Exploring Page Fault Servicing in Nvidia's UVM Driver

Pranjal Singh

November 2024

Undergraduate Project - III

Supervisors: Swarnendu Biswas and Debadatta Mishra

Nvidia Unified Virtual Memory

- Pointers can be used from CPU and GPU
- Explicit memory transfers are not needed

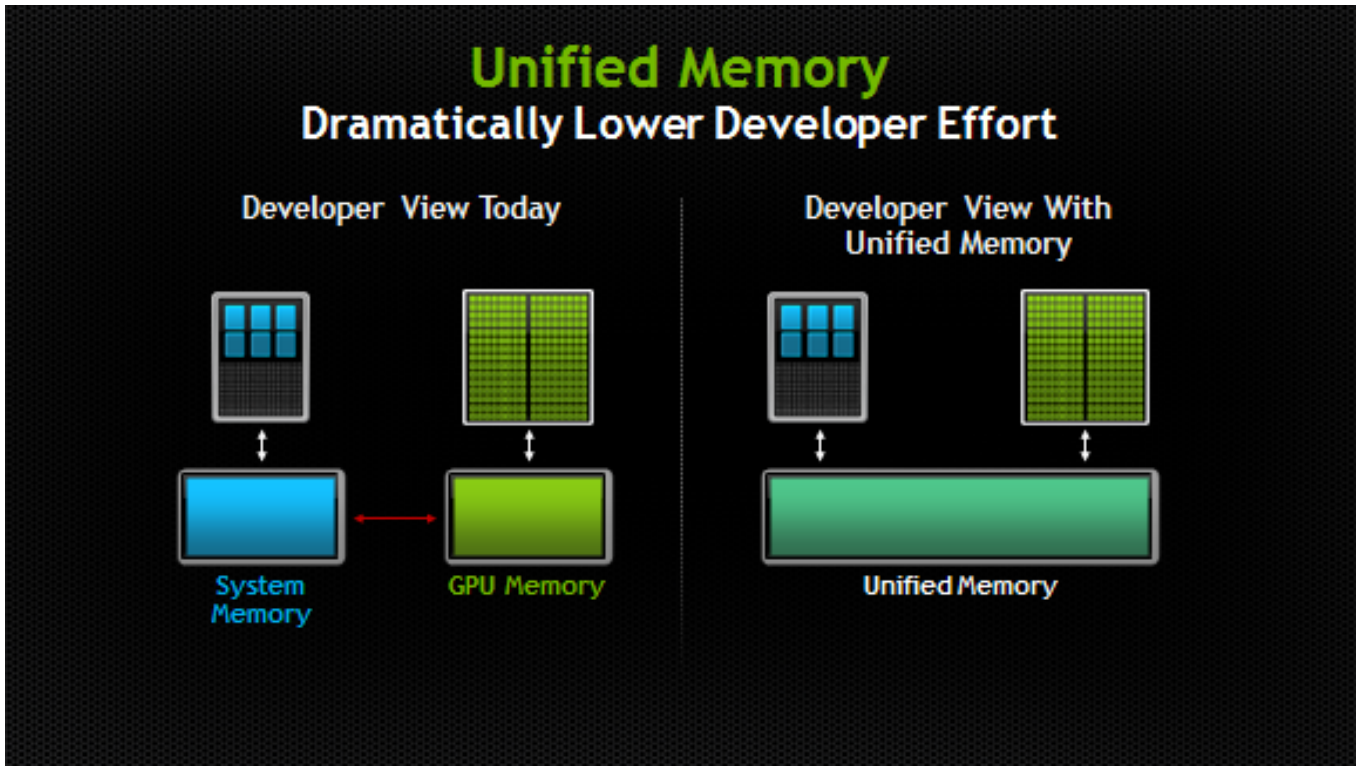


Image Source: https://developer.nvidia.com/blog/powerful-new-features-cuda-6/

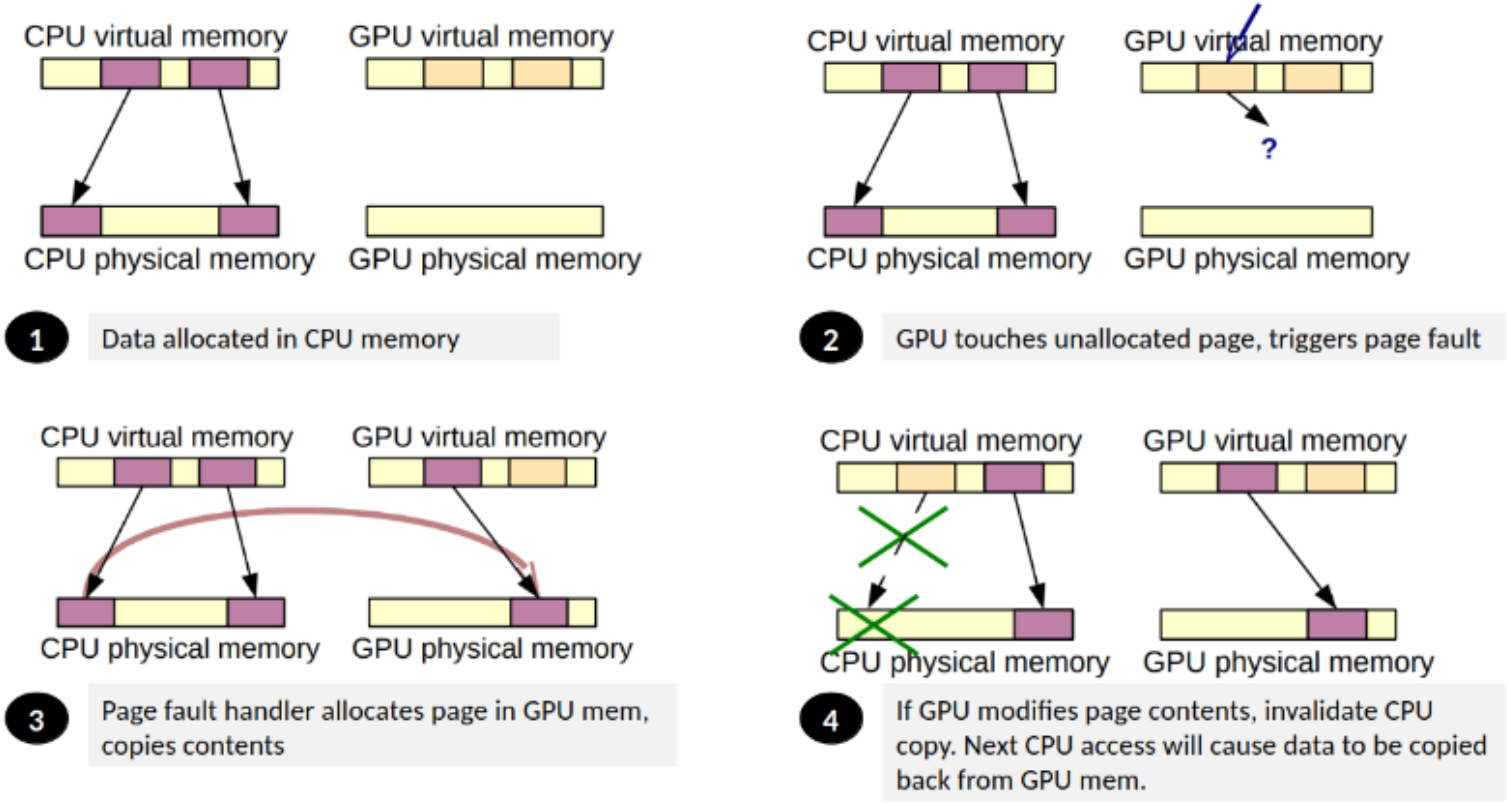
Nvidia Unified Virtual Memory

```
/*
 * Copy-then-execute
 *
 * hA resides on CPU
 * dA resides on GPU
 */

hA = malloc(N);
fread("input.txt", hA, N);

dA = cudaMalloc(N);

cudaMemcpy(dA, hA, N);
compute<<<512, 32>>>(dA, N);
cudaMemcpy(hA, dA, N);

fwrite("output.txt", hA, N);
```

```
/*
 * UVM
 *
 * uA resides in Unified
 * Virtual Memory
 */

uA = cudaMallocManaged(N);
fread("input.txt", uA, N);

// no-op

// no-op
compute<<<512, 32>>>(uA, N);
// no-op

fwrite("output.txt", uA, N);
```

Sample copy-then-execute and UVM programs

Demand-based Page Migration



- A page can reside in CPU or GPU memory
- Page faults are raised when the page is on the other device

---

https://www.irisa.fr/alf/downloads/collange/cours/hpca2020_gpu_2.pdf

https://cse.iitk.ac.in/users/swarnendu/courses/autumn2024-cs610/gpu-cuda.pdf

Demand-based Page Migration

- Permits oversubscribing GPU memory
- Pages are *evicted* if GPU memory is full

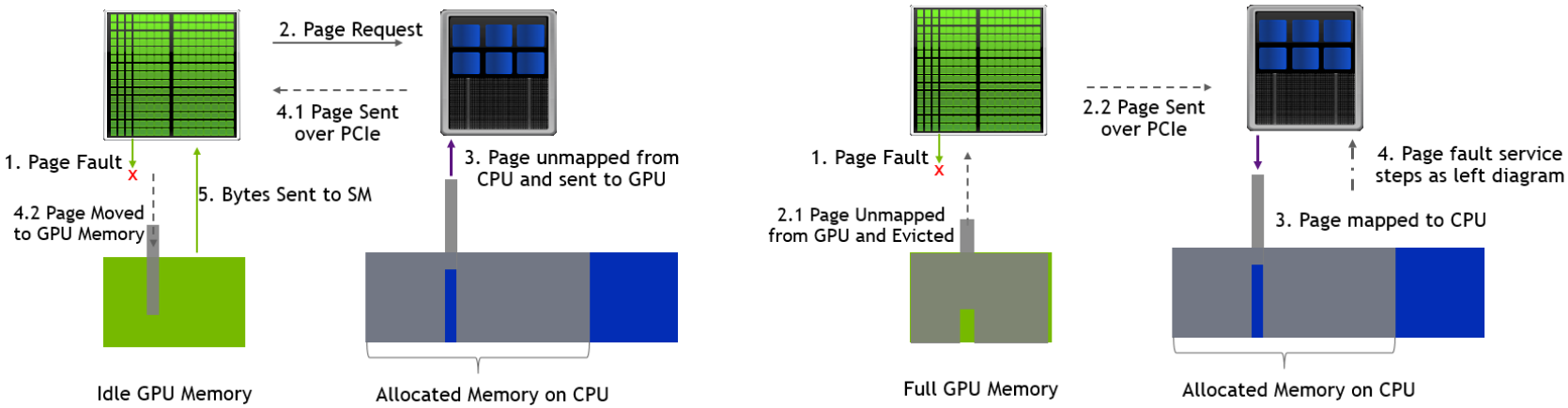


Figure: GPU far faults (left) and GPU oversubscription (right)

https://developer.nvidia.com/blog/improving-gpu-memory-oversubscription-performance/

Nvidia Unified Virtual Memory - Virtualization Overhead

- Page faults incur higher latency and CPU usage
- UVM is costlier despite overlapping computation and page migration
- Fault servicing latency: 30-45 μs

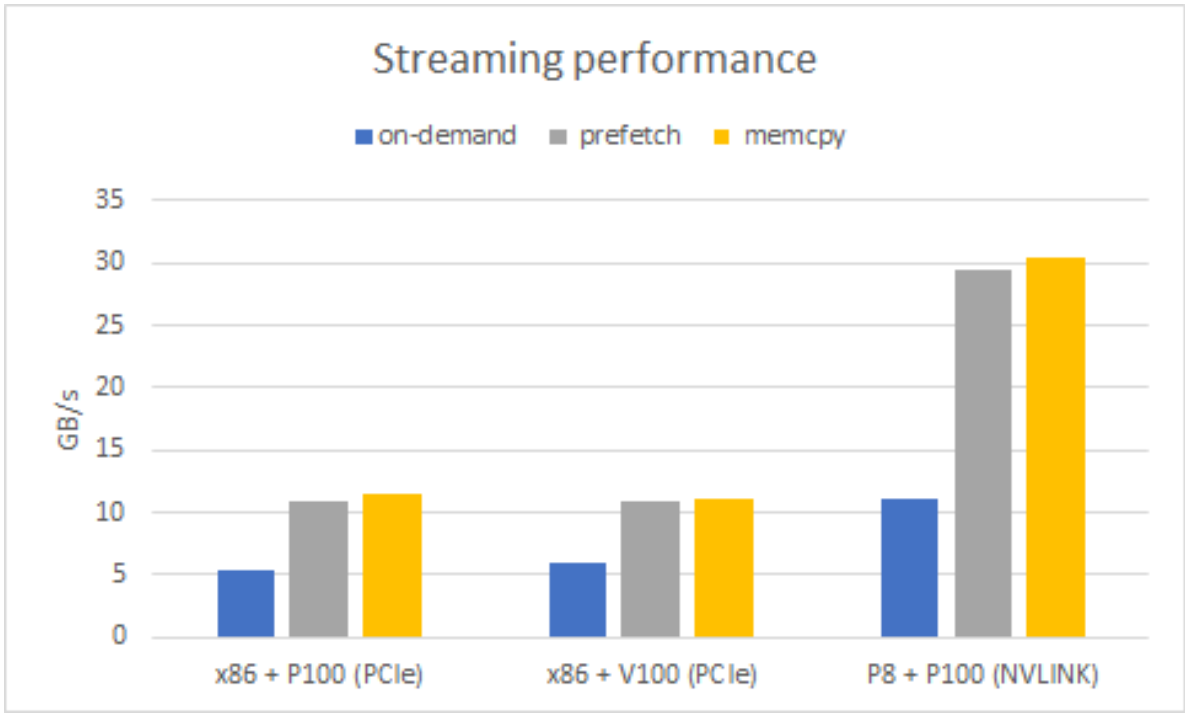


Image source: https://developer.nvidia.com/blog/maximizing-unified-memory-performance-cuda/

CUDA Software Stack

- Compiler: `nvcc`
- Userspace library/runtime: `libcudart.so`
- OS driver (loadable module): `nvidia-uvm.ko`, ...
- Profiling and debugging tools
- ...

CUDA Software Stack

- Compiler: `nvcc`
- Userspace library/runtime: `libcudart.so`
- OS driver (loadable module): `nvidia-uvm.ko`, ...
- Profiling and debugging tools
- ...
- Memory management and UVM are implemented in the driver
- Open-sourced in 2022

CPU Faults

- Page is resident on GPU. CPU accesses it
- Program execution *cannot* continue

CPU Faults

- Page is resident on GPU. CPU accesses it
- Program execution *cannot* continue
- Fault servicing is done in kernel-mode execution of the same process

CPU Faults

- Page is resident on GPU. CPU accesses it
- Program execution *cannot* continue
- Fault servicing is done in kernel-mode execution of the same process
- Service Routine:
  1. Unmap page from GPU
  2. Initiate page migration (asynchronous)
  3. Map page in CPU page table
  4. Wait for migration
  5. Return to userspace, repeat instruction

GPU Faults

- Fault is sent to CPU
- Fault buffer is periodically cleared by a thread

GPU Faults

- Fault is sent to CPU
- Fault buffer is periodically cleared by a thread
- Meanwhile, other thread blocks are scheduled
- GPU faults can be *batched*

GPU Faults

- Fault is sent to CPU
- Fault buffer is periodically cleared by a thread
- Meanwhile, other thread blocks are scheduled
- GPU faults can be *batched*
- Fault service routine:
  1. Fetch fault(s) from fault buffer
  2. Preprocess faults (de-duplicate, group)
  3. Initiate page migration (asynchronous)
  4. Unmap page from CPU
  5. Map page in GPU memory
  6. Wait for migration
  7. "push replay" on GPU, repeat faulting instruction

Memory Prefetching

- Neighbouring Pages are selected for prefetching and transferred in advance
- Current prefetching policy: tree-based neighbourhood prefetcher

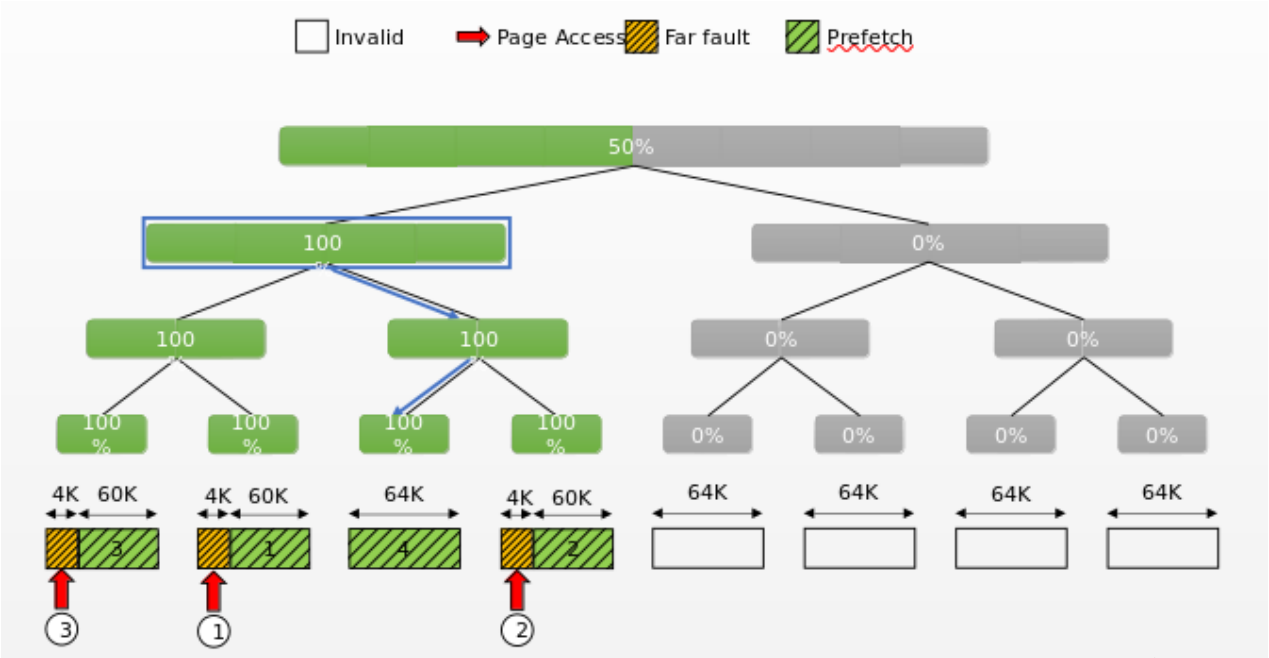


Figure: Tree-based Neighbourhood Prefetcher [1]

[1] Source: https://people.cs.pitt.edu/ debashis/presentations/ISCA2019.pptx

Performance Considerations

Performance Considerations

- Latency of fault servicing

Performance Considerations

- Latency of fault servicing
- Is fault servicing done in a single step or deferred?

Performance Considerations

- Latency of fault servicing
- Is fault servicing done in a single step or deferred?
- Is it multithreaded?

Performance Considerations

- Latency of fault servicing
- Is fault servicing done in a single step or deferred?
- Is it multithreaded?
- What is the cost breakdown?

Performance Considerations

- Latency of fault servicing
- Is fault servicing done in a single step or deferred?
- Is it multithreaded?
- What is the cost breakdown?
- Does the GPU “waste” time waiting?

Performance Considerations

- Latency of fault servicing
- Is fault servicing done in a single step or deferred?
- Is it multithreaded?
- What is the cost breakdown?
- Does the GPU "waste" time waiting?
- Prefetching policy

Performance Considerations

- Latency of fault servicing
- Is fault servicing done in a single step or deferred?
- Is it multithreaded?
- What is the cost breakdown?
- Does the GPU "waste" time waiting?
- Prefetching policy
- Eviction policy

Performance Considerations

- Latency of fault servicing
- Is fault servicing done in a single step or deferred?
- Is it multithreaded?
- What is the cost breakdown?
- Does the GPU "waste" time waiting?
- Prefetching policy
- Eviction policy
- VA translation and multiprocessing

Objectives

Objectives

- Understand UVM driver's semantics - fault buffer, metadata, PTE management

Objectives

- Understand UVM driver's semantics - fault buffer, metadata, PTE management
- Profile execution to find cost breakdown

Objectives

- Understand UVM driver's semantics - fault buffer, metadata, PTE management
- Profile execution to find cost breakdown
- Study various memory access patterns and program performance

Objectives

- Understand UVM driver's semantics - fault buffer, metadata, PTE management
- Profile execution to find cost breakdown
- Study various memory access patterns and program performance
- Study prefetching policy and explain performance variation

Objectives

- Understand UVM driver's semantics - fault buffer, metadata, PTE management
- Profile execution to find cost breakdown
- Study various memory access patterns and program performance
- Study prefetching policy and explain performance variation
- Find optimization possibilities

Profiling GPU Fault Servicing: Batch Characteristics

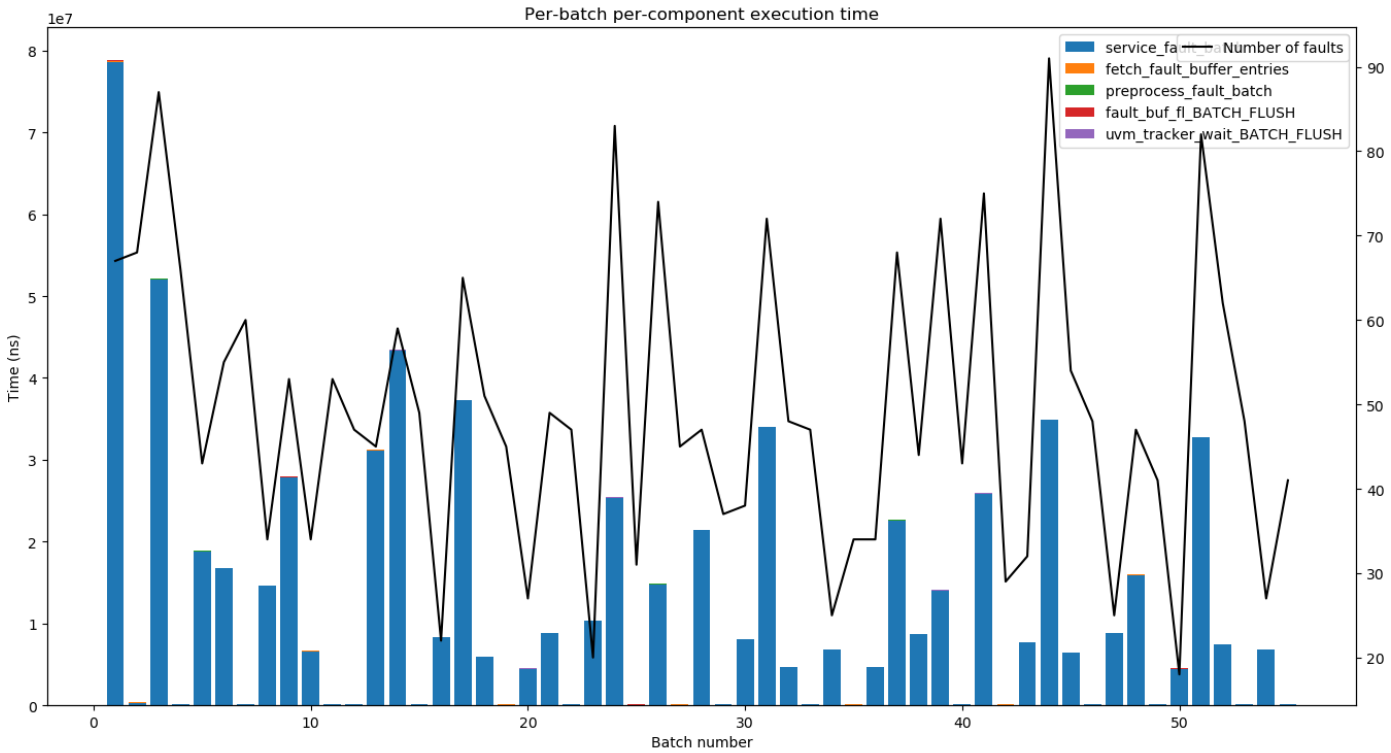


Figure: Fault batch size and service time

- Fault batches have varying sizes because of non-deterministic GPU scheduling
- Usually, a fault batch has faults from a single process

Profiling GPU Fault Servicing: Cost Breakdown

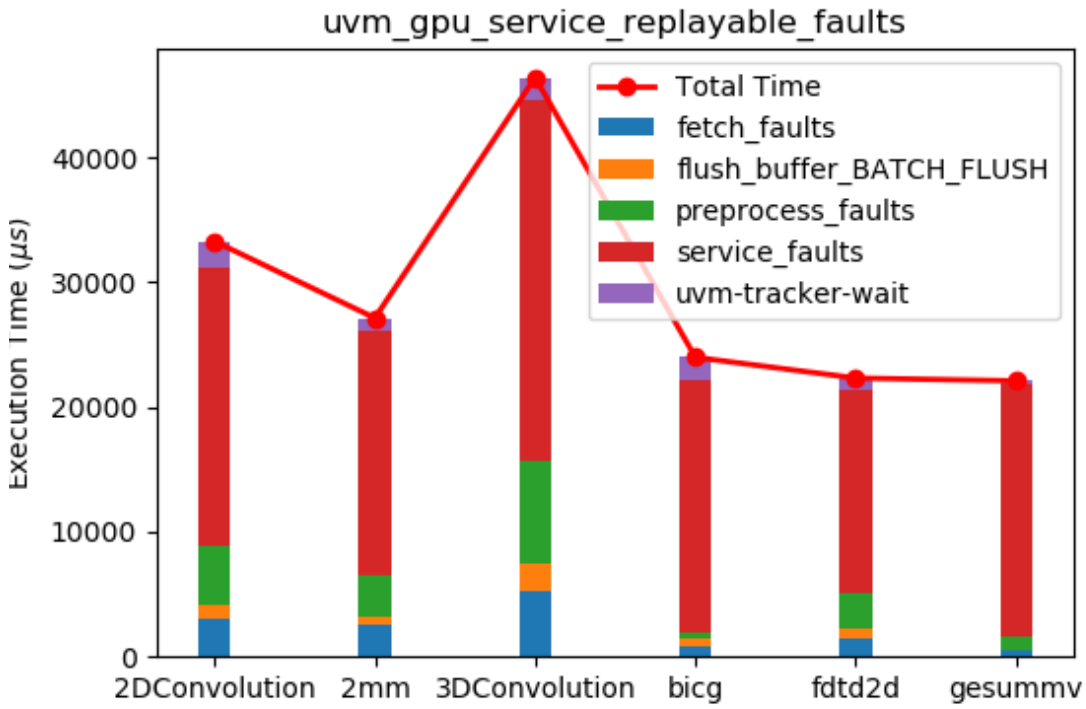


Figure: GPU fault servicing cost breakdown

Profiling GPU Fault Servicing: Cost Breakdown

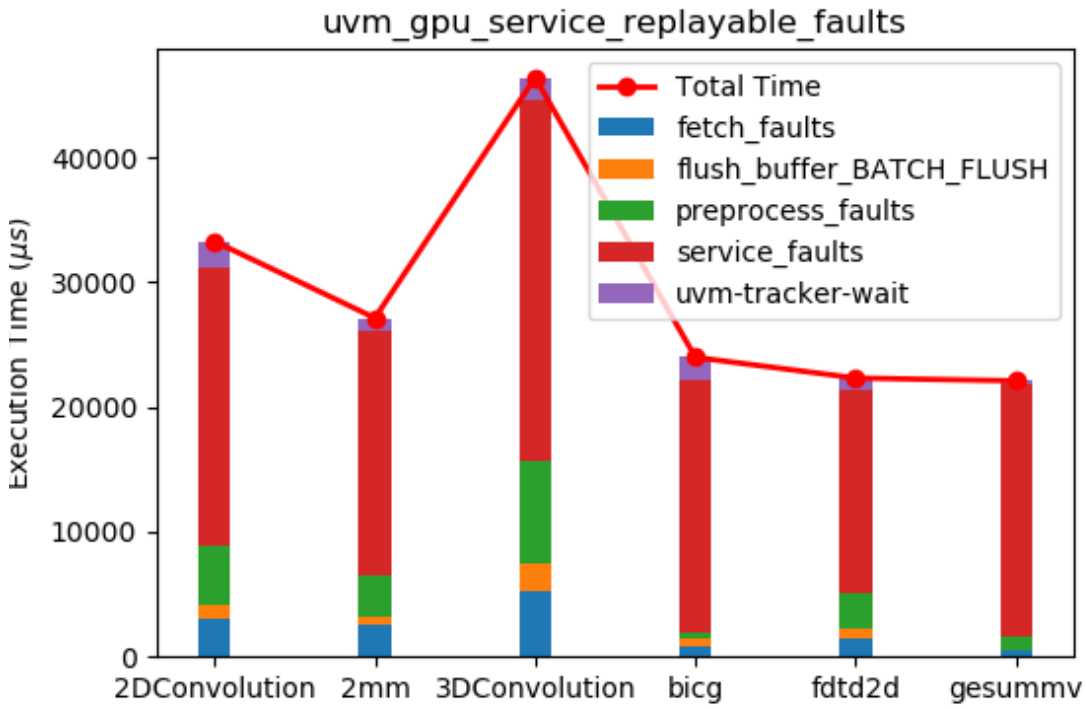


Figure: GPU fault servicing cost breakdown

- Fault servicing is the costliest step
- Page transfers are deferred: hence the `uvm_tracker_wait`
- Breakdown varies with program behaviour

Profiling GPU Fault Servicing: Cost Breakdown

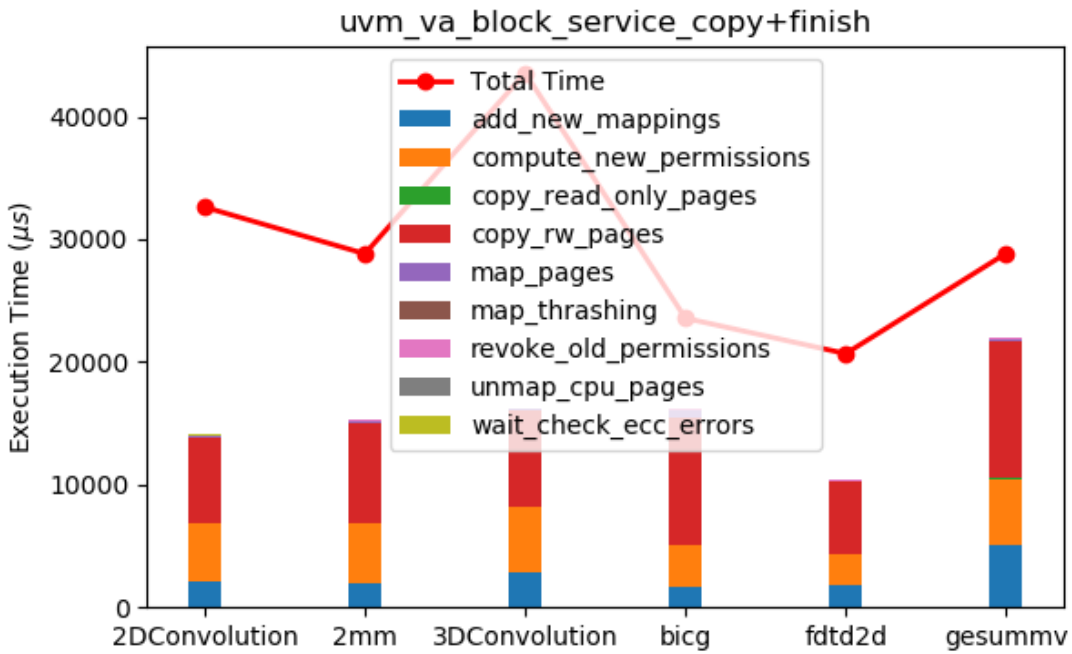


Figure: Subroutines of GPU fault servicing

Profiling GPU Fault Servicing: Cost Breakdown

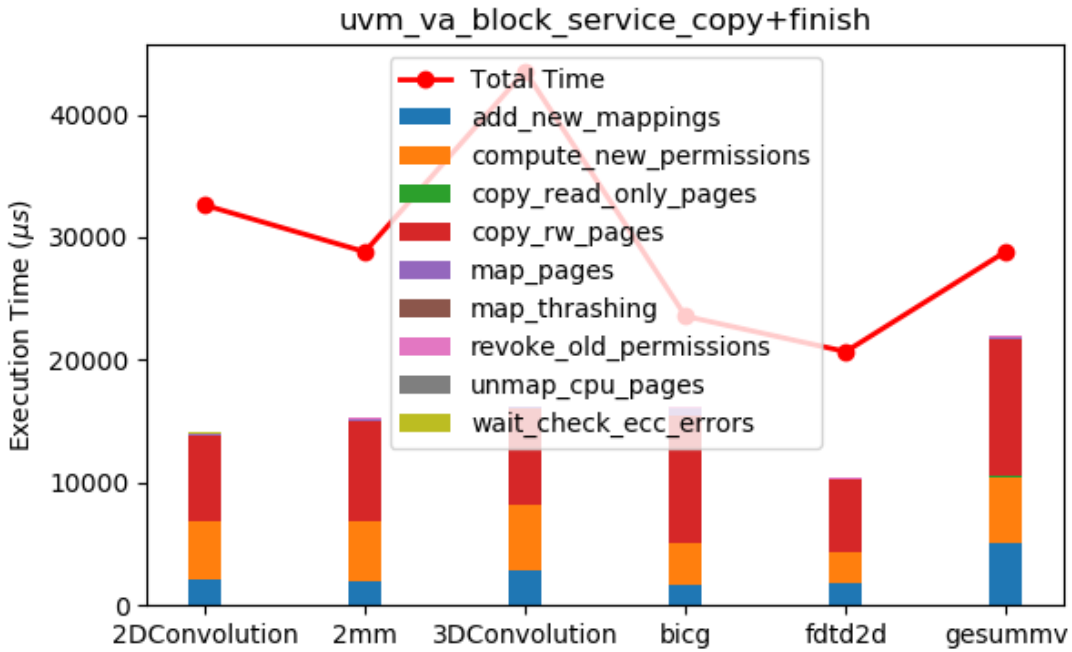


Figure: Subroutines of GPU fault servicing

- For this suite, variation is in fetching and preprocessing faults
- Metadata management cost is comparable to copying pages (at this level)

Profiling GPU Fault Servicing: Copying Pages

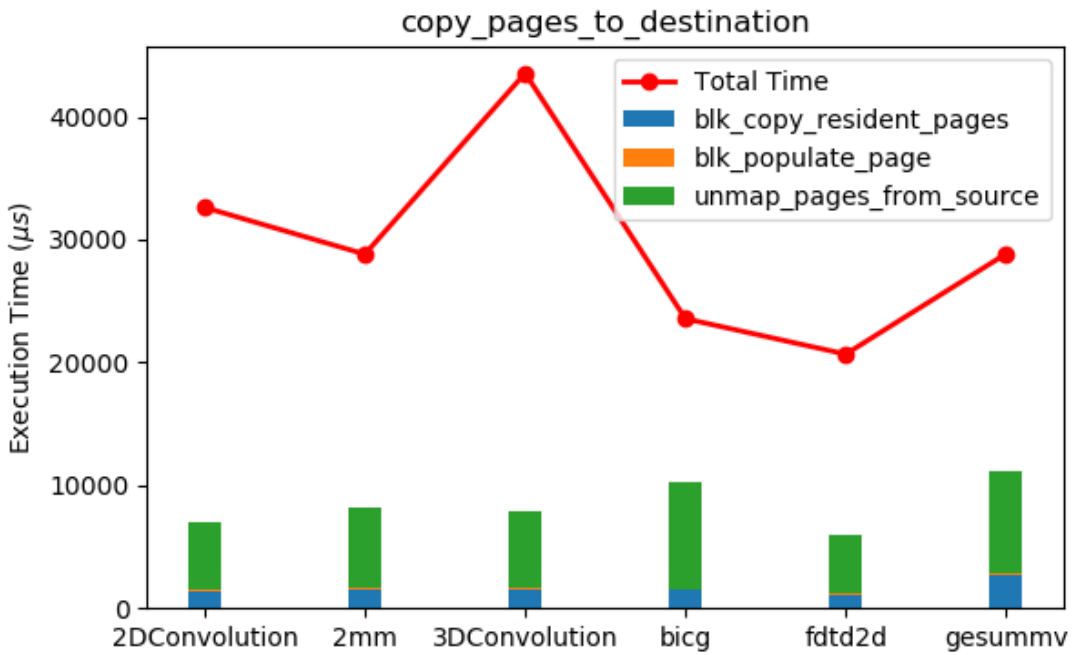


Figure: Steps in copying pages: unmap from source, allocate memory at destination and copy

Profiling GPU Fault Servicing: Copying Pages

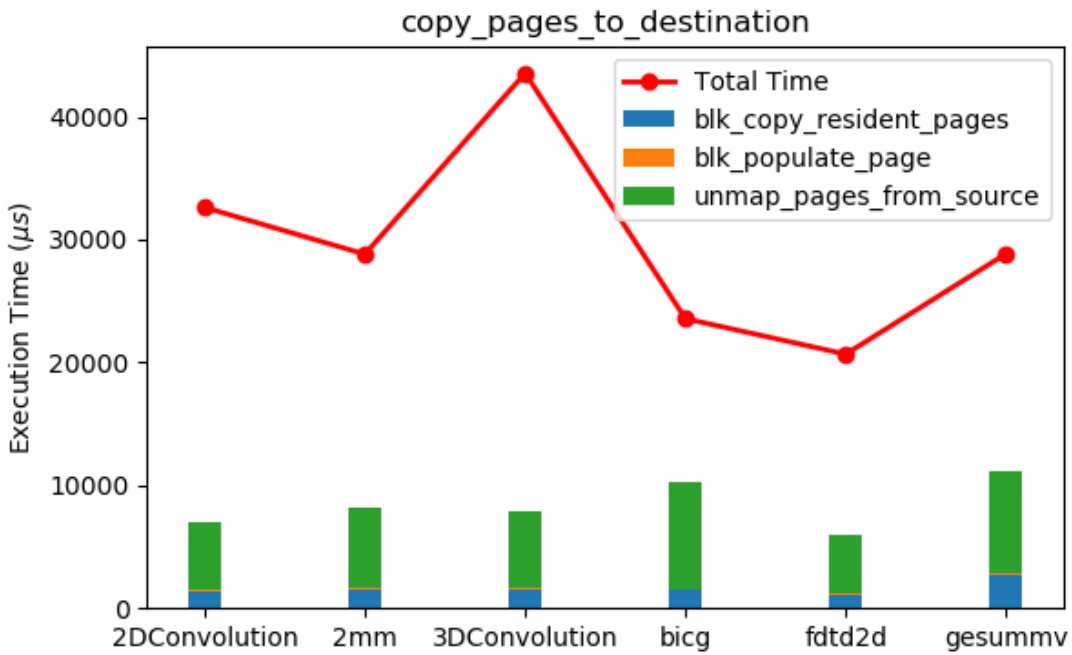


Figure: Steps in copying pages: unmap from source, allocate memory at destination and copy

- Delving deeper, we see that unmapping pages from source is a part of "copying"
- Populating and copying pages are deferred
- Unmapping from CPU cannot be deferred

Profiling GPU Fault Servicing: Unmapping CPU pages

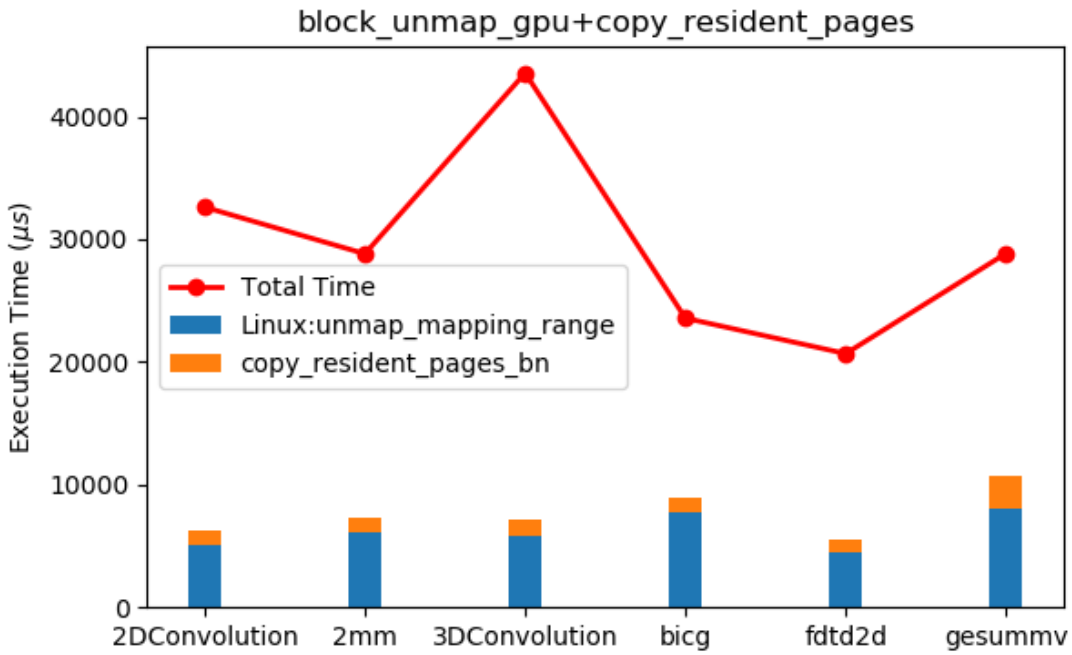


Figure: Further breakdown of previous plot (Page migration cost)

- Upon further analyzing the previous function, we confirm that there are no "hidden" steps in unmapping pages.
- `unmap_mapping_range` (single-threaded) takes about as much time as its caller.

Current Understanding

Current Understanding

- GPU page table operations and page migration is deferred. Cost cannot be estimated directly.

Current Understanding

- GPU page table operations and page migration is deferred. Cost cannot be estimated directly.
- Unmapping pages from CPU has a significant cost. For host-to-device transfers, this cannot be deferred.

Current Understanding

- GPU page table operations and page migration is deferred. Cost cannot be estimated directly.
- Unmapping pages from CPU has a significant cost. For host-to-device transfers, this cannot be deferred.
- Fault servicing is not multithreaded.

Current Understanding

- GPU page table operations and page migration is deferred. Cost cannot be estimated directly.
- Unmapping pages from CPU has a significant cost. For host-to-device transfers, this cannot be deferred.
- Fault servicing is not multithreaded.
- Policy operations are fairly costly. (computing new protections/prefetching candidates)

Current Understanding

- GPU page table operations and page migration is deferred. Cost cannot be estimated directly.
- Unmapping pages from CPU has a significant cost. For host-to-device transfers, this cannot be deferred.
- Fault servicing is not multithreaded.
- Policy operations are fairly costly. (computing new protections/prefetching candidates)
- Host-to-device memory transfers are either hidden or inexpensive

Current Understanding

- GPU page table operations and page migration is deferred. Cost cannot be estimated directly.
- Unmapping pages from CPU has a significant cost. For host-to-device transfers, this cannot be deferred.
- Fault servicing is not multithreaded.
- Policy operations are fairly costly. (computing new protections/prefetching candidates)
- Host-to-device memory transfers are either hidden or inexpensive
- A fault batch usually has faults from a single address space, although the driver code assumes multiple processes.

Current Understanding

- GPU page table operations and page migration is deferred. Cost cannot be estimated directly.
- Unmapping pages from CPU has a significant cost. For host-to-device transfers, this cannot be deferred.
- Fault servicing is not multithreaded.
- Policy operations are fairly costly. (computing new protections/prefetching candidates)
- Host-to-device memory transfers are either hidden or inexpensive
- A fault batch usually has faults from a single address space, although the driver code assumes multiple processes.
- Multiple CUDA contexts (or user processes) cannot run concurrently in the GPU. Kernels from the same thread group can run concurrently, as page tables are shared.

Future Work

Future Work

- Understand how CPU manages GPU page tables (granularity)

Future Work

- Understand how CPU manages GPU page tables (granularity)
- For deferred operations, construct the dependency graph

Future Work

- Understand how CPU manages GPU page tables (granularity)
- For deferred operations, construct the dependency graph
- Determine costs of GPU page table updates and memory transfers

Future Work

- Understand how CPU manages GPU page tables (granularity)
- For deferred operations, construct the dependency graph
- Determine costs of GPU page table updates and memory transfers
- Correlate kernel execution time with fault service latency

Future Work

- Understand how CPU manages GPU page tables (granularity)
- For deferred operations, construct the dependency graph
- Determine costs of GPU page table updates and memory transfers
- Correlate kernel execution time with fault service latency
- Find optimization and multithreading opportunities in fault servicing

Future Work

- Understand how CPU manages GPU page tables (granularity)
- For deferred operations, construct the dependency graph
- Determine costs of GPU page table updates and memory transfers
- Correlate kernel execution time with fault service latency
- Find optimization and multithreading opportunities in fault servicing
- Explain fault service cost in terms of program's memory access patterns

Future Work

- Understand how CPU manages GPU page tables (granularity)
- For deferred operations, construct the dependency graph
- Determine costs of GPU page table updates and memory transfers
- Correlate kernel execution time with fault service latency
- Find optimization and multithreading opportunities in fault servicing
- Explain fault service cost in terms of program's memory access patterns
- Improve current prefetching and eviction policies

Conclusion and Acknowledgements

Work done with Binong Kiri Bey.

Work done under the supervision of Prof Swarnendu Biswas and Prof Debadatta Mishra.

Thank You!