HPC Software Scaling for ML using CXL 3.0 GFAM

October 12, 2023

Patrick Estep

SC23
Denver, CO | i am hpc.

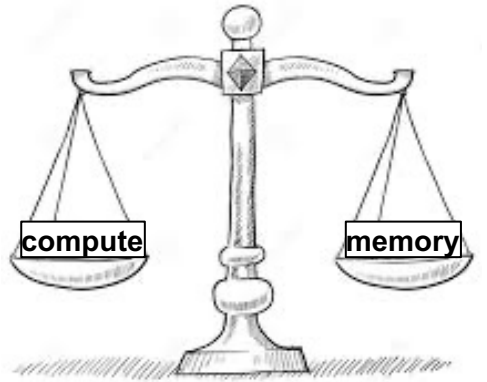
Micron

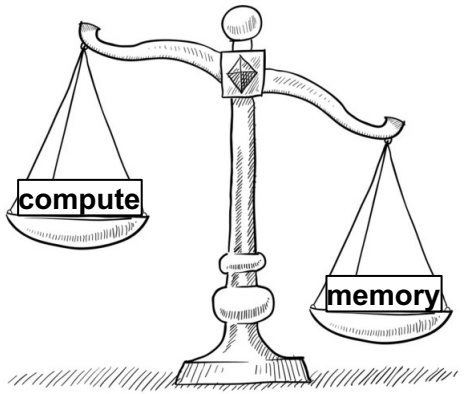
Content

Machine Learning's Affect on Scaling

LLMs break balanced soft scaling

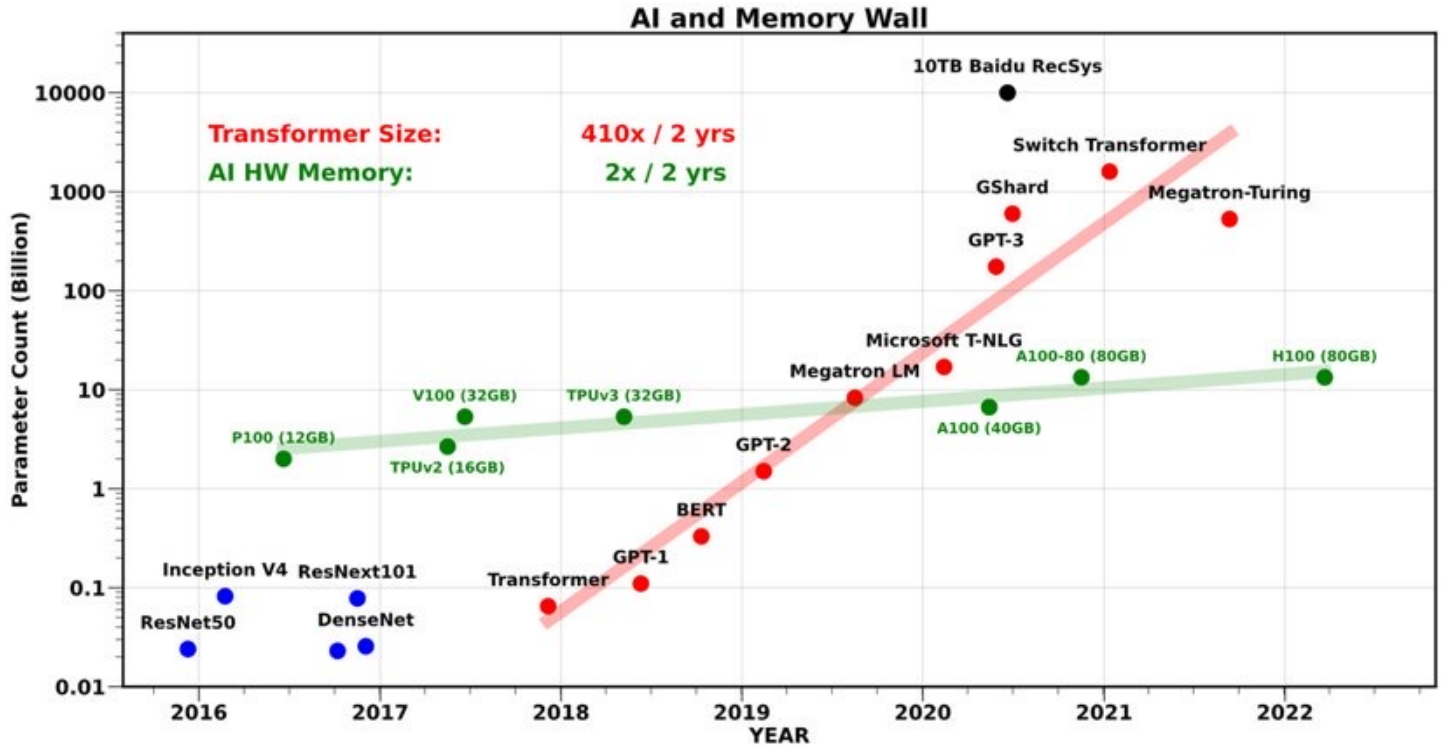
- Traditional High-Performance Computing (HPC) systems use a balanced soft scaling approach, adjusting the compute resources to memory ratio based on the workload.
- This ensures efficient resource use for most applications.
- This approach struggles with Machine Learning (ML) applications, especially Large Language Model (LLM) workloads, which demand more memory than compute resources.
- This imbalance can lead to underutilized compute resources and excessive data movement, affecting performance and energy efficiency.





AI has hit a memory wall

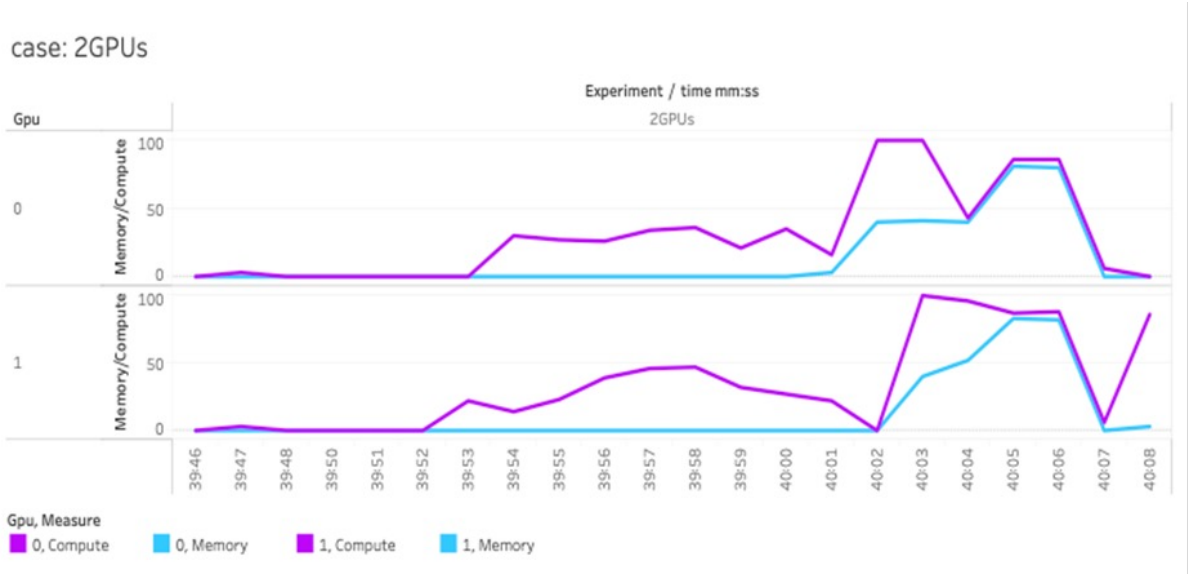
- Compute needed to train Transformer models has been growing at a rate of 750x/2yrs.
- There is an emerging challenge with training and serving these models: memory and communication bottlenecks.
  - AI applications are becoming bottlenecked by intra/inter-chip and communication across/to AI accelerators rather than compute.
- LLM model sizes has been increasing at a rate of 410x every 2 years.
- Large Recommendation System models have reached O(10) TB parameters. DRAM memory has only scaled at a rate of 2x every 2 years.

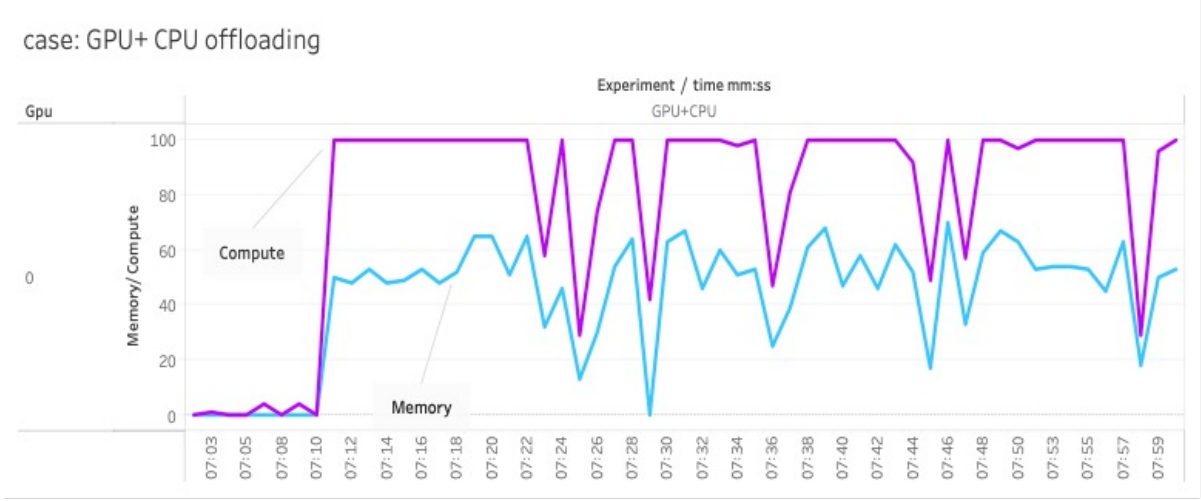


*Source: Gholami A, Yao Z, Kim S, Mahoney MW, Keutzer K. AI and Memory Wall. RiseLab Medium Blog Post, University of Califonia Berkeley, 2021, March 29.

Stranded compute resources

- In this LLM example, the model is so large that it cannot be efficiently run on a 2 GPU configuration, resulting in underutilization of GPU compute resources.
- CPU offload (e.g., Microsoft's ZeRO (Zero Redundancy Optimizer)) is a technology that enhances the efficiency of distributed training for deep-learning models. It achieves this by offloading some data from the GPU to the CPU, enabling larger model sizes per GPU to be trained.
- The results with GPU + CPU offloading show complete utilization of the GPU compute power.





The Promise of CXL 3.0 GFAM

Where can GFAM help?*

The focus is on shared memory for this presentation

AI/ML Acceleration

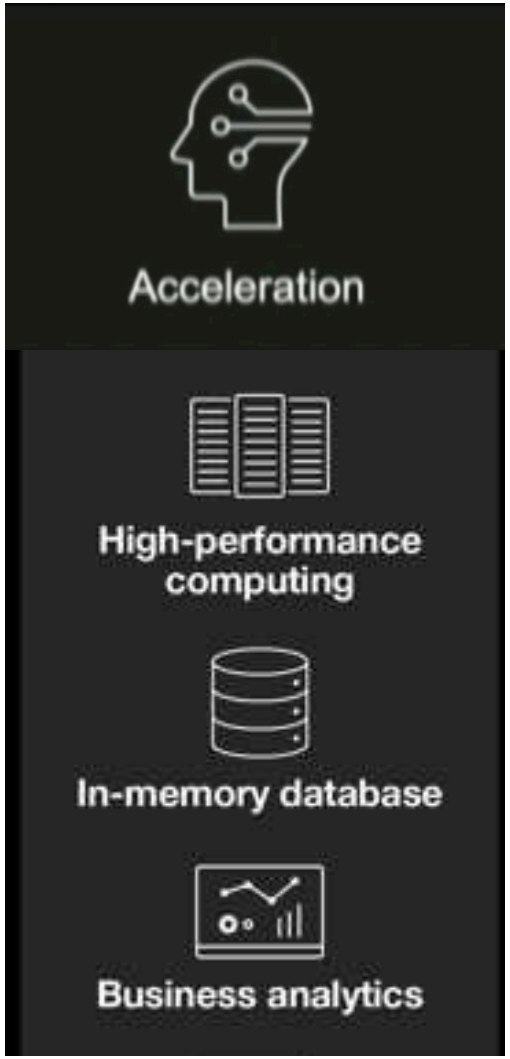
- CXL facilitates quicker and more effective connections between the CPU and devices, as well as the CPU and memory, for AI/ML accelerators like GPUs, ASICs, or FPGAs. CXL allows **a**pplications to process increasingly larger datasets and minimizes the amount of data transferred between hosts, reducing the time required to obtain results.

HPC

- CXL enhances HPC performance, scalability, and adaptability by leveraging CXL-enabled accelerators and shared memory. Multiple compute nodes can directly access data in memory, eliminating the need for local duplication before and after operations.

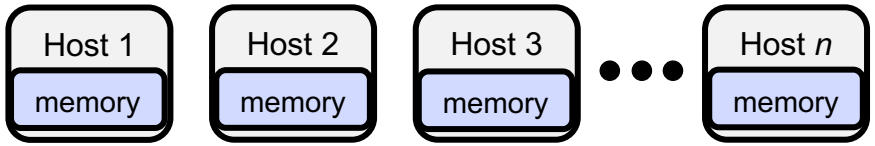
Large-scale in-memory, analytical, and graph databases

- CXL enable systems to handle significantly larger data sets by providing databases with access to vast amounts of memory that offers low latency and high bandwidth.



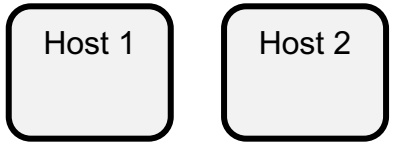
Addressing AI/ML scalability challenges with GFAM

- Imbalanced memory and compute requirements lead to underutilized compute resources and excessive data movement, affecting performance and energy efficiency.
  - There is more data to process, even though power budgets are flat or shrinking.
- The proposed solution to the challenges faced by High-Performance Computing (HPC) systems is the use of GFAM.
- GFAM allows independent scaling of compute and memory resources, leading to efficient resource utilization for memory-intensive workloads like LLMs.
- GFAM also minimizes the need for data movement.
  - Prepared data can be left in GFAM in zero-copy formats (e.g., NumPy Ndarray, pandas DataFrames, etc.)
- However, adopting GFAM requires strategic **placement of memory and compute resources** and potential **software stack adjustments**.



Aggregate memory footprint

Large # of compute resources to achieve required total memory



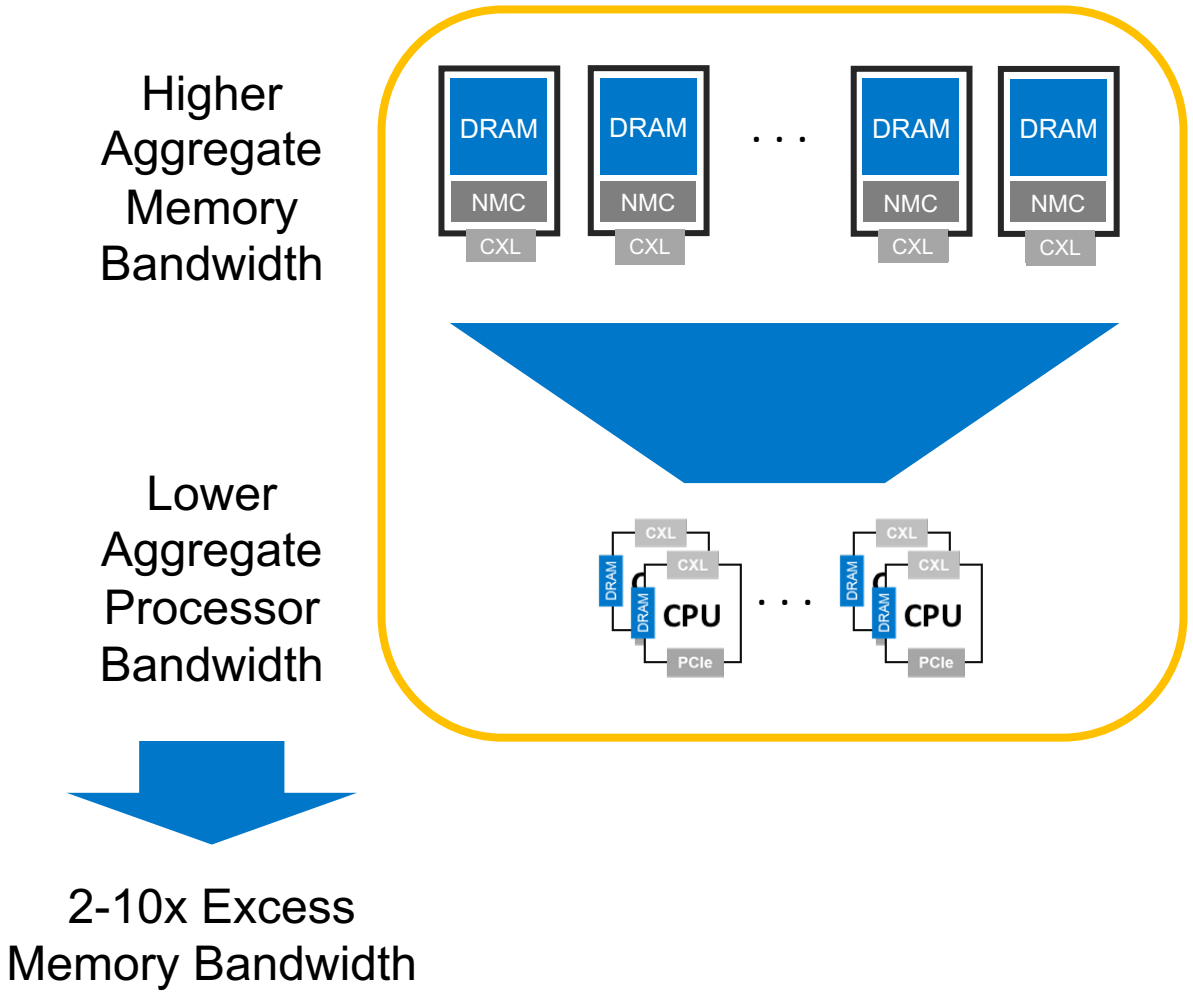
CXL GFAM memory footprint

Memory size independent of # of compute resources

Changes in memory and compute placement

Motivation for Near Memory Compute (NMC)

- Leverage excess disaggregated memory bandwidth to reduce application time-to-solution
- Reduce data transferred to host systems
  - Less fabric cost / energy consumption
- NMC on local memory
  - Reduce memory access latency



HPC software stacks + GFAM

- The larger memory footprint achievable with GFAM makes it possible to leave large datasets in memory
- In-GFAM data in a zero-copy format (e.g., NumPy ndarray, pandas DataFrame) eliminates serialization/deserialization
- This provides:
  - Improved scalability
  - Fastest Time To Insight (TTI)
  - Lowest compute requirements
  - Fewest data movements
  - Lowest energy consumption

One-sided models can be easily adapted to leverage GFAM

MPI
OpenSHMEM

An incomplete list of shared memory models that could be modified to leverage GFAM

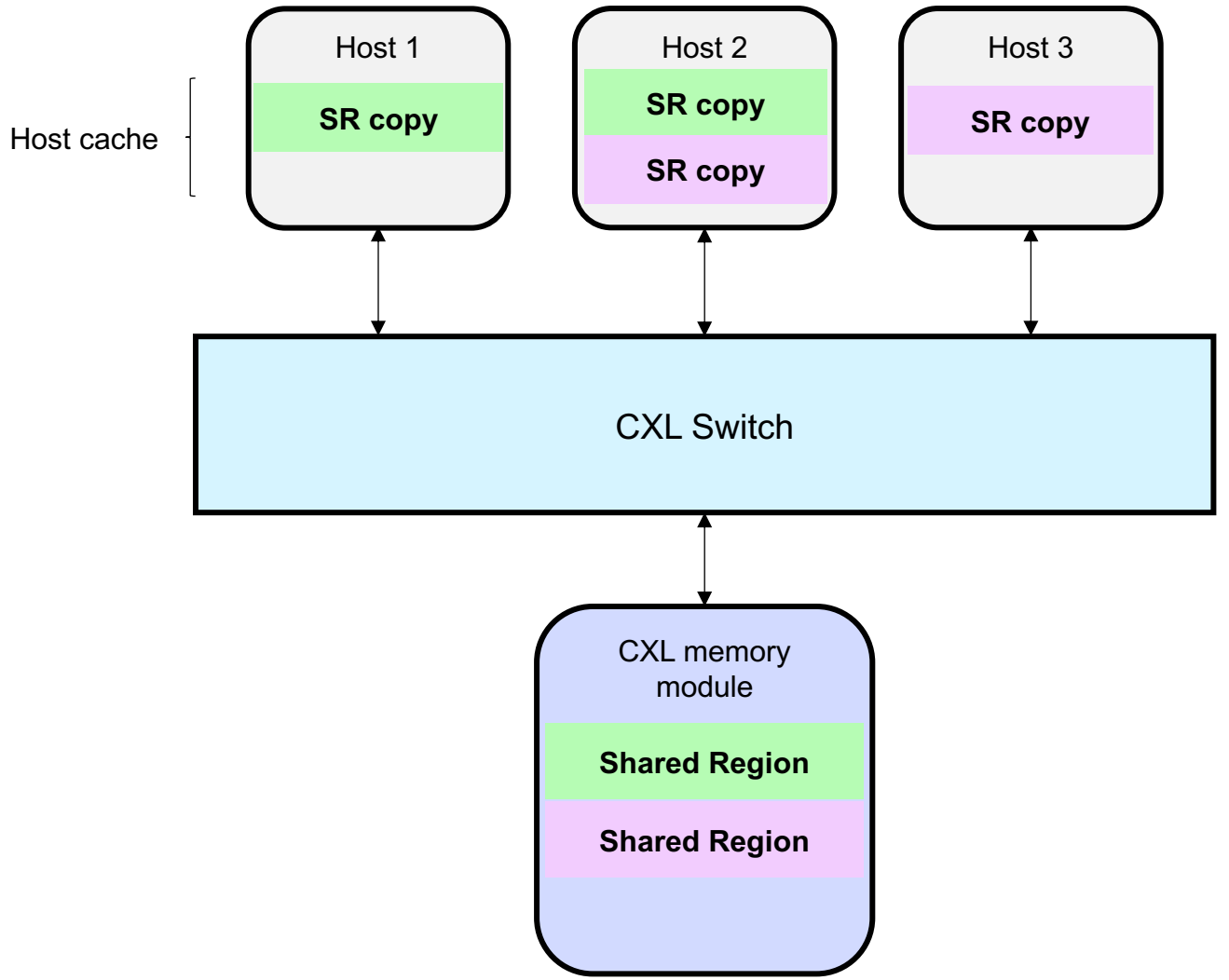
- OpenMP (offload)
- Chapel (global view data structures)
- UPC/UPC++ (global address space)
- Kokkos (DSM)
- Boost (interprocess shared memory object)

Announced software stacks leveraging GFAM

MemVerge: Project Gismo extends Ray's zero-copy-memory-centric architecture across multiple server instances using GFAM

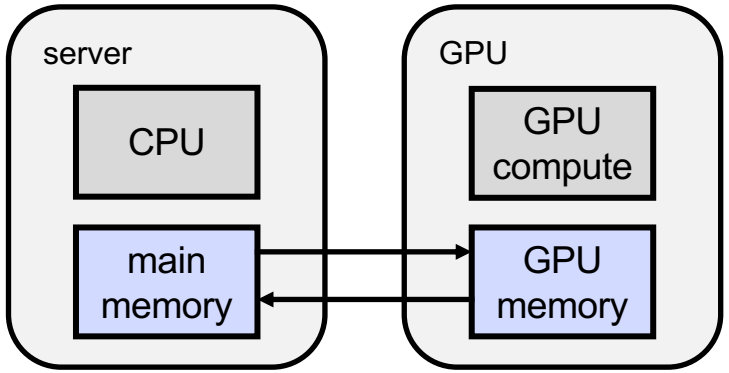
What about coherency?

- CXL 3.0 supports both hardware- and software-based coherency for shared memory
- For small systems hardware coherency is feasible
- For large systems general hardware coherency is not feasible
  - Smaller islands of coherency as a possible mitigation
- CXL features implemented by vendors are driven by use cases
- Vendors will likely start with software coherency

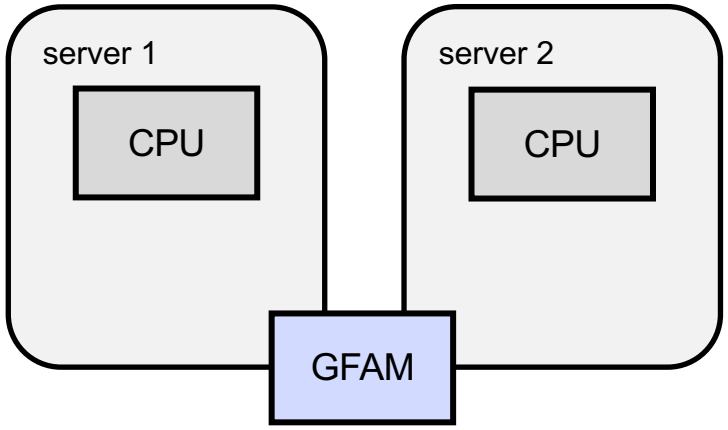


Scaling out OpenMP applications using CXL GFAM

- The OpenMP target construct allows for code regions to be executed on a device and to support multiple devices, device teams, and device synchronization
- Micron has modified the LLVM compiler to enable executions of code regions on multiple x86 servers
- This modified compiler uses CXL GFAM to share memory which eliminates the need to copy data between servers
- Requires modification/addition of OpenMP target pragmas and *omp_target_alloc()* followed by recompilation.



OpenMP offload to GPU



OpenMP offload to server + GFAM

LLM Tokenization

In-GFAM object creation/consumption scales to multiple servers, requires single transformation

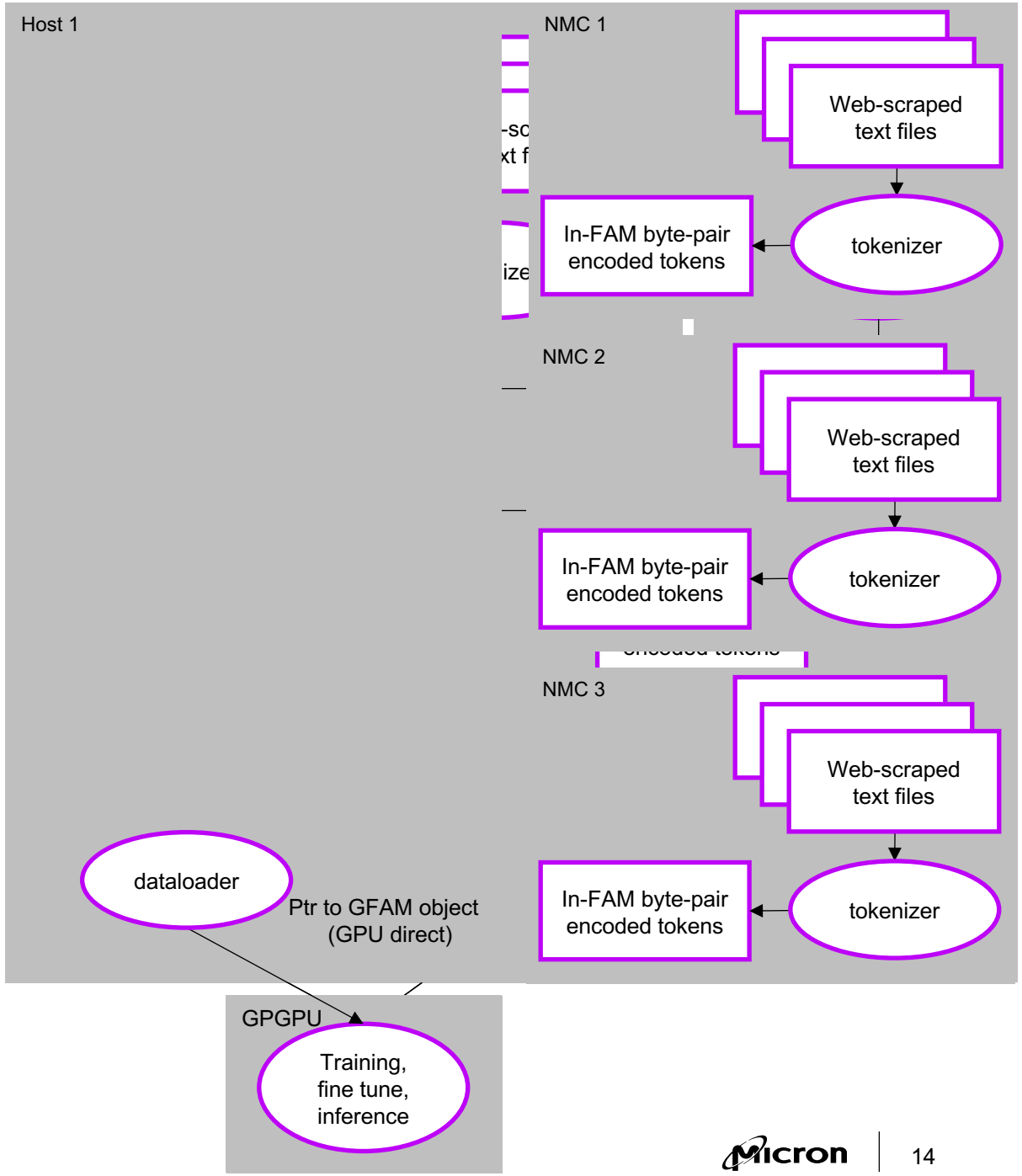
Tokenization leveraging GFAM provides maximum parallelization opportunities. Bandwidth usage is minimized with in-FAM memory object creation/consumption.

Near-memory compute capabilities facilitate further optimization

- **Shard input text files onto separate NMC devices**
- **In-GFAM token creation entirely within a NMC device**

Hugging Face GPT-NeoX tokenizer and c4 dataset

- 1024 files, 770GB
- Tokenization scaleup limited by memory requirements
  - ~175GB per tokenizer

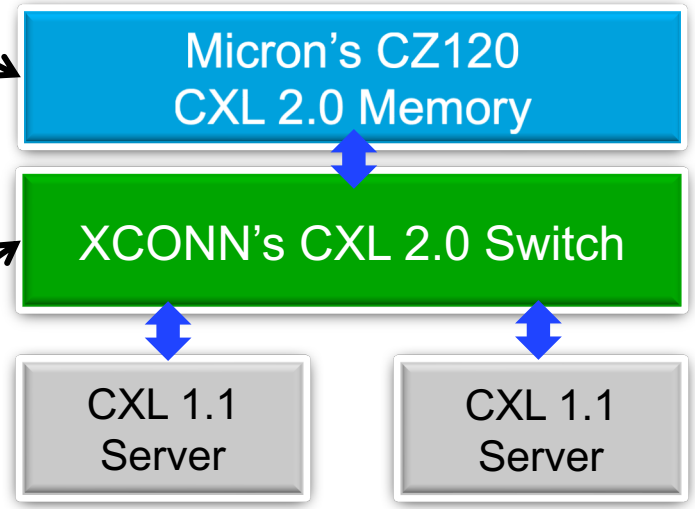


Results of GFAM as a Scaling Solution

Demonstration platform

Micron is producing a series of FPGA based prototype hardware platforms that model Micron's memory centric architecture (via partnership with PNNL)

First platform was delivered mid 2023 (2 host with shared memory)



U.S. DEPARTMENT OF ENERGY
Office of Science

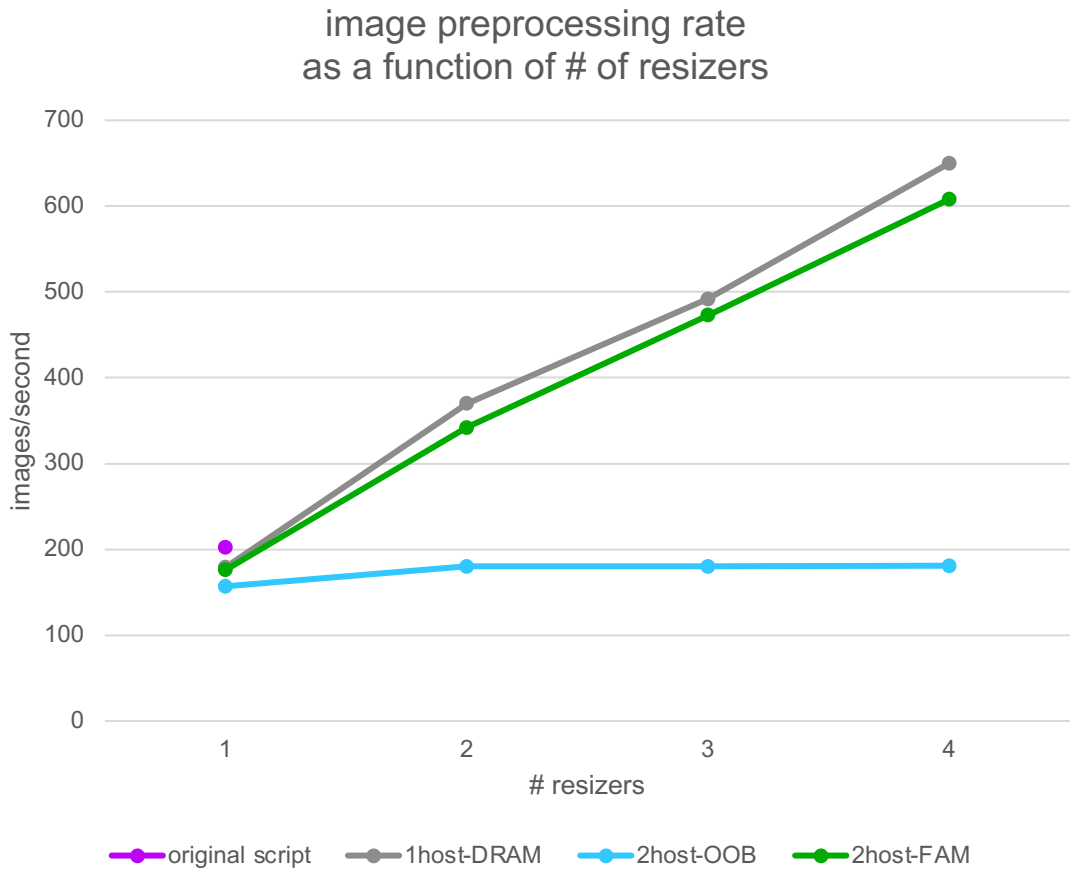
XCONN TECH

CXL FAM Scaling Demonstration

The benefit of splitting computationally bound applications across multiple servers has been limited by the overhead of copying data.

Using shared CXL FAM (**zero-copy, modify-in-place**) provides the utility of shared DRAM while enabling computational scale out across servers.

The shared CXL FAM approach also facilitates scale out using *domain specific accelerators* (e.g., image scaler)

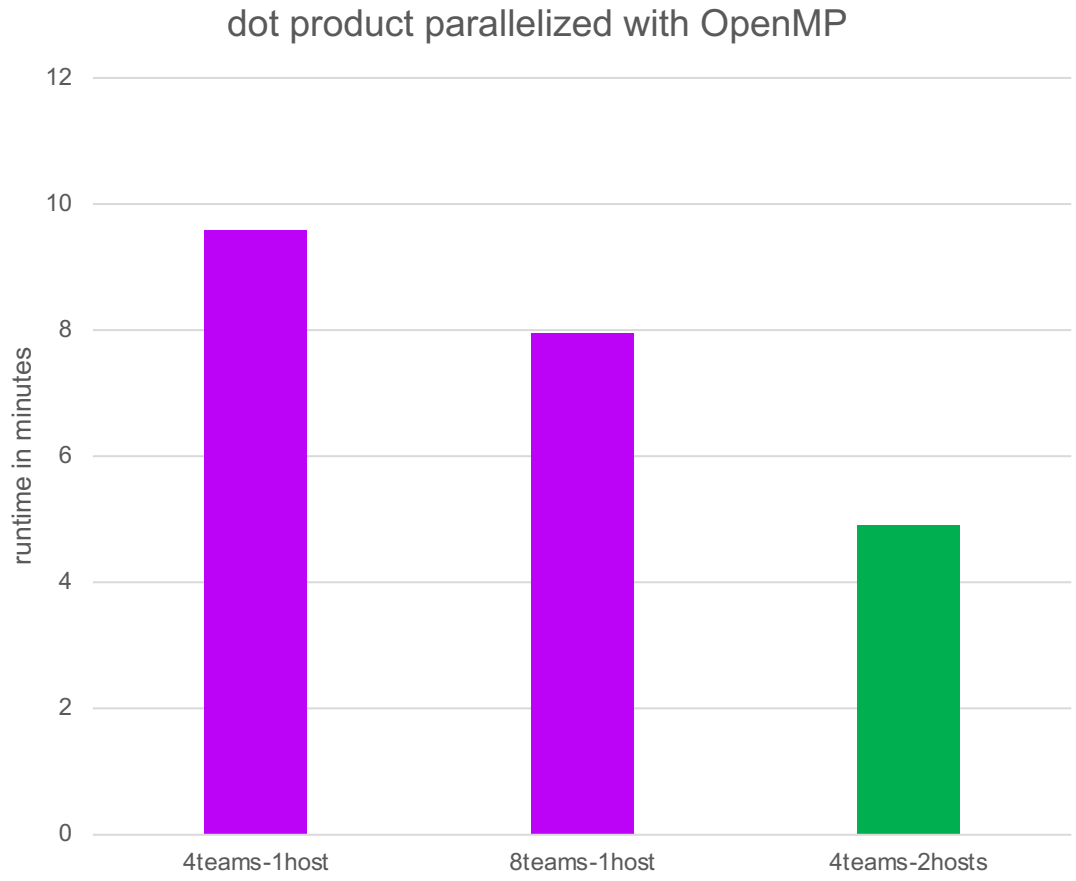


Example with a typical ML image preprocessing pipeline python script

Scale-out OpenMP applications using CXL GFAM

Results shown for 3 configurations

- 4teams-1host: 4 teams of 32 threads each run on a single x86 server
- 8teams-1host: 8 teams of 32 threads each run on a single x86 server
- 4teams-2hosts: 4 teams of 32 threads run on two x86 servers, data shared between servers using CXL GFAM



Example OpenMP dot product test, modify pragmas to scale loops out across servers using CXL GFAM

Using tokenized data

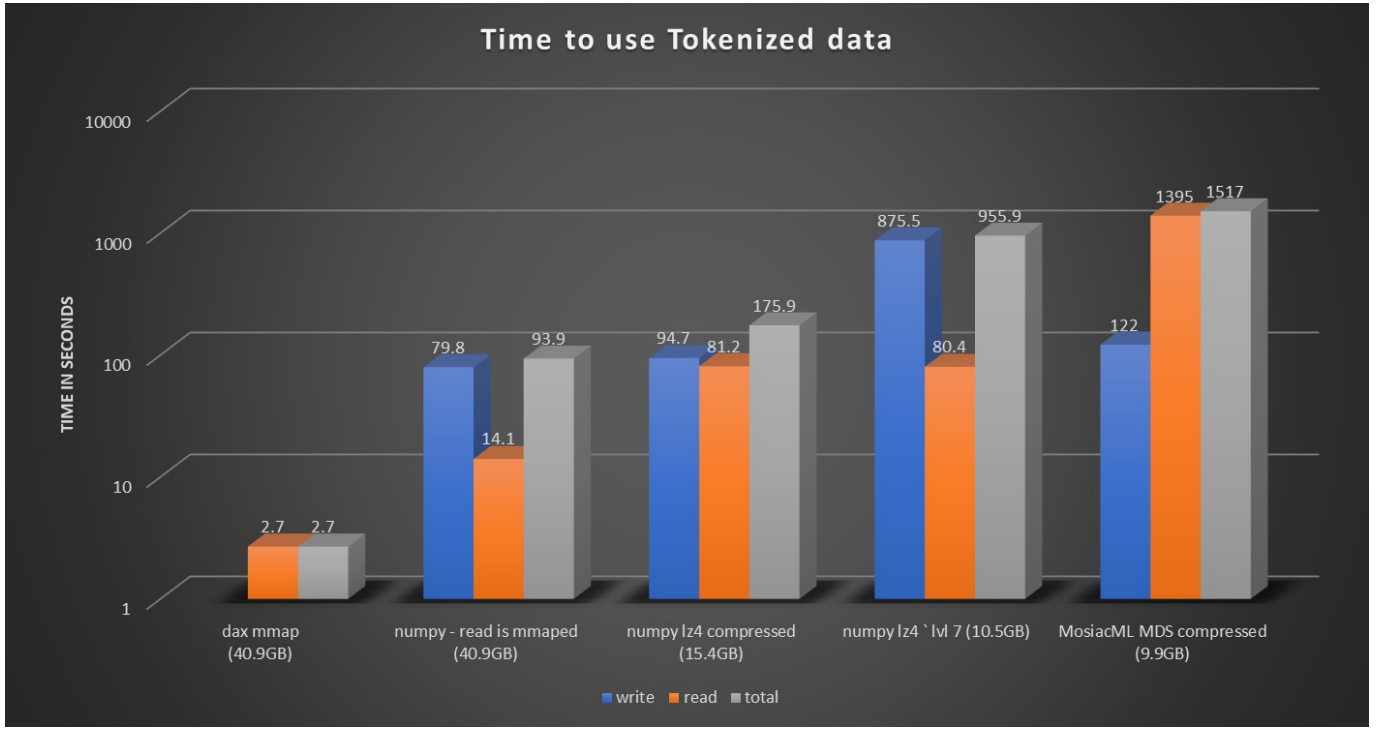
Larger memory footprint enabled by GFAM enables a better solution by leaving data in memory

Leaving tokenized data in GFAM in a zero-copy format (NumPy Ndarrays) eliminates serialization/deserialization.

- Fastest Time To Insight (TTI)
- Lowest CPU requirements
- Lowest network and CXL congestion
- Lowest energy consumption

The time to retrieve the data and place it in a torch tensor is represented by read

For approaches that aren't kept in GFAM, the time to write to a file is also shown

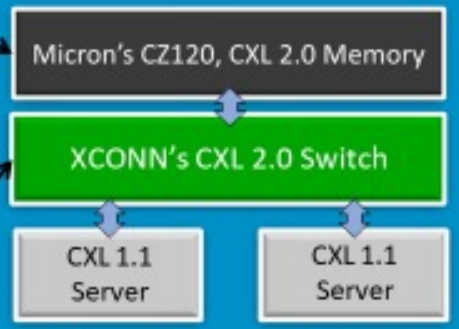


Results with a small input data (30 files from Google's C4 dataset)

A shameless plug for our demonstration ☺

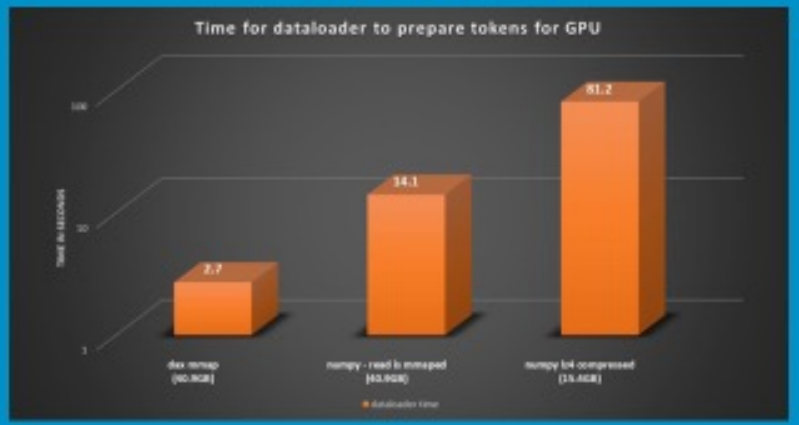
Building Tomorrow's
Multi-Host, Shared Memory
CXL Clusters
with Today's Technology

Demos: Tues & Wed: 1pm & 3pm:
At the CXL: XCONN Kiosk



See a Running Multi-Host, Shared Memory Prototype

- Tokenized Data Processing
- Image Pre-Processing



Micron XCONN TECH
U.S. DEPARTMENT OF ENERGY | Office of Science

Read more about Micron and CXL