML), this problem is aggravated with large amounts of data movement.

Figure 1 below is a side-by-side conceptual illustration of the three architecture types. The blocks show the minimum circuit blocks required for the different architectures to compute something. A primary difference that becomes visible is that CIM structures do not need additional memory to perform their computations. The basic structure can perform as memory and compute.



Feature	Compute Near Memory (CNM)	Compute-In-Memory (CIM)
Computation directly on memory bit lines	No	Yes
Distance between additional processing logic and memory	~100-1000 microns	~10 microns
Data movement overhead	Lower than traditional architectures, but higher than CIM	Lowest
Technology maturity	More mature, can be implemented with existing technologies	New commercial architectures
Performance improvement potential	Lower than CIM, but still significant	Highest

Figure 1: Computing Architectural Types

Example Compute in Memory Architecture

To solve the computation within memory challenge that brings CIM the power and performance benefits, we must look at memory and build on its capabilities. Two structures provide a base to build from. First is a structure called content-addressable-memory or CAM. The second deals with loosening some operational rules that is inherent to traditional memory.

CAM is unique but a simple twist on standard memory. In standard memory a specific location is addressed for writing and then reading data. In CAM the read operation is not performed as with traditional memory. Data is presented to CAM, instead of an address, which the device will use to simultaneously compare all locations it has to that data and provide the result address if the data is stored in the part. This may sound weird, but if you think about it, that is similar to how our own brain compares things to memories: have I seen this before and if so, what is it, when did that occur, what should I remember about it. The CAM provides the first step of this flow. A specialized version of CAM adds a “don’t care” capability. That is, data that is sent for comparison can have certain portions or bits defined as “don’t care” if they match. This specialized operation again is similar to our brain operation where we may filter looking for our car in a parking lot and not caring about any vehicle that is not our make, model and color. This type of memory is called a ternary content addressable memory or TCAM. TCAM can be very useful for processing, but by itself it is just a memory with massive lookup capability. Providing the ability to loop back CAM data results for iterative searches or other processing starts to add discerning capability to this function. See **Figure 2**.

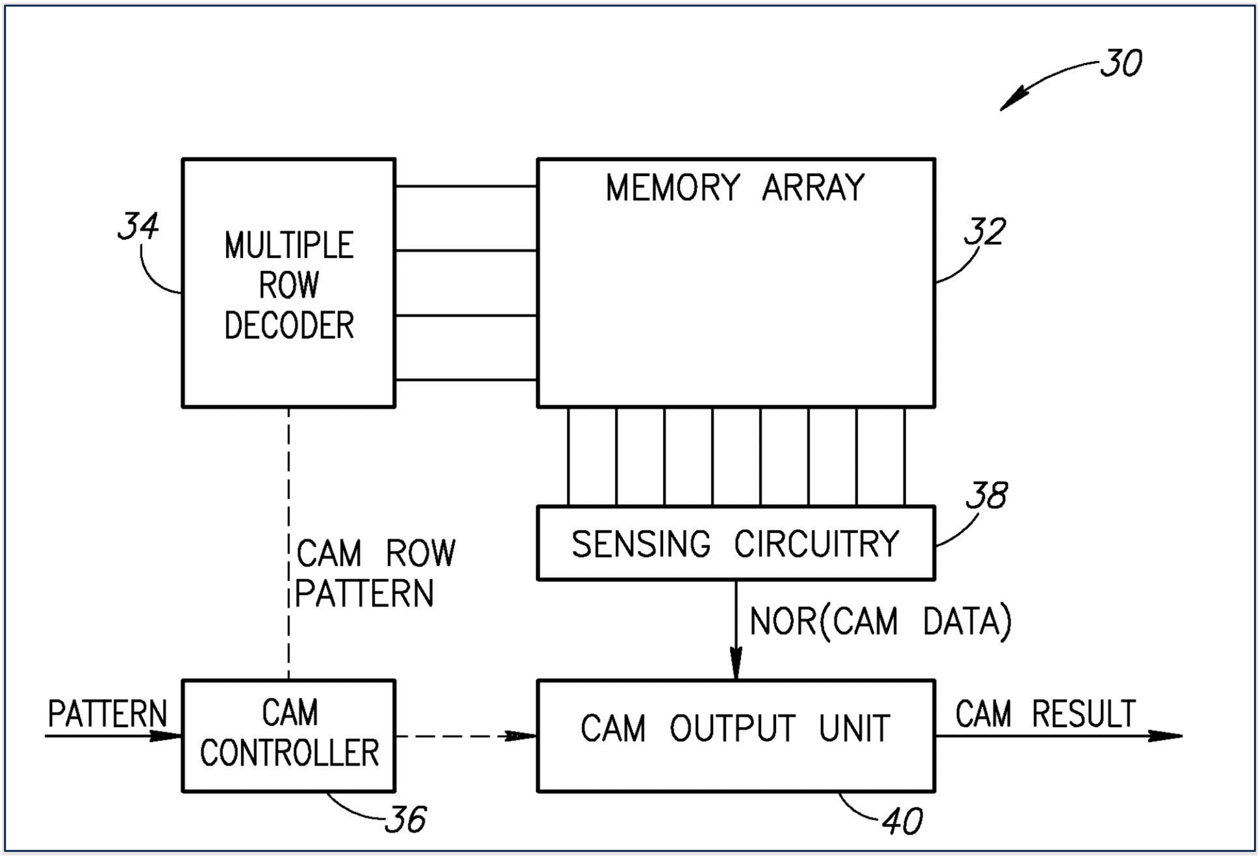


Figure 2: CAM Pattern Processing within a Memory Construct

From Patent US 8238173B2

Traditional memory, not CAM, has an array of memory and a unique address for each location. Memories are organized into rows and columns that is a decoding of the address and operate by reading charge from a particular bit at that intersection at a time. These are organized such that a particular memory width (multiple bits) is supported for a particular memory. This is often 8 bits, 16 bits, 32 bits, or 64 bits. Let's loosely define the width that the memory is configured as a "word"[1]. Only one bit is read per bit position in a word address. Only the specific result that is at the intersection of the row and column is read and latched into a result register for each bit location in the word. This is a standard rule in memory.

In order to read the charge from a memory during the read process, memory lines holding bits are pre-charged or have other mechanisms to maintain charge. Instead of reading only one bit from a memory row as is done in a standard memory, if we were to allow multiple bits on a bit line to be triggered an interesting function occurs. Normally, this is prohibited as the state of several bits would collide and we wouldn't read what we wanted. But if we understood what could happen and controlled it, then we could get logic functions to occur from memory reads like what is shown in **Figure 3**. In that figure we see that if multiple bits can be enabled at the same time onto the same bit line, then just the operation of reading the data incurs a NOR gate compute function. Since memory reads occur at the fastest time in a processor such a construct could be created to create a very fast Boolean logic processor *with just the memory lines*.

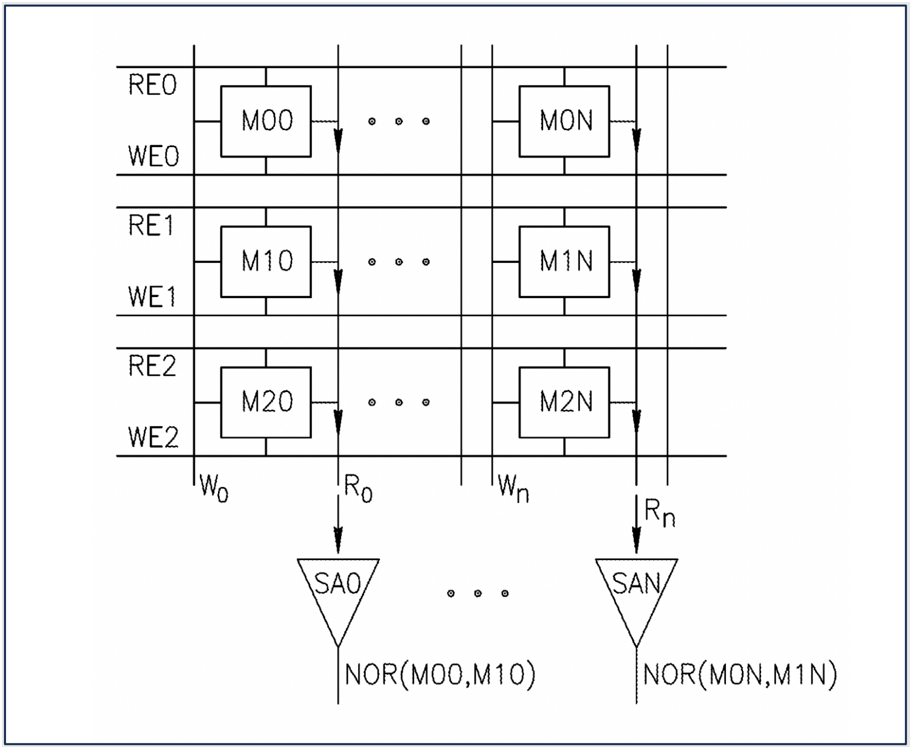


Figure 3: Using Storage Cells to Perform Computation: Boolean

From Patent US 8238173B2

Alternative Architectures

Near-Memory Compute Architectures

Since compute blocks or Arithmetic Logic Units (ALUs) are well known and implement compute structures the use of these with memory have been the traditional method of creating processors. Putting memory very close to these or in-between stages has been one way of improving local performance.

GPUs and high performance CPUs with cache close to the processing elements are the best examples of near-memory compute architectures. Compute near memory can also be implemented as separate ALU structures with tightly coupled memory. In these structures the tensor core (in the GPU), or multiply/accumulate (MAC) or other ALU functions in special purpose parts are not inherent functions that occur in a special memory construct. The block diagram of the latest Nvidia GPU H100 is shown in **Figure 4** below to provide an illustration.

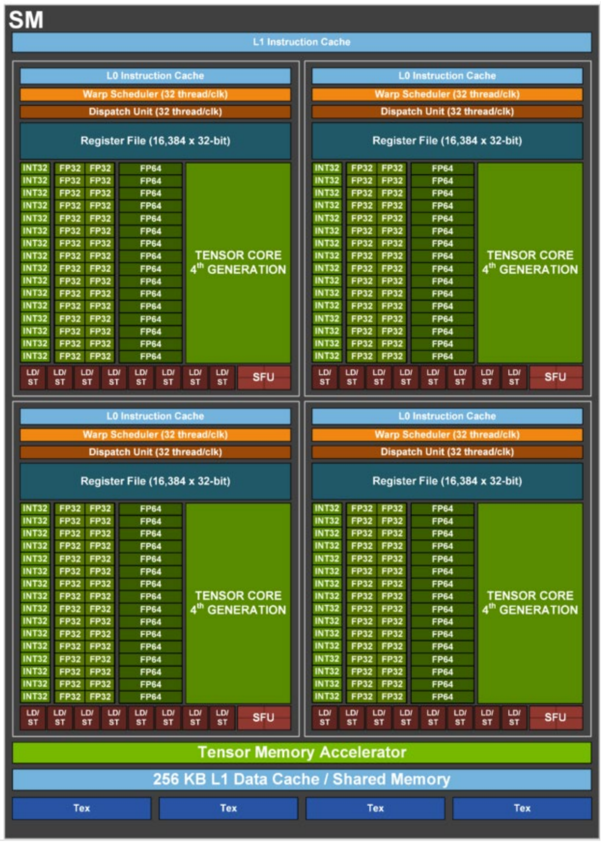

SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE 4th GENERATION
LD/ST
SFU
Tensor Memory Accelerator
256 KB L1 Data Cache / Shared Memory
Tex

Figure 4: Nvidia H100 GPU Block diagram

Three key points to note:

1) There are multiple types of cores surrounded by memory. Nvidia continues to evolve their cores to support different data formats independently from the memory. For example, the H100 adds FP8/Int8 precision to the Tensor cores. The register file sizes can and do vary separately from the cores from part to part.

2) The register files, L1 cache and shared caches are significantly farther from the tensor compute cores, which would result in higher energy spent in data transfers between compute and memory.

3) As we get farther from the compute cores, more levels of cache are required to keep the cores busy with larger cache sizes but lower bandwidth as we get farther away.

As one can see from the wide list of features and data format support, a general-purpose GPU or CPU may not deliver the best performance/watt/$ and overall TCO efficiency because not all resources would be used and memory transfers has to be routed to where processing is occurring.

Another example of a near-memory AI accelerator is Google's TPU ASIC. The block diagram of TPU v1 (**Figure 5**) clearly illustrates the near-memory architecture where the compute and memory blocks are physically separate. The concept is extended to subsequent generations where external HBM memory is used for storing data with scaler and vector units having limited internal SRAM memory. See **Figure 6**.

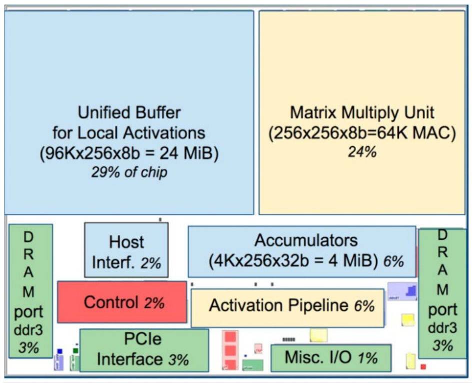

Unified Buffer for Local Activations (96Kx256x8b = 24 MiB) 29% of chip
Matrix Multiply Unit (256x256x8b=64K MAC) 24%
D R A M port ddr3 3%
Host Interf. 2%
Accumulators (4Kx256x32b = 4 MiB) 6%
D R A M port ddr3 3%
Control 2%
Activation Pipeline 6%
PCIe Interface 3%
Misc. I/O 1%

Figure 5: TPU v1 Block Diagram

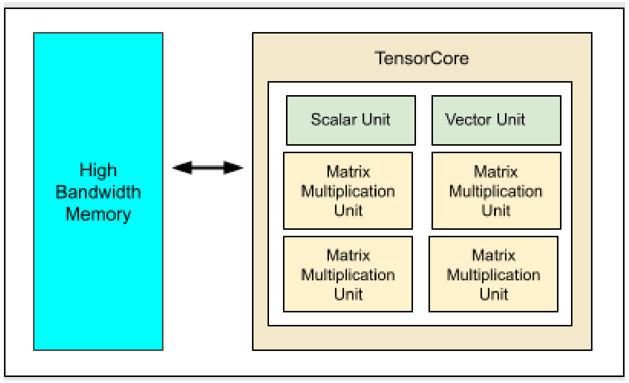

High Bandwidth Memory
TensorCore
Scalar Unit
Vector Unit
Matrix Multiplication Unit
Matrix Multiplication Unit
Matrix Multiplication Unit
Matrix Multiplication Unit

Figure 6: TPU v5e Block Diagram

Similarly, many other AI accelerator ASICs developed by startups are mainly near-memory as illustrated in the generic block diagram of **Figure 7** below. The ALUs could be multipliers, multiplier/accumulators (MACs), digital signal processing units (DSP), other general mathematical arithmetic logic units, or vector processing units.

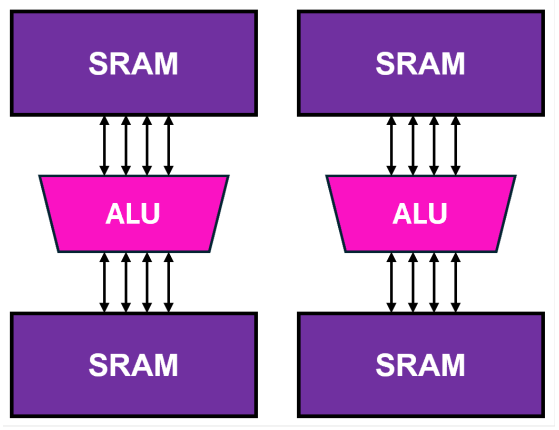


Figure 7: AI Accelerator ASIC Block Diagram

Referring back to **Figure 1**, CNM structures are different from CIM in that the structure that does the compute is not a memory in itself. It requires memory to operate, and this memory is positioned closely. This memory is operating as register memory for the actual compute engine, so a CNM structure is a more optimized von Neumann structure with memory close by. The power savings in these structures diminishes as you scale up to more parts or connect to larger memory storage for bigger workloads. It is for this reason that GPUs and multi-core CPUs with large caches for each core and ALU end up consuming large amounts of data when scaled to large arrays. At the high system level, the benefits of the local cache are all but lost when there are so many transfers required to get the data to the actual compute structures.

PIM (Processing-In-Memory)

The Processing-in-Memory or PIM concept does attempt to reduce energy associated with data transfers between compute and memory, but in current form is merely a near-memory architecture and not true compute-in-memory, because computing is not performed on bit lines of the memory array. Instead, arithmetic logic units or ALUs are embedded close to the memory arrays as we have defined for CNM architectures. At the component level, the difference generally between CNM processors and PIM comes down to the device function—primarily being a processor (CNM) or memory (PIM).

With current implementation, the computation is limited to fixed data formats, such as FP16 multiply and add. As data formats continue to evolve, as highlighted with the formation of the recent Microscaling Formats (MX) Alliance, there is a need to support flexibility in supporting data formats with lower or mixed precision. Efficient software implementations could also be delayed due to changes required in the hardware to address new frameworks when using parts with pre-defined data formats.

Samsung's PIM integrates limited and non-programmable compute capability in their HBM2 stack, which could limit flexibility and applicability in future.

Analog CIM (Compute-In-Memory)

Analog compute-in-memory leverages designed memory elements by using tunable resistors, such as Memristors, to perform computations directly inside the memory array itself.

This architecture can be called true compute-in-memory (CIM) and can deliver computational efficiency, but there are many technical challenges with this approach as described below.

Analog CIM has been explored by many research labs for a long time, and startups have started creating test parts for the direction, but due to challenges described below, this approach has hurdles to becoming mainstream in near future. See **Figure 8**.

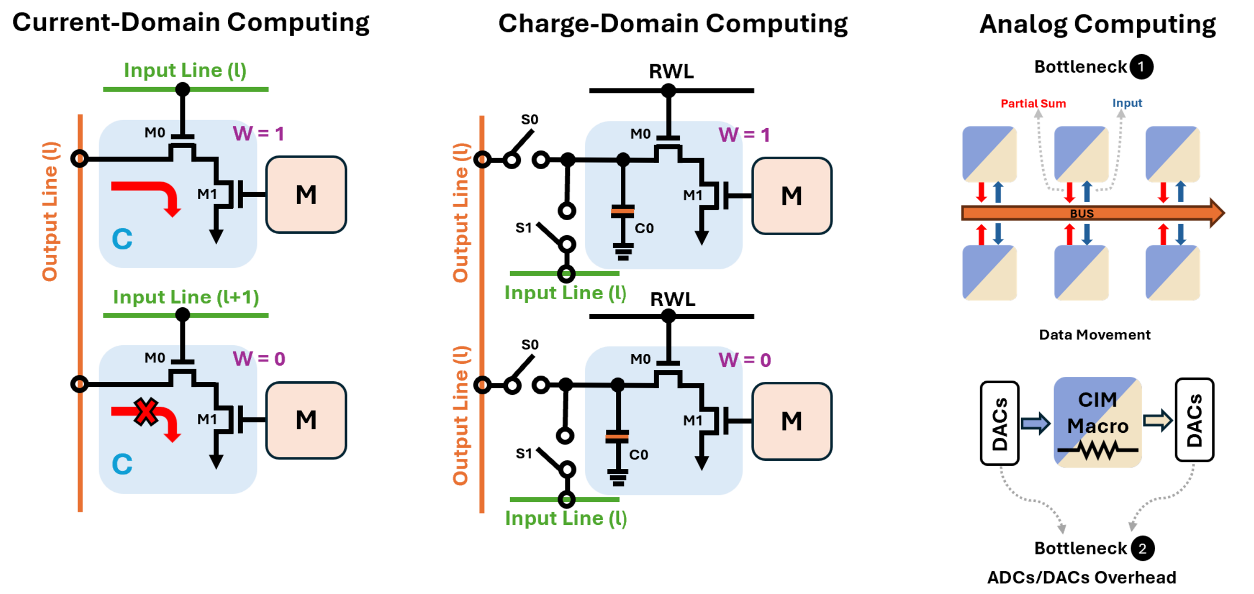


Figure 8: Computing Architectures

The limitations of analog compute-in-memory architecture include the following:

Compute precision: There are concerns about whether this approach can provide sufficient compute precision to ensure that application accuracies are not compromised. Increasing precision can be achieved by increasing the dynamic range of analog components, but due to noise limitations, this can lead to higher voltage requirements. Analog resistive memory devices also suffer from device-to-device and cycle-to-cycle variances that can limit absolute precision (see Ref. 3, Le Gallo 2023).

Integration with digital compute: The seamless integration of analog compute with digital compute is still a question, particularly in the context of achieving full end-to-end realization of deep neural networks on-chip. The ADC/DACs used for analog/digital conversion are susceptible to noise that limits the performance too.

Latency and cost challenges: The architecture introduces challenges related to overall latency of computation. This occurs in charge and current domain architectures because of integration steps for results. For analog resistive memory devices, it results from the analog conversion steps. Cost too is a challenge, since most of the companies pioneering analog compute are small startups and the technology hasn't been deployed in high volume to gain associated cost advantages.

Design complexity: While analog compute-in-memory architecture offers some power advantages, its design and implementation can be more complex compared to traditional digital computing architectures.

In summary, while analog compute-in-memory architecture offers a few advantages, such as energy efficiency, it also presents challenges related to precision, integration, and design complexity that need to be carefully addressed before it is ready for high volume deployment in Data Centers or Edge.

Technology Challenges and Market Needs

Two key technology challenges are challenging the supremacy of the von Neumann CPU and by extension, the GPU architecture.

General Purpose GPUs Are Not Keeping Up with Compute Demand

As ChatGPT demonstrated the fastest adoption of any technology to-date, the underlying generative AI technology and large-language models (LLMs) have tremendous promise of increasing productivity of many applications. This has led to an exponential rise in compute demand that is currently met with general purpose GPUs, primarily from Nvidia and AMD. Unfortunately, traditional GPU architectures are not suitable for these emerging AI workloads, which need compute to scale 750 times or more every 2 years, as shown by the red line in chart below (see **Figure 9**). Furthermore, these general-purpose GPUs traditionally relied on Moore's law scaling that delivered 2x transistor packing density and roughly the same 2x improvement in performance every 2 years, as illustrated by the gray line in the image below.

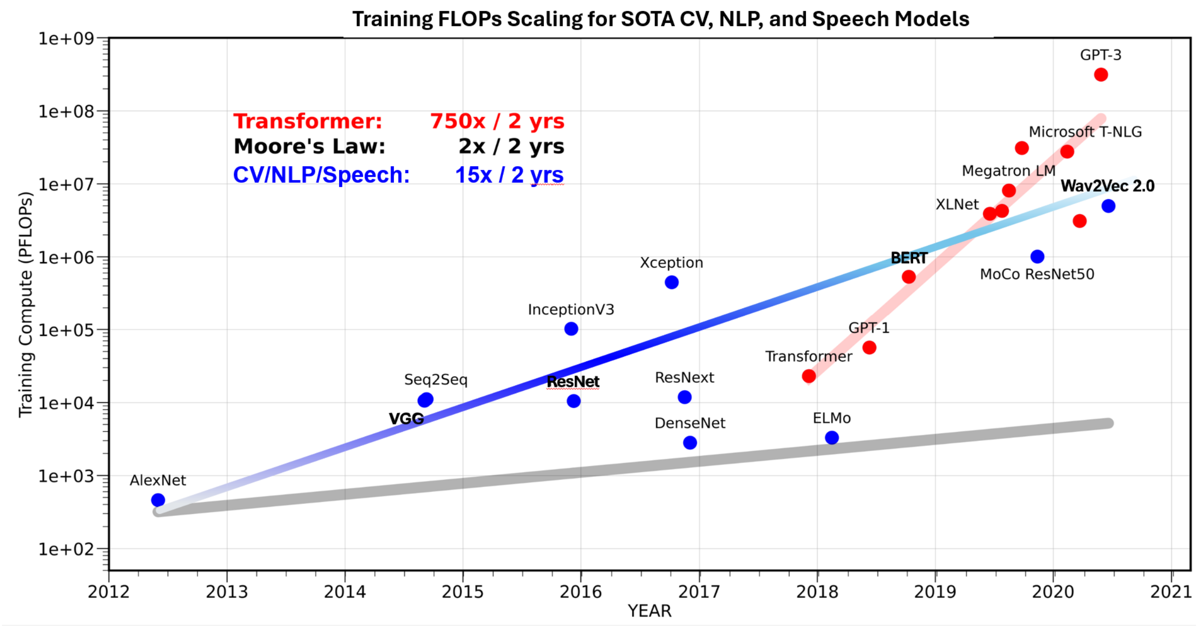


Figure 9: Training FLOPs Scaling

Source: https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8

Memory and IO Bandwidth Are Not Keeping Up with AI Workload Demand

Besides falling short of meeting the compute demands ~3x/2 years vs 750x/2 years required for training Transformer network, the external memory and IO are not keeping up with the generative AI workload demand either as illustrated by green and blue trajectories respectively (see **Figure 10**).

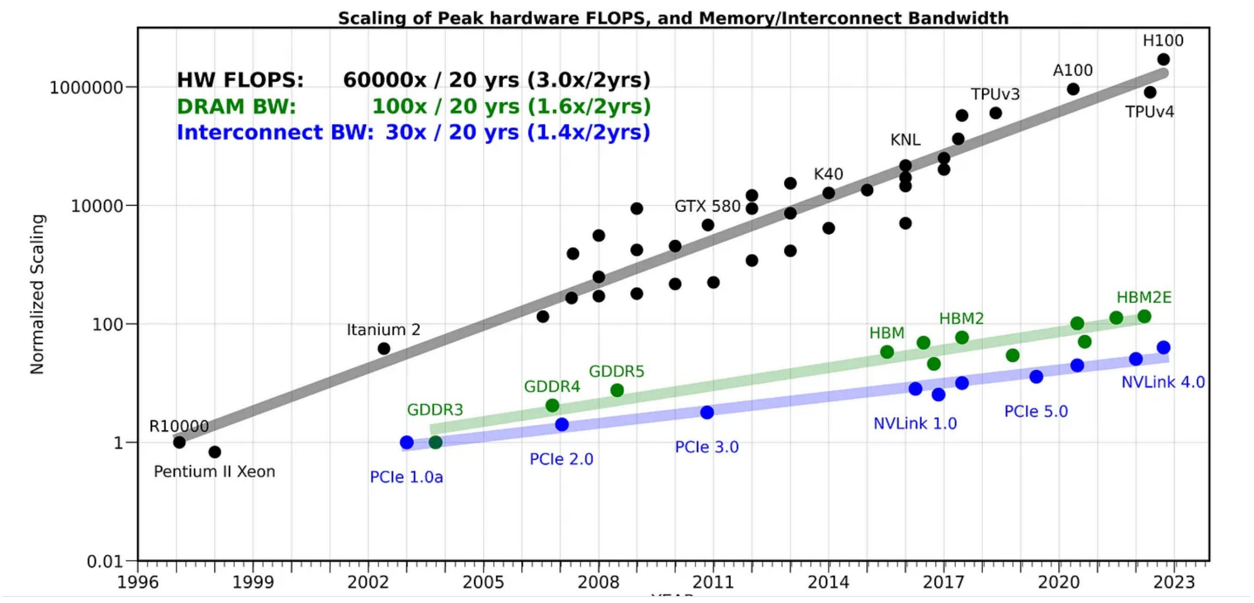


Figure 10: Scaling of Peak Hardware FLOPs

Source: https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8

Advantages and Business Value

Compute-in-memory represents a paradigm shift in computer architecture, offering a solution to the diminishing returns traditional processors have been facing of controlling power while increasing performance.

1. **High power efficiency and low power consumption:**

CIM architectures can compute on data that is in memory. Specifically, data computations within the storage elements can have compute operations applied in-place and looped if required for iterative processing to build up complex functions with reduced instructions and minimal data transfers to storage memory. Data transfer power associated with CIM architectures can also be very low compared to traditional von Neumann architectures by keeping additional memory very close. The combination of these characteristics can offer orders of magnitude less power than the comparable distances in traditional von Neumann architectures, which rely on data transfers from register files and various levels of cache memory to the processing elements for all compute operations.

2. **Sustainable Application Performance:**

Because compute-in-memory devices are inherently storage, immediate shaped inputs from sensors can be placed in them for processing. With appropriate register buffer memory and I/O port architecture devices can be achieved that offer "through-wire" processing or "bump-in-the-wire" processing for real-time data. This provides a paradigm change in low power and high throughput which can be taken advantage of for AI and HPC processing for real-time edge applications. These architectures are capable of eliminating application memory bottlenecks for models that can be housed in its array. This leads to 100% core utilization.

3. **Scalability:**

Systems utilizing compute-in-memory components can potentially scale as you would add memory to a system: by merely adding more components and without the need for complex connectivity. Using this methodology, it becomes quite normal to have a single low-cost server system with many cards in place. Massive datacenter magnitude scaling now becomes possible with commodity 10gE server interconnects.

Conclusion

Compute-in-memory represents a significant shift in computing paradigm with proven SRAM technology that is ready for high volume deployment. As the cost of compute and memory—in terms of energy, cooling, and diminishing returns on scale—continues to increase and the volume of data generated by businesses grows, compute-in-memory is becoming an increasingly vital component of modern IT infrastructure, enabling applications and actions that were previously economically unattainable. This whitepaper serves to clearly describe a true digital compute-in-memory architecture and contrast it with alternative architectures, particularly near-memory and PIM. We would like to leave you with a simple checklist or test to determine whether the architecture is true compute-in-memory. If the architecture that is evaluated does not meet the criteria below, they are not compute-in-memory.

CIM—Simple Checklist/Test

There are myriad implementations in industry referred to as compute-in-memory. Here is a simple list of checks that should define true compute-in-memory architecture.

1. Computation occurs where the data resides. For example, external sensor data could be stored in the compute structure, rather than storage memory. This data can then be computed in place, without moving from storage memory. This data transfer savings is significant to performance and low power performance of CIM.

2. Local or Register memory is not required for the compute operation but may be available to quickly transfer in new data or switch tasks.

3. Compute logic, if present, is embedded within the memory lines themselves, typically within a few microns. This characteristic is the most open to re-interpretation, as ALUs or computation blocks may be embedded with memory. Such structures will be greater than 100 microns from memory and will not conform to rule 1 above.

4. Offers the lowest data movement overhead (pJ/bit) and the best TCO measuring performance and power.

References

1. “Benchmarking and Modeling of Analog and Digital SRAM In-Memory Computing Architectures”. Houshmand, Pouya. Sun, Jiacong. Verchelst, Marian. arXiv:2305.18335[cs.AR], 2023

2. “AiDAC: A Low-Cost In-Memory Computing Architecture with All-Analog Multi-Bit Compute and Interconnect”. Xuan, Zihao. Chen, Song. Kang, Yi. arXiv:2312.11836[cs.AR], 2023.

3. “Using the IBM Analog In-Memory Hardware Acceleration Kit for Neural Network Training and Inference”. Le Gallo, Michael. Lammie, Corey. Buchel, Julien. Carta, Fabio. Fagbohungbe, Omobayode. Mackin, Charles. Tsai, Hsinyu. Narayanan, Vijay. Sebastian, Abu. El Maghraoui, Kaoutar. Rasch, Malte. APL Machine Learning. Volume 1, Issue 4. AIP Publishing, 2023.

4. Akerib, Avidan. Agam, Oren. Ehrman, Eli. Meyassed, Moshe. 2012. Using Storage Cells to Perform Computation (U.S. Patent No. 8238173B2). U.S. Patent and Trademark Office. https://patents.google.com/patent/US8238173B2/en

Endnote

1. **Word—**While not technically precise, this term is used in this paper to reflect different widths that may be inherently stored in a memory. In computer architecture, various words, bytes, words, long words, etc. are used to reflect the different widths possible. For simplicity these are all attributed to the use of “word” in this description.

For more information, please contact **associativecomputing@gsitechnology.com**.