In-Depth Analyses of Unified Virtual Memory System for GPU Accelerated Computing

Tyler Allen, Rong Ge

Clemson University

Accelerated Computing

- Accelerators are hot trend HPC/Cloud systems.
  - System Model: CPU host offloading “hard” tasks to accelerator.
  - Accelerators are frequently GPUs; other devices are interesting.

Accelerated Computing

- Accelerators are hot trend HPC/Cloud systems.
  - System Model: CPU host offloading “hard” tasks to accelerator.
  - Accelerators are frequently GPUs; other devices are interesting.
- Increasingly complex system $\rightarrow$ complex programming; difficulties:
  - Programming against independent architectures.
  - Different programming paradigms.
  - Low level hardware management and coordination.

Accelerated Computing

- Accelerators are hot trend HPC/Cloud systems.
- Increasingly complex system → complex programming.

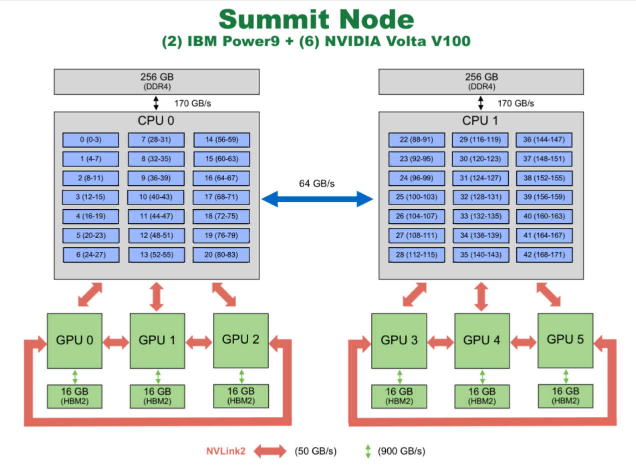


image credit: ORNL/OLCF

Programmability of Accelerated Computing

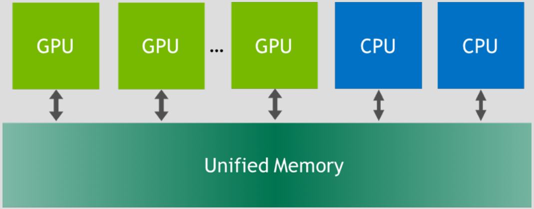
- High programmability is required for heterogeneous systems.
  - Hardware control code, e.g. memory management, is notoriously difficult.
  - Heterogeneous systems require far more low-level control.
  - HPC systems are frequently used by domain scientists - not software engineers.

Programmability of Accelerated Computing

- High programmability is required for heterogeneous systems.
- Memory management is increasingly hard for heterogeneous memory systems.
  - Data must be explicitly transferred between host, accelerator.
  - This is hard for deep data structures, e.g. sparse matrix representations.
  - Code is now increasingly about memory management, not domain science.

Programmability of Accelerated Computing

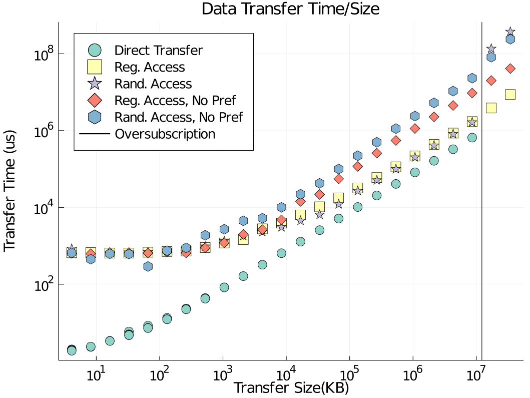
- High programmability is required for heterogeneous systems.
- Memory management is increasingly hard for heterogeneous memory systems.
- Systems should carry more of the programming burden - particularly memory:
  - Shared memory semantics have been applied to heterogeneous systems.
  - This provides a **unified memory** address space.
  - This can effectively eliminate memory programming - with a catch.
  - Should support a common interface, mechanism for arbitrary accelerators.



source: https://developer-blogs.nvidia.com/wp-content/uploads/2017/11/Unified-Memory-MultiGPU-FI.png

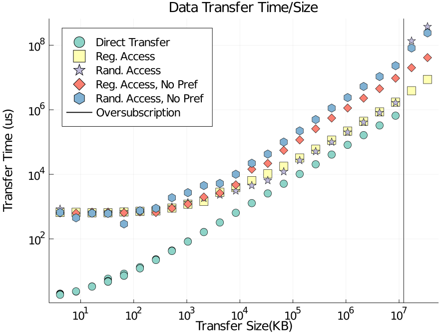
The Performance Problem

- Unified Memories for Heterogeneous Computing are slow.
  - Figure: performance (hexagon, diamond) is *bad* compared to direct transfer.
  - Special optimizations can narrow the gap (square, star).
  - Still significant gap between direct management and systems software.



The Performance Problem

- Unified Memories for Heterogeneous Computing are slow.
  - Figure: performance (hexagon, diamond) is *bad* compared to direct transfer.
  - Special optimizations can narrow the gap (square, star).
  - Still significant gap between direct management and systems software.



- Research Questions:
  - What causes the gap between management and systems software?
  - Are there specific costs that can be optimized?

Our Work

- Our work investigates system-level performance issues for unified memory.
  - We look at system-level root causes of performance loss.
  - We examine software and hardware costs at host, accelerator.
- Three key sections:
  - Accelerator system-workload creation
  - Top-down driver/runtime performance analysis
  - Performance analysis with advanced features

State of the Art

- Two main categories: Application Analysis and Redesign/Optimization.
- Application Analysis:
  - Most study overall application performance vs. direct transfer.
  - **Our work** takes system-level perspective, identifies root causes.

State of the Art

- Two main categories: Application Analysis and Redesign/Optimization.
- Application Analysis:
  - Most study overall application performance vs. direct transfer.
  - **Our work** takes system-level perspective, identifies root causes.
- Redesign/Optimization:
  - Primarily GPU, interconnect, and/or memory hierarchy alterations
  - Requires full-system hardware design changes - mid-to-far future.
  - **Our work** focuses on existing, near-future software, hardware.

Environment: Unified Virtual Memory

- UM defines a generic hardware/software interface and behavior.
- Standard Characteristics of Unified Memory:
  - Extend host virtual memory to support accelerator.
  - This includes data migration systems, specifically **paging**.

Environment: Unified Virtual Memory

- UM defines a generic hardware/software interface and behavior.
- Standard Characteristics of Unified Memory:
  - Extend host virtual memory to support accelerator.
  - This includes data migration systems, specifically **paging**.
- Our experiments use Unified Virtual Memory (UVM).
  - NVIDIA's unified memory implementation

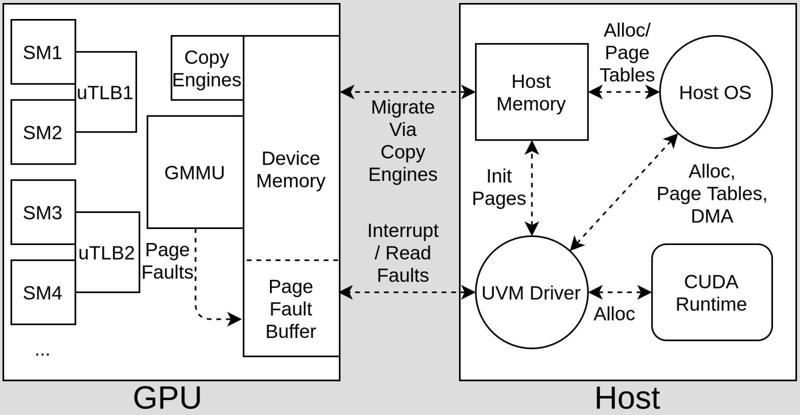
Environment: Unified Virtual Memory

- UM defines a generic hardware/software interface and behavior.
- Standard Characteristics of Unified Memory:
  - Extend host virtual memory to support accelerator.
  - This includes data migration systems, specifically **paging**.

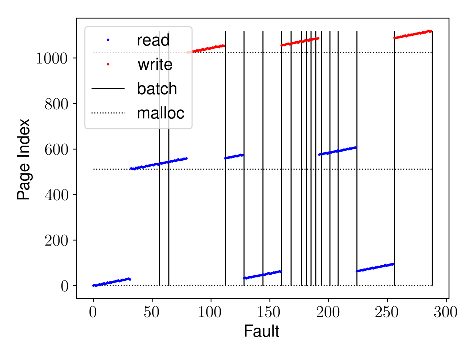
- Our experiments use Unified Virtual Memory (UVM).
  - NVIDIA's unified memory implementation
- Heterogeneous Memory Management (HMM) as an alternative?
  - HMM is Linux-integrated support for unified memory.
  - HMM only resolves host-side management; requires companion device driver (UVM).
- We use UVM as an example implementation and testbed.

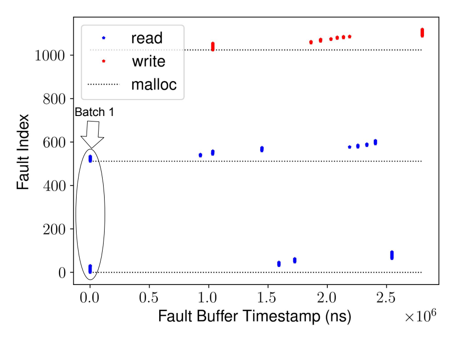
Background: UVM System Architecture

- UVM is managed by the UVM driver on the host.
  - Interactions triggered by interrupt to wake up driver.
  - Faults are written to fault buffer by GPU, read by host.
  - Optimization: Faults read, serviced in **batches**.
    - Many faults handled at the same time.



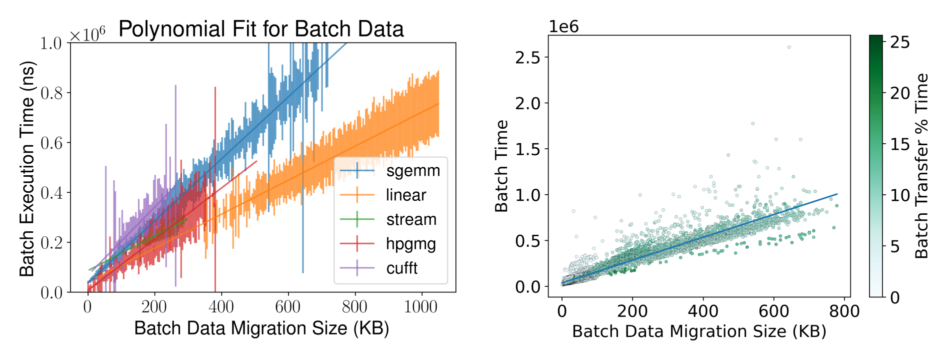
Characterizing Fault Batch Workload





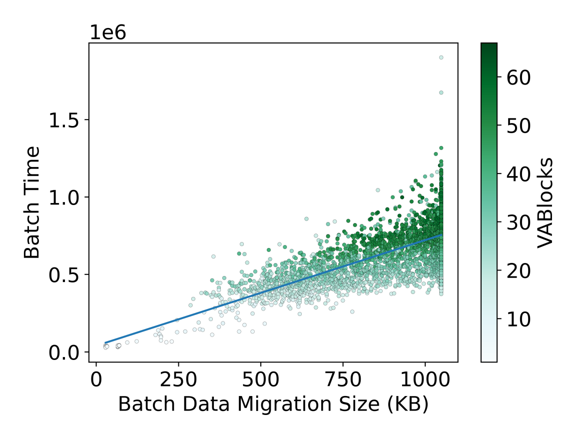
- This is a simple parallelized vector addition.
- UM causes GPU computation, fault handling to be largely serialized.
  - Page faults are generated much faster than they are serviced.
- At scale, fault batches contain similar number of faults from all SMs.
  - Several rate-limiting mechanisms enforce this.

Batched Data Movement Across Applications



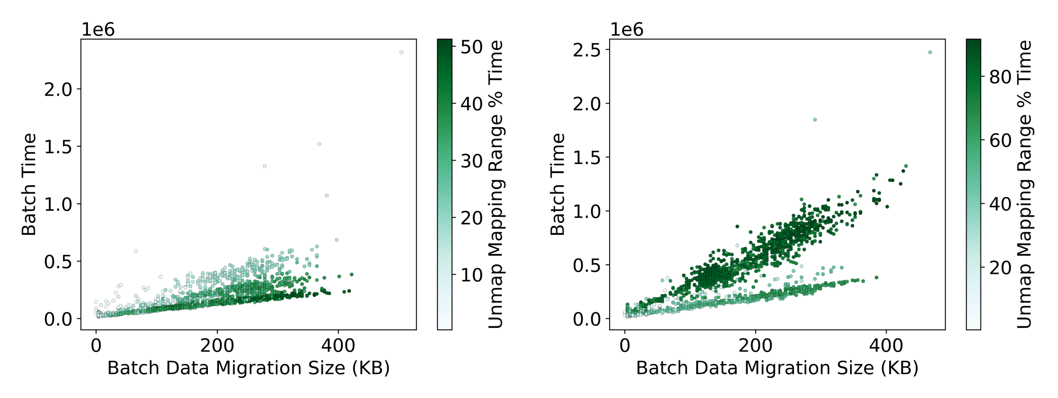
- Data migration cost varies between applications due to batch composition.
- Data migration itself is **not** the majority of the time cost.

Batch-Level Access Pattern



- VABlock: Memory abstraction, 2MB contiguous chunks.
  - This is a rough metric for page-level locality.
- Each VABlock is processed independently.
  - This creates a source of performance variance in batch processing.

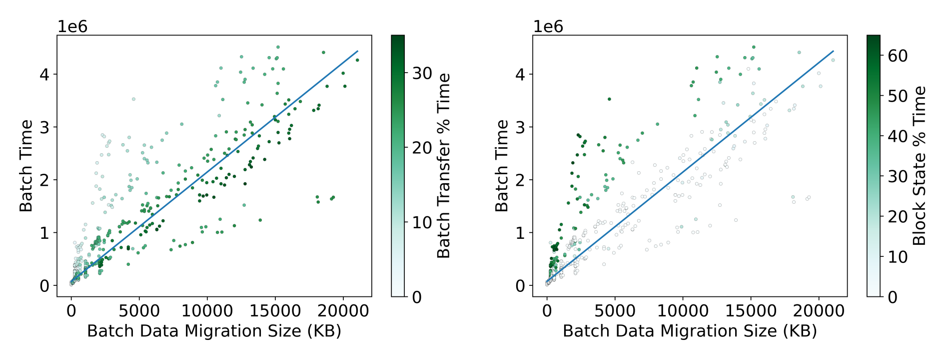
Case: Host Page Unmapping Cost



(a) OMP - 1 Thread (b) OMP

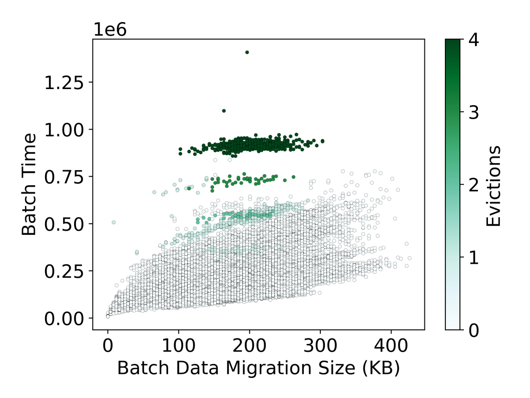
- Pages are unmapped from the host during a fault before migration.
- We find this cost takes a significant amount of batch time.
- We also find application parallelism can exacerbate this*.
- Ongoing work identifies this cost to be primarily **TLB Shootdowns**.

Batch Analysis with Prefetching Enabled



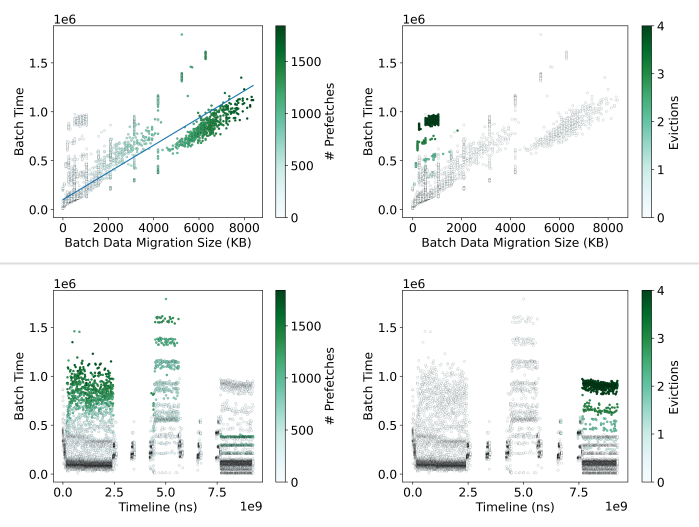
- Overall, prefetching reduces the number of batches.
- High-cost batches are still present (unmapping).
- Outliers are attributed to DMA tracking and management data structures (Right).
  - Example of software and host OS impacting latency.

Batch Analysis with Memory-Oversubscribed Workloads



- Oversubscription: Feature enabling page swapping for out-of-core GPU workloads.
- Due to problem size, naturally there are more batches.
- Evictions distinguish themselves as a flat cost per eviction.

Prefetching+Eviction Batch Analysis



- Prefetching and eviction are indirectly related performance-wise.

Key Insights, Contribution, and Future Work

- Contributions
  - Characterization of GPU hardware fault creation.
  - Comprehensive batch-level performance analysis.
  - Reproducable data collection methodology for future work.

Key Insights, Contribution, and Future Work

- Contributions
  - Characterization of GPU hardware fault creation.
  - Comprehensive batch-level performance analysis.
  - Reproducable data collection methodology for future work.
- Key Insights
  - Interconnect bandwidth is **not** the key bottleneck.
  - Host architectural issues are also a major component, e.g. TLB coherency.
  - Advanced features alter fault generation rate, but fundamentals stay the same.

Key Insights, Contribution, and Future Work

- Contributions
  - Characterization of GPU hardware fault creation.
  - Comprehensive batch-level performance analysis.
  - Reproducable data collection methodology for future work.
- Key Insights
  - Interconnect bandwidth is **not** the key bottleneck.
  - Host architectural issues are also a major component, e.g. TLB coherency.
  - Advanced features alter fault generation rate, but fundamentals stay the same.
- Future Work
  - Investigate GPU-to-GPU UM behaviors.
  - Optimize page unmapping, tlb coherency for improved generic UM performance.
  - New algorithms for eviction to eliminate flat cost.

Acknowledgements

- Tyler Allen
  - tnallen@clemson.edu
  - tnallen.people.clemson.edu
  - **Defending in Spring 2022**
  - Looking for research, academic positions
- Rong Ge
  - rge@clemson.edu
  - people.cs.clemson.edu/~rge/

- See our related paper! - *Demystifying GPU UVM Cost with Deep Runtime and Workload Analysis*
- Support from NSF: Grants CCF-1551511 and CNS-1551262.