MemVerge

Solving HPC/AI's Biggest Memory problems

Bernie Wu

VP Strategic Partnerships/Business Development

Introducing MemVerge

Memory Machine™ X
Memory Expansion & Sharing for AI

Memory Machine™ Cloud
Transparent Checkpointing & Cloud Automation

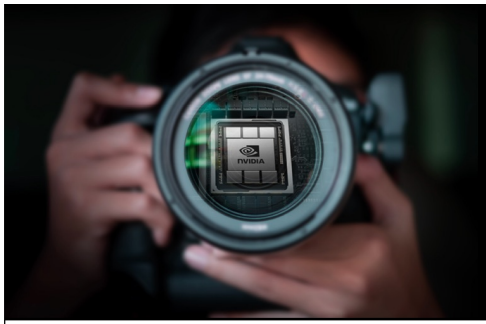
Raised $100M from industry leaders and top-tier VCs

NetApp · CISCO investments · intel capital · JVP · Lightspeed

Key Innovation: Transparent Memory Snapshotting

Checkpoint/Restart of long-running HPC batch jobs/pipeline

- Lower compute/speed up time to insight
- Simulations/EDA/computational biology, etc.
- Cloud/On-prem
- Container/VM/Bare metal
- Integrate Checkpoint engine via REST/ACLI or use MMC Cloud platform
  - Automated Spot Instance surfing & Instance rightsizing
- GPU/Graviton-ARM support in preview mode
  - GPU Resource scheduling/resilience



Achieving K8S and Public Cloud Operational Efficiency Using a New Checkpoint/Restore for GPUs

Author: B. Wu, T. Zhang, Z. Xiong, and Y. Tian

MemVerge

NVIDIA Inception Program

Introduction and Problem Statement

Many AI/ML frameworks and related workflow platforms have built-in checkpoint/restore mechanisms that enable periodic checkpointing of training epochs, MLOp pipeline stages, etc. to provide some level of fault tolerance, tuneability, cloning, and versioning. However, other types of CPU/GPU-based applications may lack such capabilities and/or have either extremely long run times or restart times. Complex initiation sequences and large CPU/GPU memory footprints can contribute to prolonged restart times. Furthermore, there may be a need for system administrators/infrastructure operators to transparently pre-empt running applications to allow re-prioritization and migration workloads with minimum loss of productivity while running at scale. Operators will also need to drain nodes for maintenance/upgrades or for cost-containment reasons, desire to run AI/ML workloads on preemptible public cloud spot instances. These types of use cases require a system-level, application-transparent checkpoint-restart mechanism to be implemented.

A new CUDA driver (12.x) is being developed by Nvidia to support the system-level checkpoint/restore of GPU-**specific memory and machine state within and across Nvidia**'s GPU product lines. The driver deposits the checkpointed state into system memory where it can then be transferred to files as needed. MemVerge and Nvdia have collaborated to modify the open-source CRIU (**C**heckpoint **R**estore **I**n **U**serspace) project to work with this driver and will be upstreaming the changes to the CRIU open-source repository.

Furthermore, MemVerge is characterizing and optimizing the performance and overhead of the checkpoint/restore function as well as integrating it into proof-of-concept demonstrations intended to show how operational efficiency and resiliency can both be increased for K8s and public clouds. The initial Proof-of-Concept showing coordinated CPU-GPU machine state and memory checkpointing and restoration is described below.

Architecture

MemVerge software supercharges checkpointing for applications running on CPU and Nvidia GPU, with no application change and minimal downtime. Here's how:

a. Simultaneous Freeze: pauses both CPU and GPU processes at the same time, leveraging a new CUDA driver function to copy GPU memory to system memory.

b. Quick Resume: The application resumes running immediately while MemVerge efficiently constructs checkpoint metadata, while the complete image is offloaded to storage in the background.

This performance boost enables frequent checkpoints without significant overhead, ensuring application resilience and flexibility.

The restore works in reverse, as follows:

c. Restore CPU memory, GPU memory, and file content from checkpoint image in storage

d. Restore process states in CPU and GPU, and restart the application.

Proof of Concept

System setup:

- CPU: Intel Xeon Gold 5120 with 512 GB DRAM
- Operation System: Ubuntu 20.04.6 LTS
- GPU: x2 NVIDIA V100 with 16 GB GPU RAM
- CUDA Version: 12.X
- Server: Supermicro SYS-1029GQ-TVRT

Application: Tensorflow *mnist* sample with 14GB Data Set

Testing Process:

1. Start *mnist* sample, and confirm the application is running on both CPU and GPU.
2. Use MemVerge *MemoryMachine* to checkpoint the *mnist* process, and save the image to storage. Note that the *mnist* process is killed upon checkpoint completion, so we can show restore on the same node.
3. Restore *mnist* from the checkpoint image. Check the log to confirm the process starts from where it was left off.
4. After *mnist* completes, confirm correctness by checking the result.

Result

The Checkpoint was initiated at Epoch 12/20 , completed in 21 seconds, and then the job was killed manually. The restoration process took 18 sec and the process ran to completion. Correctness was verified.

Please see this QR code to see a recording of the demo.

Conclusion

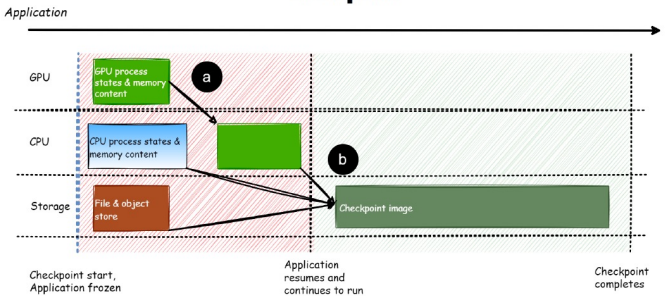
This Proof-of-Concept has successfully demonstrated that the entire CPU/GPU memory state can now be checkpointed/restored transparently. An asynchronous snapshot approach can also be implemented to minimize production interruptions and reduce the snapshot quiescent period. A K8S operator has also been prototyped, however, further development/testing is needed to solidify GPU snapshot error handling The updated CUDA driver is scheduled to be released in H1'24. MemVerge and Nvidia expect to upstream the corresponding changes to the CRIU project in that same timeframe.

This new feature allows Infrastructure architects and MLOps teams to supplement existing application/pipeline-specific snapshot tools by facilitating the automation of other system-level use cases such as spot instances, cloud bursting, job/batch prioritization, and instance rightsizing.

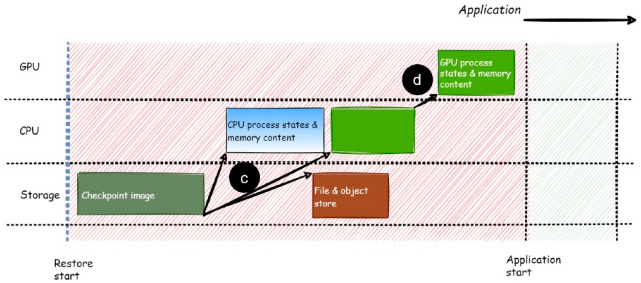
Acknowledgments

The authors would like to acknowledge the contributions of S. Gurfinkel and J. Ramos from Nvidia. Inquiries on this poster may be addressed to bernie.wu@memverge.com

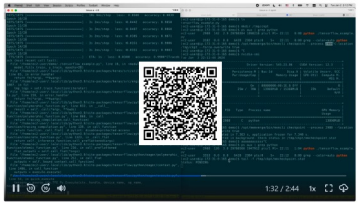
Checkpoint



Restore

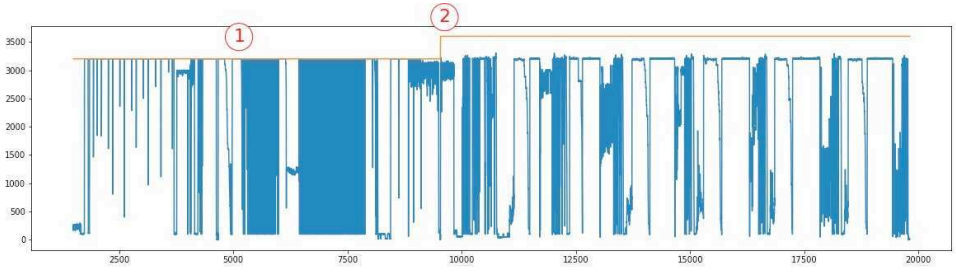


Watch Demo

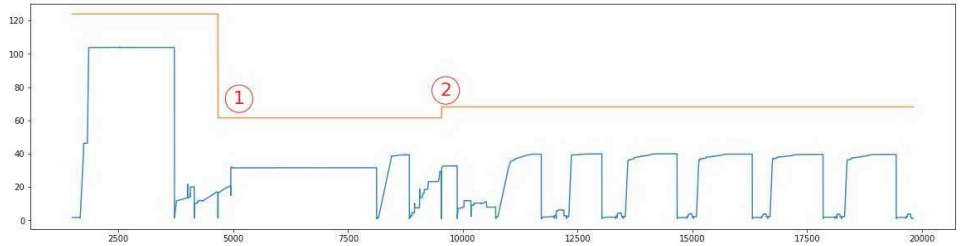


Computational Biology: MEGAHIT* (26GB fastq file)

CPU



Memory



VM Type	WaveRider	Cost	Savings
On-demand	OFF	$9.10	Baseline
On-demand	ON	$7.56	16.9%
Spot	OFF	$2.92	67.9%
Spot	**ON**	**$2.26**	**75.2%**

* Li D, Liu CM, Luo R, Sadakane K, Lam TW. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. Bioinformatics. 2015 May 15;31(10):1674-6. doi: 10.1093/bioinformatics/btv033. Epub 2015 Jan 20. PMID: 25609793.

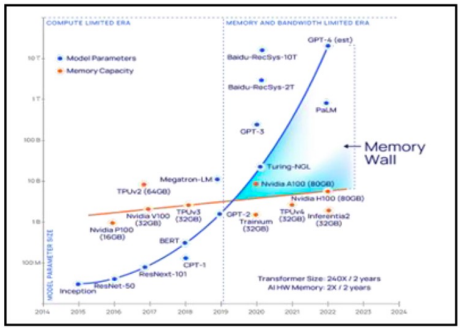
MemVerge

MemVerge

Computer Express Link (CXL): Emergence of New Memory Architectures

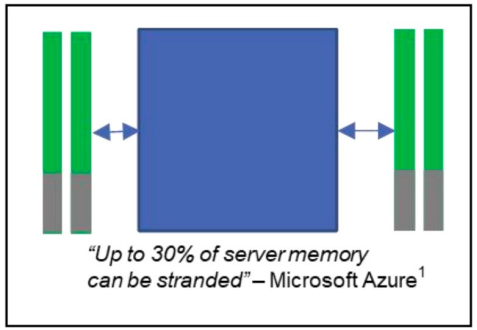
CXL Motivations

AI Memory Wall: Need More High-BW Mem



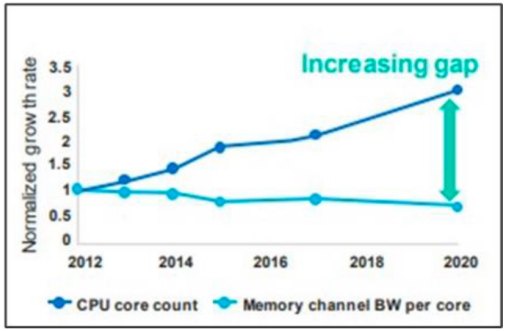
https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8

Unutilized Memory: Need Access



Source: https://ieeexplore.ieee.org/document/10034802

More Cores: Need more Memory BW



Source: Meta OCP Presentation Nov 2021

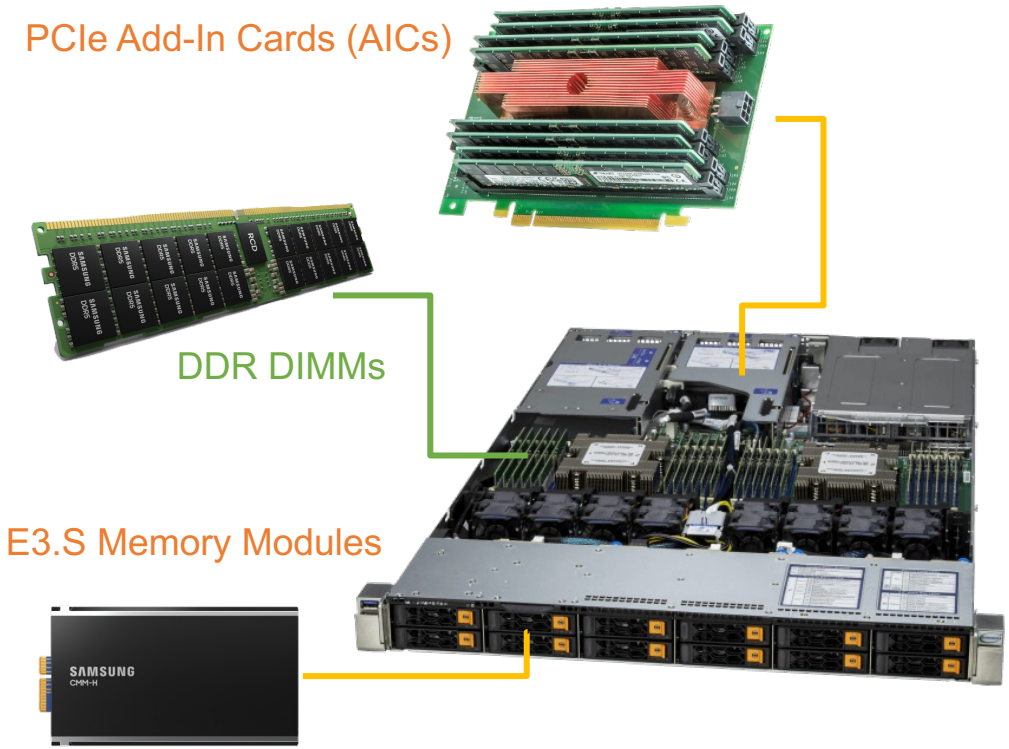
CXL Board of Directors

CXL Compute Express Link™

Industry Open Standard for High Speed Communications

255+ Member Companies

CXL 1.1 Memory Expansion



Add-in Card (AIC)

- Flexible capacity, up to 2 TB per card
- Higher bandwidth, up to x16 PCIe5 lanes (~ 1x DDR5 channel)

E3.S Modules

- Easy front loading, same as SSDs
- Fixed capacity – 128, 256, & 512 GB
- Lower bandwidth at x8 PCIe5 lanes

Key Innovation: CXL Memory software

Memory Tiering/Bandwidth Optimization software for CXL Memory (CXL 1.1)

- Address memory-bound applications such as AI Inferencing, simulations
- Cost reduce compute for applications needing high GB/Core configurations
- Tiering software automatically promotes/demotes hot/cold memory pages

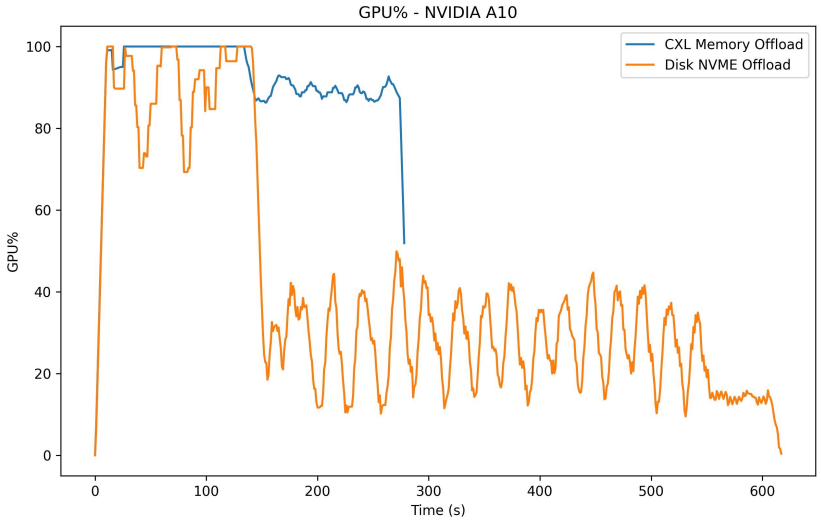
Memory Pooling/Sharing software (CXL 2.0/3.1)

- AI/ML: Embedding/Activation offloads, KV cache, checkpoints, weights, etc.
- High performance pub-sub
- Reference dataset sharing
- Composable remote memory

Use MemVerge CXL Memory Expansion/Tiering Software to Increase GPU Utilization of Batch Inferencing

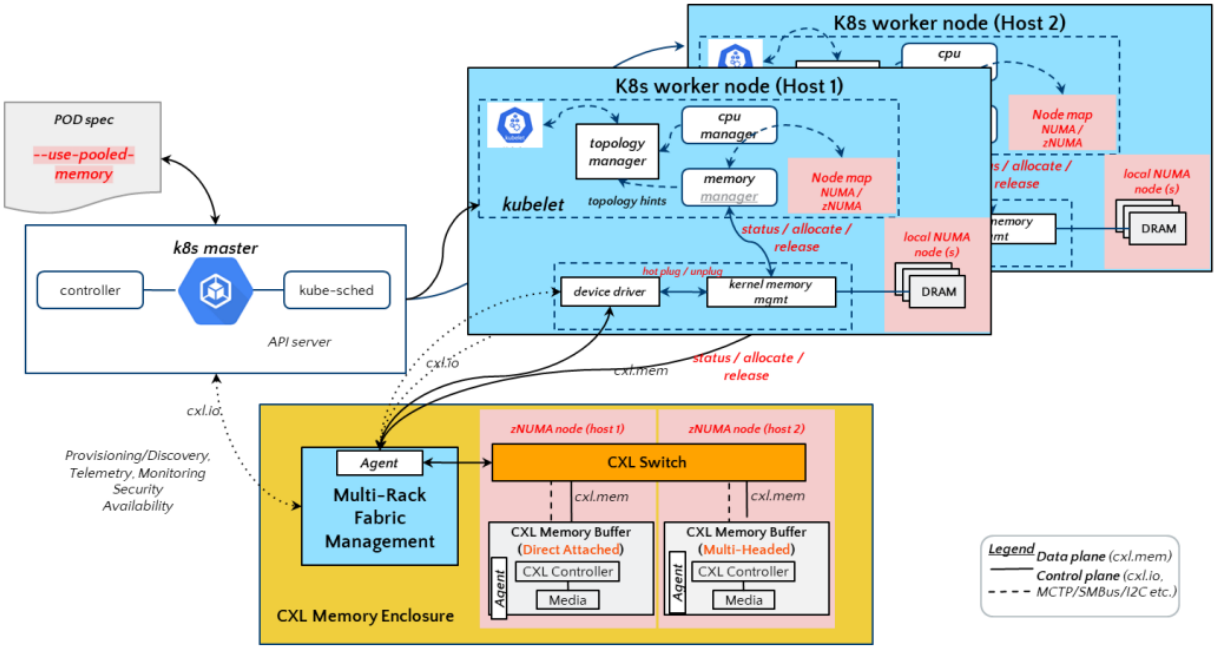
Flexgen Inferencing Benchmark OPT-66B

Batch size=24
Prompt=512
GPU=4



- model size: 122.375 GB cache size: 109.688 GB hidden size (p): 0.857 GB
- peak gpu mem: 20.130 GB projected: False
- prefill latency: 123.268 s prefill throughput: 398.740 token/s
- decode latency: 146.758 s decode throughput: 4.579 token/s
- total latency: 270.026 s total throughput: 2.844 token/s

CMS – Pooled Memory (K8s Example)



- **Memory attributes not visible to service owners** when requesting deployment of their instances
- Uses **cpuset.mem semantics** that use linux NUMA topology maps
- Pooled memory useful for **burstable / best effort type PODs**/containers without “local numa settings”
- During bringup, **appliance registers itself with k8s** master as a “MemoryClass”
- When POD spec is parsed, k8s sched will check if local memory on nodes can be used to place container
- If none of the nodes have enough local memory, k8s sched will hotplug a slice from the appliance to one of the hosts. host device driver will process the hot plug event and assign memory within the remote numa node
- Kubelet on that node will see memory in the updated node map and accept the pod spec for deployment on pooled + local memory
- once the pod is unallocated, memory is released back to the pool

MemVerge

Thank You
&
See you at SC24!

April 2024

bernie.wu@memverge.com