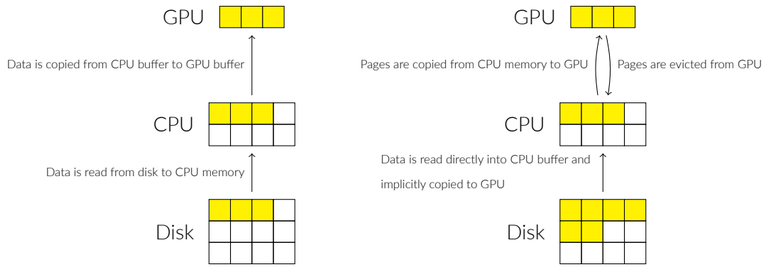
Flexible Prefetching on GPUs with DRAGON-DIRECT

Benjamin Wagley [1] Mehmet E. Belviranli [1] Pak Markthub [2] Bo Wu [1]

[1]Colorado School of Mines, Department of Computer Science [2]NVIDIA Japan

Background and Motivation

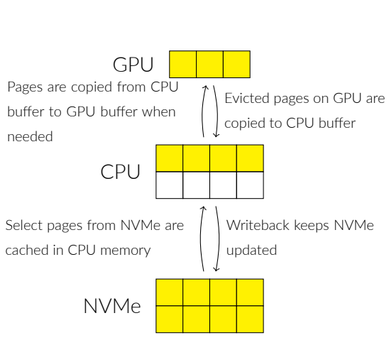
Managing memory in GPU accelerated workloads is a difficult task especially as dataset sizes grow beyond available host memory on an individual system. With GPU accelerated systems ranging from nodes on supercomputers, to data science workstations, working around traditional GPU memory limitations is important. On traditional GPU memory architectures if data is larger than GPU memory, the user must process their data in batches, adding complexity to the task. Architectures like NVIDIA's Unified Virtual Memory (UVM) allow for users to allocate GPU memory across CPU and GPU memory, removing the limitations that data must be smaller than GPU memory. However, UVM still limits data sizes to be smaller than the combined CPU and GPU memory on the system.



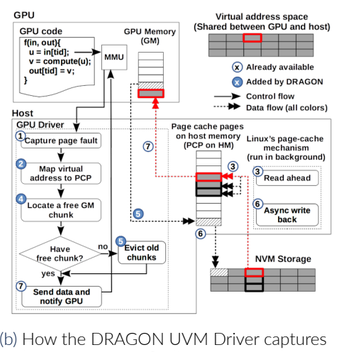
(a) Traditional GPU memory Architecture (b) Unified Virtual Memory Architecture

Figure 1. Architectures from NVIDIA's standard CUDA environment

DRAGON expands on UVM by utilizing high speed NVMe storage and UVM's page-faulting subsystem, in an mmap-like interface that allows terabyte-scale data processing on GPUs. This brings out-of-core processing to any CUDA application. Since DRAGON is implemented at the driver level, it requires no changes to CUDA kernel logic. DRAGON catches page-faults from the UVM driver, and ensures that the required page is copied into GPU memory for use by the application. It will evict unused pages from GPU memory when more GPU memory is needed. By using CPU memory as a cache, DRAGON can utilize basic prefetching systems such as linux's `readahead`. However, it does not allow for more complex prefetching, and bandwidth remains limited due to the fact that pages must be copied into CPU memory before being transferred onto the GPU [1].



(a) DRAGON Memory Architecture



(b) How the DRAGON UVM Driver captures and processes page faults

Figure 2. DRAGON Internals

A Different Data Paradigm

As a byproduct of DRAGON, data can be stored directly in binary files on the NVMe storage, which is then mapped directly into GPU memory when reading. For commonly read data, we can amortize dataset I/O to constant time, once the dataset is processed into the desired form. This is similar to using `mmap()` on preprocessed data for I/O.

Contributions

- We present DRAGON-DIRECT, a novel expansion of DRAGON that supports direct data migration between NVMe storage and GPU memory, with a flexible user-controlled prefetching system.
- We integrate DRAGON-DIRECT into NVIDIA's RAPIDS data science libraries.
- We showcase a novel approach utilizing DRAGON-DIRECT's prefetching system for creation of mini-batches in machine learning workloads.

DRAGON-DIRECT

Utilizing *Direct Memory Access* (DMA), DRAGON-DIRECT builds on DRAGON by allowing direct mapping of NVMe storage into GPU memory address space, bypassing host memory. Like DRAGON it is implemented at the driver level and does not require modification of CUDA kernels. Pages can still be cached in CPU memory if desired. This not only reduces overhead in transferring data from NVMe to GPU, but also allows for more advanced user-controlled prefetching.

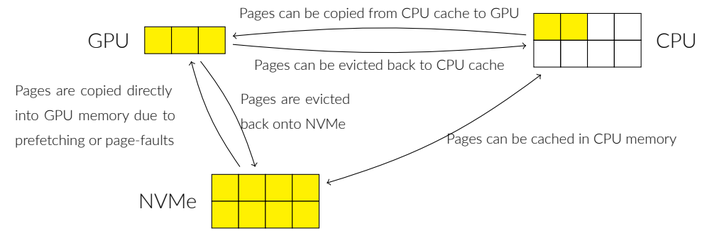


Figure 3. DRAGON-DIRECT Memory Architecture

Prefetching with DRAGON-DIRECT

We can see that without prefetching, DRAGON-DIRECT would perform poorly on data larger than GPU memory due to excessive thrashing. DRAGON-DIRECT provides a basic prefetching interface to both CPU and GPU code that can be adapted to an applications needs with page-level granularity. To do this, DRAGON-DIRECT provides 3 basic page-level operations:

- *Checking residency:* This reports if a page is resident within GPU memory.
- *Fetch-and-pin:* If a page is not in GPU memory, it will be brought into GPU memory, and pinned (so that it cannot be evicted).
- *Unpin:* Allows a page to then be evicted, as it is no longer needed.

Integration with RAPIDS

NVIDIA's RAPIDS [2] is a suite of data science and analytics python libraries accelerated with CUDA, that aim to provide similar experiences to common libraries such as pandas, sklearn and networkx. We chose to integrate DRAGON-DIRECT into the RAPIDS suite to showcase the flexibility inherent with integrating DRAGON-DIRECT into pre-existing workflows, as well as to showcase DRAGON-DIRECT's real world performance. Our current integration lies mainly in the RMM (Rapids Memory Manager) library, and is accompanied by additional I/O calls within the cuDF (Data-Frame) library as well as basic integration into the cuML (Machine Learning) library. Our integration has *no changes to existing CUDA kernels within RAPIDS* – the only changes required to use DRAGON are in CPU code.

This means that for the end user, using DRAGON in a project is as simple as:

```
import cudf
import rmm
#Set memory resource to DRAGON
rmm.mr.set_current_device_resource(rmm.mr.DragonMemoryResource())
```

Flexible, User Controlled Prefetching Accelerating Mini-Batches

The prefetching that DRAGON-DIRECT provides allows for a novel new workflow in mini-batch creation. We modify cuML's Mini-Batch SGD Classifier to utilize this workflow. This new workflow allows for easier out-of-core computation with a reduction of processing time spent creating the mini-batches. We used DRAGON-DIRECT's prefetching capabilities to implement an index-based prefetcher for arrays. Our mini-batch training algorithm can be seen in figures 4 and 5.

We can see that this algorithm allows us to create mini-batches with dataset sizes well beyond GPU memory or even CPU memory. It has the advantage that data for the next mini-batch is being prefetched onto the GPU while training on the current mini-batch. Ideally, the next mini-batch will be resident in memory before it is needed, reducing processing overhead inherent in traditional out-of-core processing.

$T \leftarrow$ Training Dataset;
$N \leftarrow$ Number of Mini-Batches;
$M \leftarrow |T|/N$;
$E \leftarrow$ Number of Epochs;
prefetch(X, y) $\leftarrow$ Prefetches indices y from array X;
for $e \in [0, E)$ do
  $I \leftarrow [0, |T|)$;
  randomize(I);
  $m \leftarrow 0$;
  //Immediately Start Prefetching First Mini-Batch;
  prefetch(T, $I[m, (m+1) \times M]$);
  for $n \in [0, M)$ do
    //Copy current mini-batch into training tensor;
    $t \leftarrow T[I[m, (m+1) \times M]]$;
    $m \leftarrow m + 1$;
    //Prefetch the next mini-batch, before we start computing the current one;
    prefetch(T, $I[m, (m+1) \times M]$);
    //Train on current mini-batch;
    train(t);
  end

Figure 4. Pseudocode for our mini-batch prefetching
(Green highlights prefetching the next mini-batch prior to training on the current mini-batch)

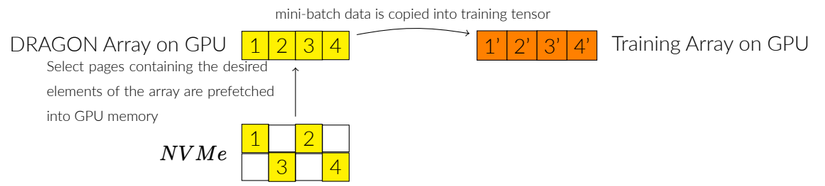


Figure 5. Prefetching for mini-batches using DRAGON-DIRECT

Conclusion

We are still actively performing experiments, and do not currently have data to reflect our implementations performance. We expect to see similar kernel execution time to UVM since DRAGON-DIRECT is built into the UVM driver when optimally prefetching. We can see that DRAGON-DIRECT provides a powerful new memory system for CUDA programs that allows for terabyte scale out-of-core processing, without requiring changes to pre-existing kernels. DRAGON-DIRECT allows for data science workstations and supercomputer nodes with unprecedented data scale, through mapping NVMe storage into GPU memory.

References

[1] Pak Markthub, Mehmet E Belviranli, Seyong Lee, Jeffrey S Vetter, and Satoshi Matsuoka.
Dragon: breaking gpu memory capacity limits with direct nvm access.
In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 414–426. IEEE, 2018.

[2] Nvidia rapids.
http://rapids.ai.

DRAGON:

NVIDIA RAPIDS: