



# DRAGON: Breaking GPU Memory Capacity Limits with Direct NVM Access

Pak Markthub<sup>T</sup>, Mehmet E. Belviranli<sup>O</sup>, Seyong Lee<sup>O</sup>,  
Jeffrey S. Vetter<sup>O</sup>, and Satoshi Matusoka<sup>R,T</sup>



<sup>T</sup>Tokyo Institute of Technology

<sup>O</sup>Oak Ridge National Laboratory

<sup>R</sup>RIKEN Center for Computational Science





# In a nutshell...

- GPUs are largely used in HPC and ML
  - **Workload sizes** and user productivity have been **limited by GPU memory capacity**
- Meanwhile, memory systems are evolving...
  - NVMs provide larger capacities at lower costs & power compared to Host and GPU mem
- In this study, we propose **DRAGON**:
  - **Enables GPU** kernels to **directly access NVMs**
  - **Transparently provides massive memory to GPUs**
  - It is **open source**: CUDA driver extension + User-level API

Available at <https://github.com/pakmarkthub/dragon>



# Motivation



# Memory systems started diversifying...

## • Architectures

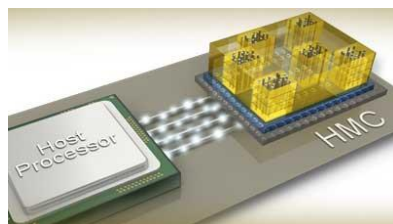
- HMC, HBM/2/3, LPDDR4, GDDR5X, WIDEIO2, etc
- 2.5D, 3D Stacking

## • Configurations

- Unified Memory
- Scratchpads
- Write through, write back, etc
- Consistency and coherence protocols
- Virtual v. Physical, paging strategies

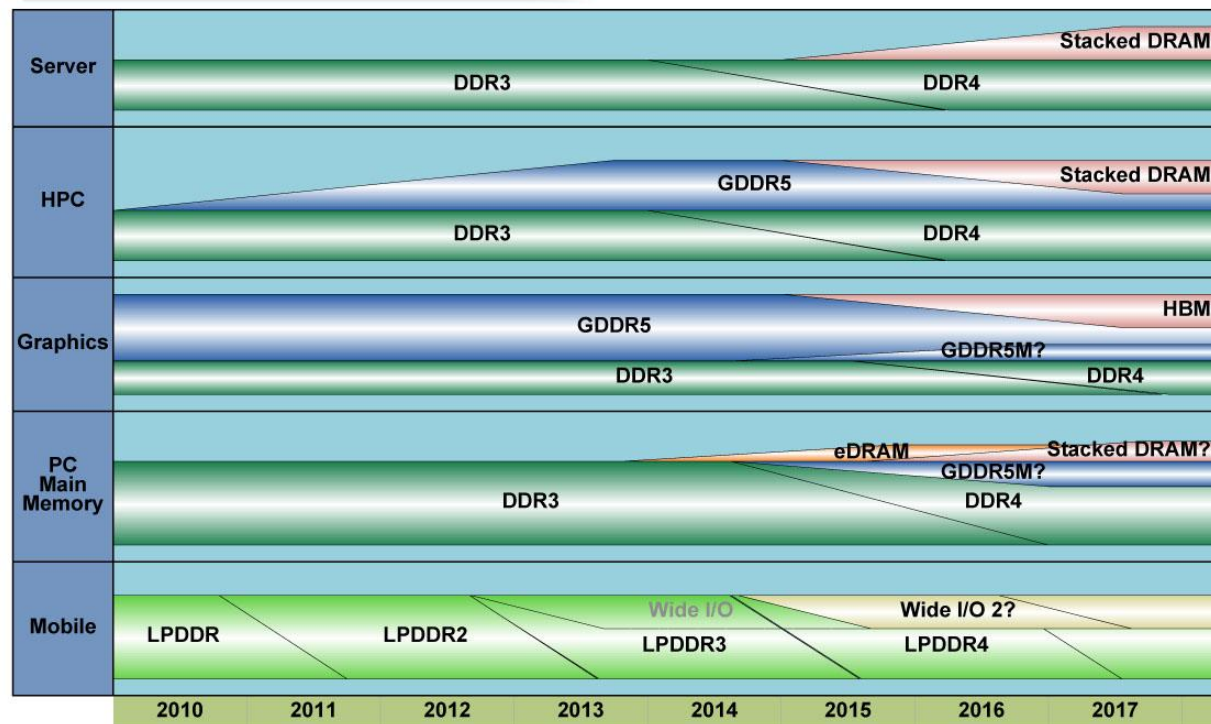
## • New devices

- ReRAM, PCRAM, STT-MRAM, 3DXpoint



[https://www.micron.com/~media/track-2-images/content-images/content\\_image\\_hmc.jpg?la=en](https://www.micron.com/~media/track-2-images/content-images/content_image_hmc.jpg?la=en)

DRAM Transition



Copyright (c) 2014 Hiroshige Goto All rights reserved.

[http://gigglehd.com/zbx/files/attach/images/1404665/988/406/011/788d3ba1967e2db3817d259d2e83c88e\\_1.jp](http://gigglehd.com/zbx/files/attach/images/1404665/988/406/011/788d3ba1967e2db3817d259d2e83c88e_1.jp)

	SRAM	DRAM	eDRAM	2D NAND Flash	3D NAND Flash	PCRAM	STTMRAM	2D ReRAM	3D ReRAM
Data Retention	N	N	N	Y	Y	Y	Y	Y	Y
Cell Size (F <sup>2</sup> )	50-200	4-6	19-26	2-5	<1	4-10	8-40	4	<1
Minimum F demonstrated (nm)	14	25	22	16	64	20	28	27	24
Read Time (ns)	<1	30	5	10 <sup>6</sup>	10 <sup>6</sup>	10-50	3-10	10-50	10-50
Write Time (ns)	<1	50	5	10 <sup>6</sup>	10 <sup>6</sup>	100-300	3-10	10-50	10-50
Number of Rewrites	10 <sup>6</sup>	10 <sup>6</sup>	10 <sup>6</sup>	10 <sup>3</sup> -10 <sup>4</sup>	10 <sup>3</sup> -10 <sup>4</sup>	10 <sup>3</sup> -10 <sup>4</sup>	10 <sup>3</sup> -10 <sup>4</sup>	10 <sup>3</sup> -10 <sup>4</sup>	10 <sup>3</sup> -10 <sup>4</sup>
Read Power	Low	Low	Low	High	High	Low	Medium	Medium	Medium
Write Power	Low	Low	Low	High	High	High	Medium	Medium	Medium
Power (other than R/W)	Leakage	Refresh	Refresh	None	None	None	None	Sneak	Sneak
Maturity	High	High	High	Low	Low	Low	Low	Low	Low

J.S. Vetter and S. Mittal, "Opportunities for Nonvolatile Memory Systems in Extreme-Scale High Performance Computing," CISE, 17(2):73-82, 2015.

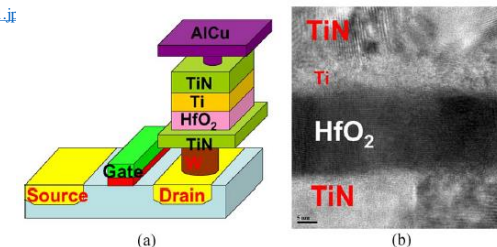
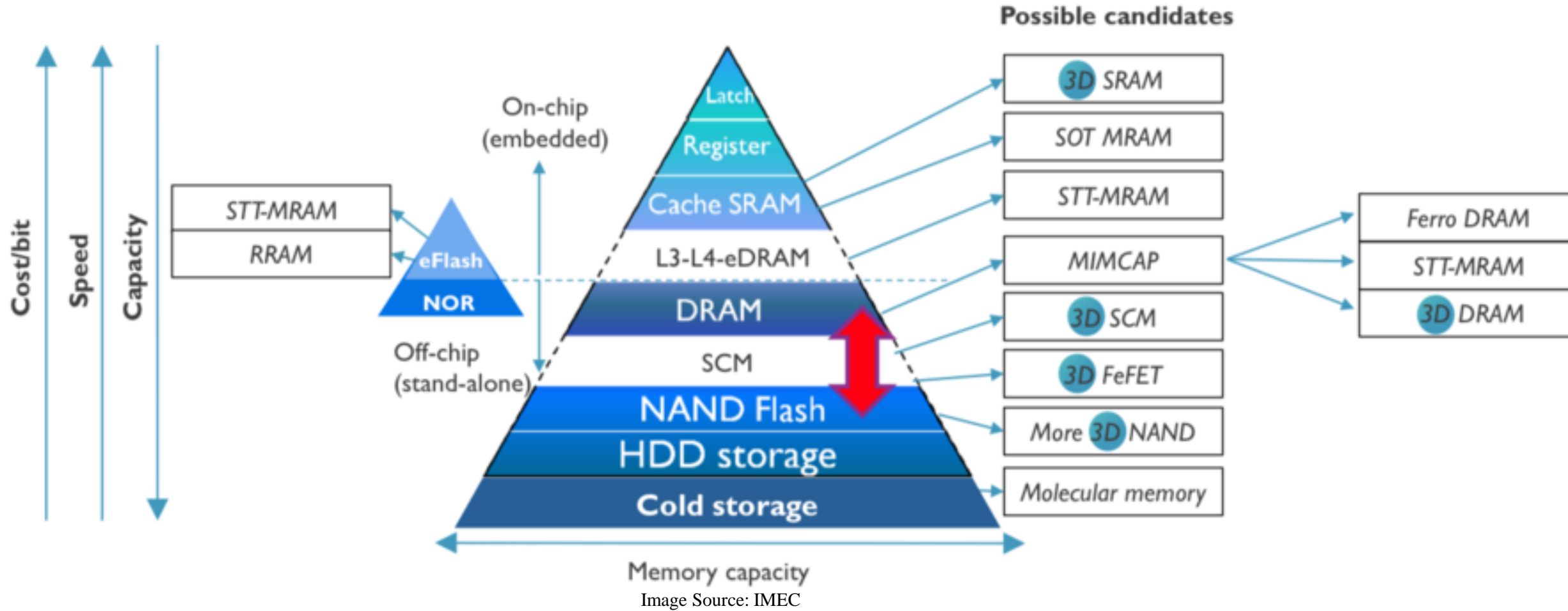


Fig. 4. (a) A typical 1T1R structure of ReRAM with HfO<sub>2</sub>; (b) HR-TEM image of the TiN/Ti/HfO<sub>2</sub>/TiN stacked layer; the thickness of the HfO<sub>2</sub> is 20 nm.



# NVM are moving up in memory hierarchy



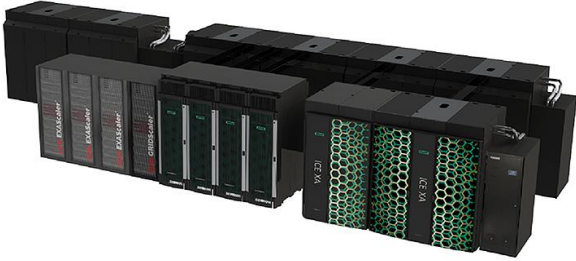
**Non-Volatile, Large Capacity, Low Cost, Low Power, High Speed**





# GPUs are widely adopted

- Employ in supercomputers, clusters, and Clouds



**TSUBAME3.0 @ TITECH**  
540 nodes; 4 P100 per node



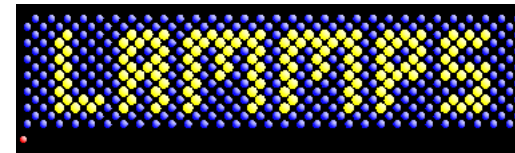
**Summit @ ORNL**  
4,608 nodes; 6 V100 per node



Google Cloud

- Use by thousands of applications

**Caffe**



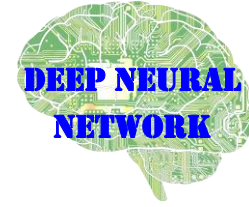
**GROMACS**  
FAST. FLEXIBLE. FREE.




















# Problem: GPU memory is too small!!

- **Workload sizes grow** larger than GPU and host memory
- **Complex GPU algorithms** to handle out-of-core processing



	On-chip (GPU Memory)	Off-chip (Host Memory)	Off-chip (I/O)
Programming Complexity	Standard (cudaMalloc) 	Standard (Unified Memory) 	High (I/O interfaces, pipelines, overlapping...) 
Implementation technique	Applicable broadly 	Applicable broadly 	Algorithm-specific 
Performance	High 	Lower (due to PCI-e) 	Lowest (PCI-e + I/O) 
Problem size	Must fit in GPU memory 	Must fit in System memory 	Unlimited 
Data movement	Host $\leftrightarrow$ GPU; GPU kernel has direct access 	On-demand & implicit data copy, HW paging 	Manual buffer management via I/O and CUDA calls 

**Can we use NVMs for GPUs without incurring application complexity while maintaining reasonable performance?**

# Related Work

- **Out-of-core Processing for GPU**
  - Many research papers and implementations
  - Efficient but **algorithm-specific**
- **NVIDIA's Unified Memory (UM)**
  - On-demand paging for GPUs
  - **Cannot go beyond host memory** due to page pinning
  - **Separate from storage space** → Need fread/fwrite!!!
- **Other Hardware- and Software-based Approaches**
  - Ex. GPUfs, ActivePointers
  - Prior work is mostly **obsoleted by UM**
  - Many suggested **impractical HW modifications**

## A Framework for Out of Me

Khairul Kabir<sup>4</sup>, Azzam Haidar<sup>1</sup>,  
Aurelien Bouteiller<sup>1</sup>, and J

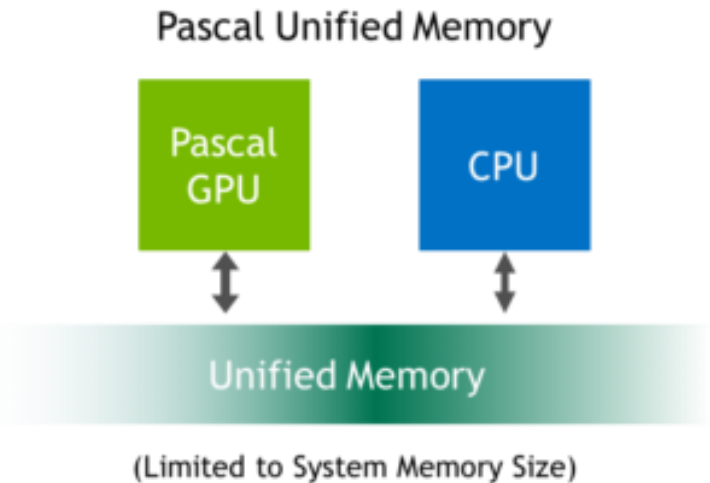
2015 International Conference on Parallel Architecture and Compilation

## Scalable SIMD-Efficient Graph Processing on GPUs

## Realizing Out-of-Core Stencil Computations using Multi-Tier Memory Hierarchy on GPGPU Clusters

Toshio Endo  
Global Science Information and Computing Center,  
Tokyo Institute of Technology, Japan  
Email: [endo@is.titech.ac.jp](mailto:endo@is.titech.ac.jp)

mi N. Bhuyan  
Department  
CA, USA  
r.edu



<https://devblogs.nvidia.com/beyond-gpu-memory-limits-unified-memory-pascal/>







# Solution & Contributions

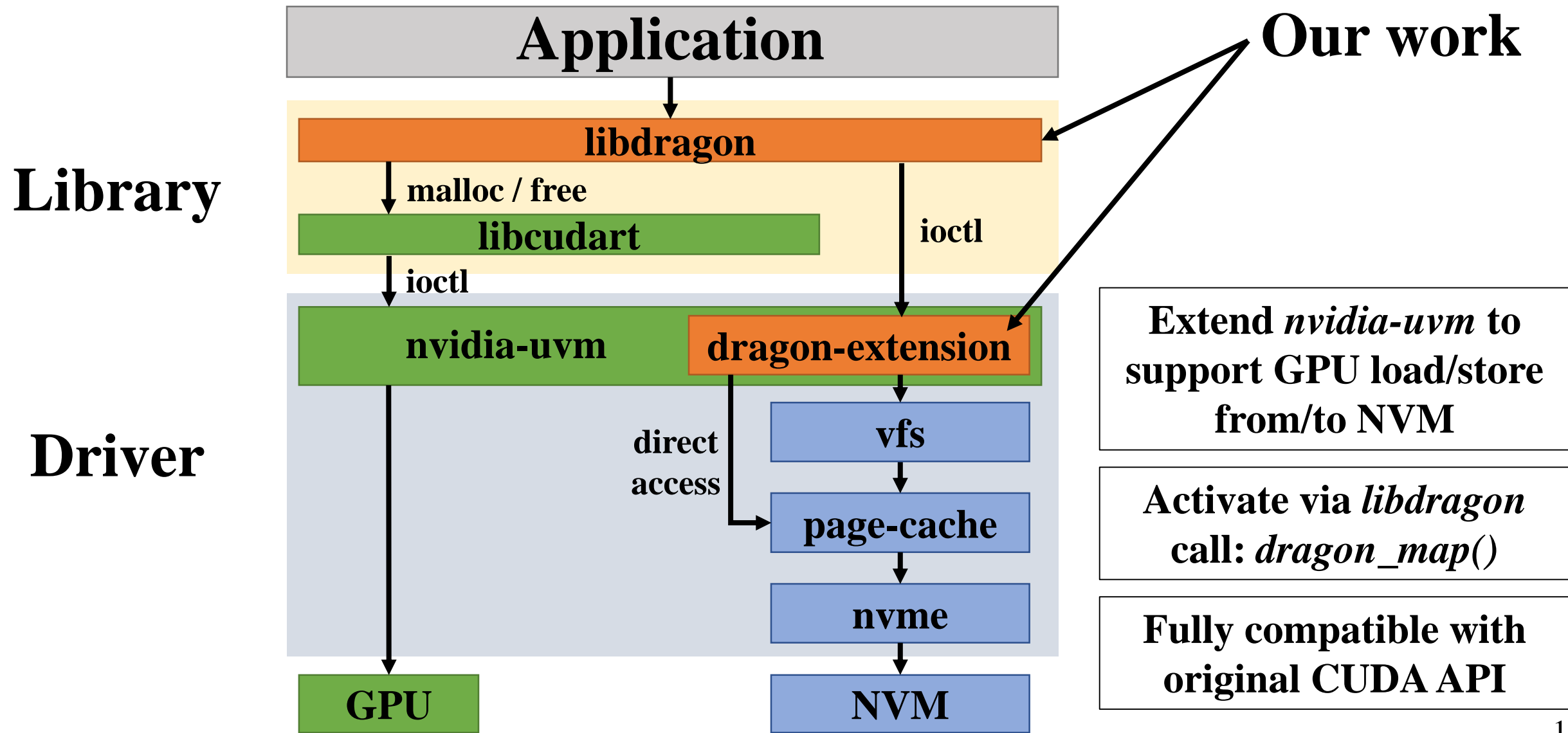
- **DRAGON: Direct Resource Access for GPU over NVM**
  - Enable “mmap” from NVM to GPU memory space
  - **Unified GPU memory, host memory, and NVM spaces** with strong and weak coherency
  - Utilize GPU HW page-faulting with our customized *nvidia-uvm* driver
  - **Overlap GPU computation with data transfer by nature** (2.3x speedup over UM + IO)
- **Key Contributions:**
  - **Transparently extends GPU memory** space to NVM devices
    - Lead to virtually unlimited GPU memory capacity
    - **No need to modify GPU kernels**
  - **Eliminate manual buffer management** on data larger than the system memory
  - Present **NVM-optimized data access pattern** types to decrease IO overhead
  - Evaluate functionality and performance on scientific and ML applications



**DRAGON – New solution to  
provide transparent access to  
massive NVM capacity**

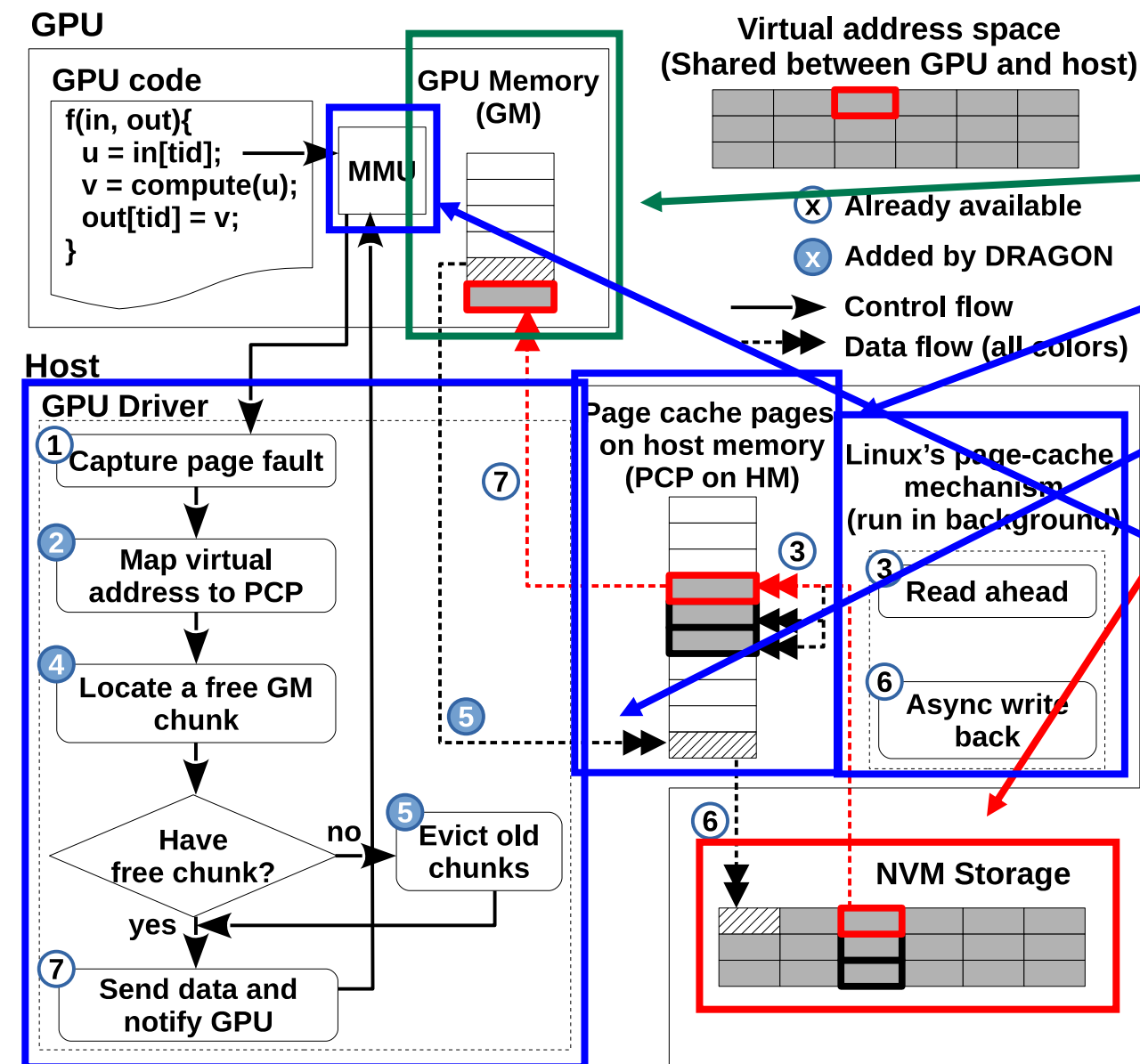


# DRAGON: Overview





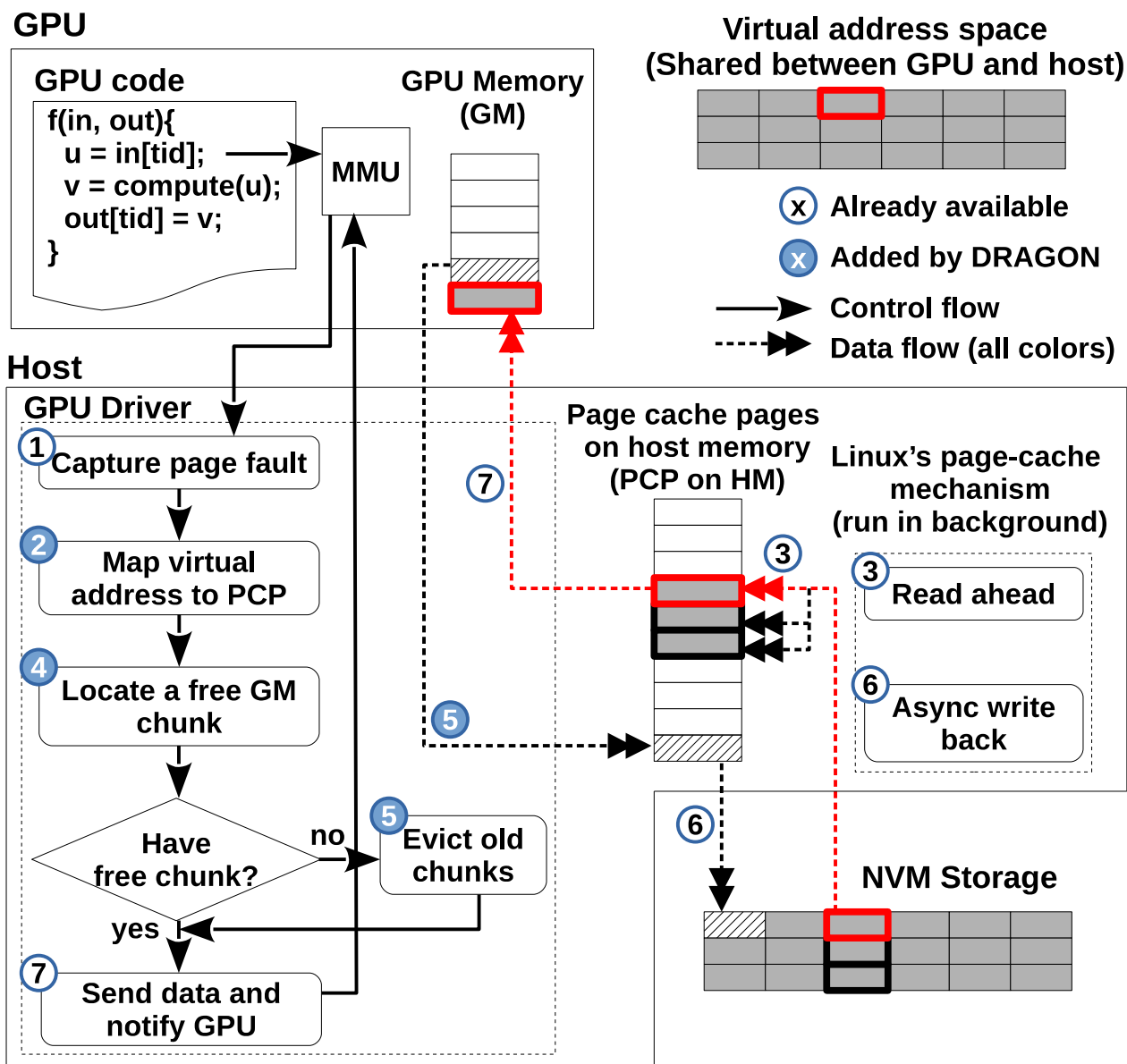
# DRAGON Operations: Key Components



- **Three memory spaces:**
  - **GPU Mem (GM)** as 1<sup>st</sup> level cache
  - **Host Mem (HM)** as 2<sup>nd</sup> level cache
  - **NVM** as primary storage
- **Modified GPU driver**
  - Manage data movement & coherency
- **GPU MMU with HW Page Fault**
  - Manage GPU virtual memory mapping
- **Page cache**
  - Buffer & accelerate data access



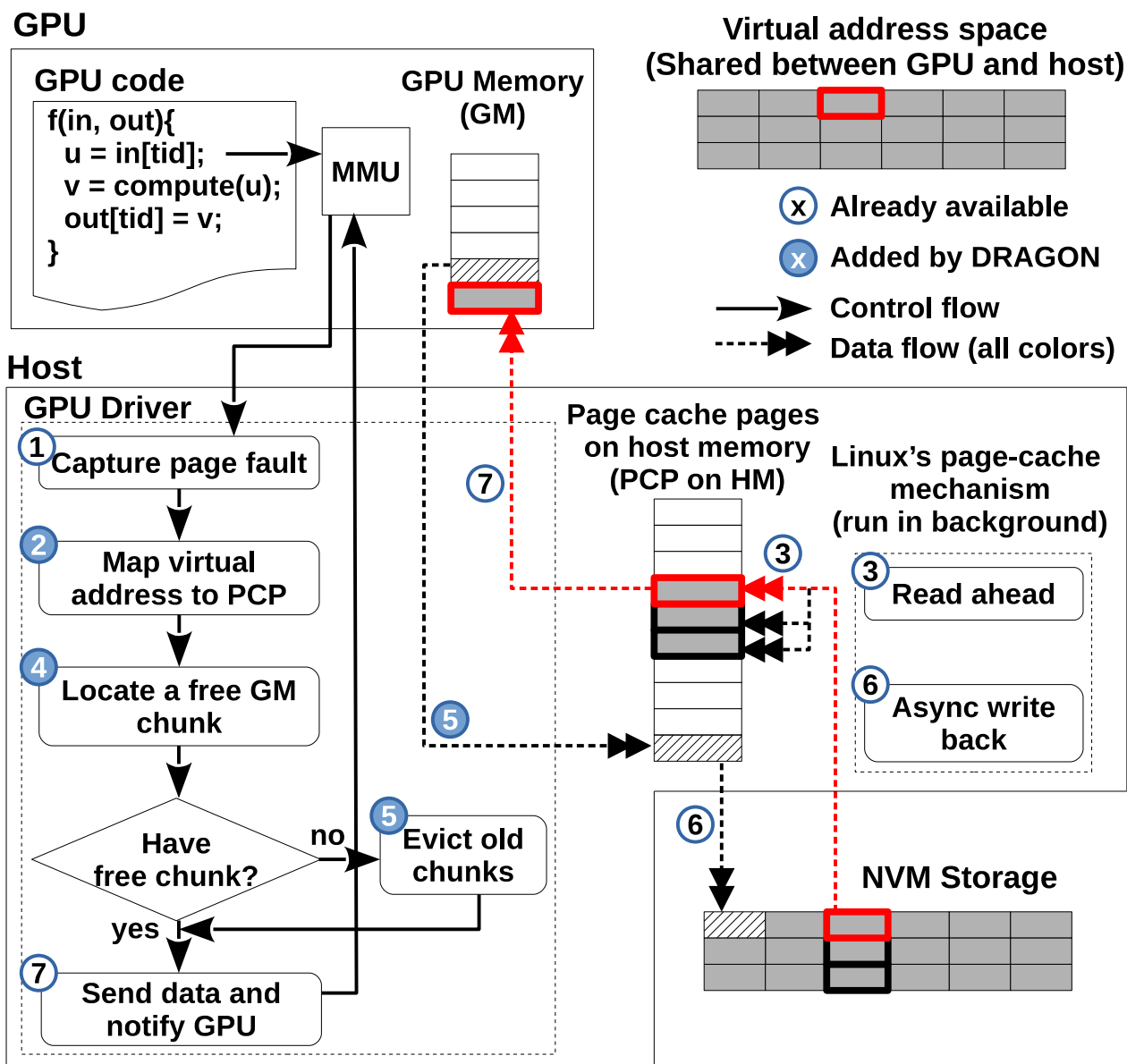
# How Data Move: NVM → GPU



- The GPU load operation triggers **GPU page fault**.
- DRAGON driver extension transfers data NVM → HM → GM.
- Next data is **prefetched on HM via page-cache read-ahead**.
  - If cache hits on HM, data transfer becomes HM → GM.
  - Prefetching occurs **while GPU is computing**.
- **Adaptive HM → GM granularity**
  - 4 KiB, 64 KiB, 2 MiB
  - If cache hits on GM, no data transfer.



# How Data Move: GPU → NVM



- The GPU store operation writes data on Global Memory.
- If no free GPU chunk, GM → HM on a victim chunk.
  - Eviction policy follows UM.
  - V100 does better than P100 due to HW counter.
- While GPU is computing, HM → NVM by page-cache write-back.
  - Eviction policy follows page-cache.

**Data is kept on the fastest-possible memory**



# Optimizations

```
IN = read(...);  
tmp = compute0(IN);  
OUT = compute1(tmp);  
write(OUT, ...);
```



- **Intermediate Data (Temporary Workspace)**
  - Keep the data on the fastest-possible memory (GM > HM > NVM)
  - Disable automatic write-back and flushing
- **Read-Only and Write-Only Data**
  - One-directional data movement → No write out and read in respectively
- **Eviction Policies**
  - **Intermediate Data:** Least Recent Use (LRU)
  - **Input/Output Data:** Follow UM and Page Cache
- **Two-level Prefetching**
  - **Host to GPU:** 4 KiB, 64 KiB, 2 MiB
  - **NVM to Host:** Follow Page Cache Read-Ahead



# DRAGON: API and Integration

Out-of-Core using CUDA

```
// Allocate host & device memory
h_buf = malloc(size);
cudaMalloc(&g_buf, size);

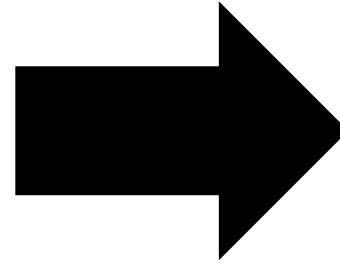
while() { // go over all chunks
    // Read-in data
    f = fopen(filepath, "r");
    fread(h_buf, size, 1, f);

    // H2D Transfer
    cudaMemcpy(g_buf, h_buf, H2D);

    // GPU compute
    compute_on_gpu(g_buf);

    // Transfer back to host
    cudaMemcpy(h_buf, g_buf, D2H);
    compute_on_host(h_buf);

    // Write out result
    fwrite(h_buf, size, 1, f);
}
```



## DRAGON

```
// mmap data to host and GPU
dragon_map(filepath, size,
    D_READ | D_WRITE, &g_buf);

// Accessible on both host and GPU
compute_on_gpu(g_buf);
compute_on_host(g_buf);

// Implicitly called when program exits
dragon_sync(g_buf);
dragon_unmap(g_buf);
```

## Notes

- Similar to NVIDIA's Unified Memory (UM)
- Enable access to large memory on NVM
  - **UM is limited by host memory**



# **Evaluation on Scientific and ML workloads**



# Evaluation

## Data movement methods

1. cudaMemcpy + fread/fwrite (original)
2. cudaHostRegister + mmap
- 3. UM-P + fread/fwrite (baseline)**
- 4. DRAGON**

## Benchmark applications

Application	Category	Vol:NonVol
backprop	Unstructured Grid	1:1
binomialOptions	Linear Algebra	0:1
BlackScholes	Linear Algebra	0:1
hotspot	Structured Grid	0:1
lavaMD	N-Body	0:1
pathfinder	Dynamic Programming	0:1
srad_v2	Structured Grid	5:1
vectorAdd	Dense Linear Algebra	0:1

## Environment

CPU	Dual 12-core Intel Xeon E5
Memory	<b>64 GiB</b> DDR 4
GPU	One NVIDIA P100 ( <b>12 GiB</b> ) PCIe
NVM	<b>2.4TB</b> Micron 9100 HHHL U.2 PCIe NVMe formatted with ext4
Connection	PCIe gen.3 x16
OS	CentOS7 Kernel 3.10.0-693.5.2.el7.x86_64
CUDA	V9.0 with driver 384.81

**No change to the CUDA kernels**

Measured the entire execution time including reading in input from and writing out result to files





# Case Study: Caffe

## Data movement methods

1. cudaMemcpy + fread/fwrite (Default)
- 2. UM-P + fread/fwrite**
3. CPU 4 threads (ATLAS)
4. CPU all cores (OpenBLAS OMP)
- 5. DRAGON (our solution)**

## Environment

CPU	Dual 12-core Intel Xeon E5
Memory	<b>24 GiB</b> DDR 4
GPU	One NVIDIA P100 ( <b>12 GiB</b> ) PCIe
NVM	<b>2.4TB</b> Micron 9100 HHHL U.2 PCIe NVMe formatted with ext4
Connection	PCIe gen.3 x16
OS	CentOS7 Kernel 3.10.0-693.5.2.el7.x86_64
CUDA	V9.0 with driver 384.81

## Methodology

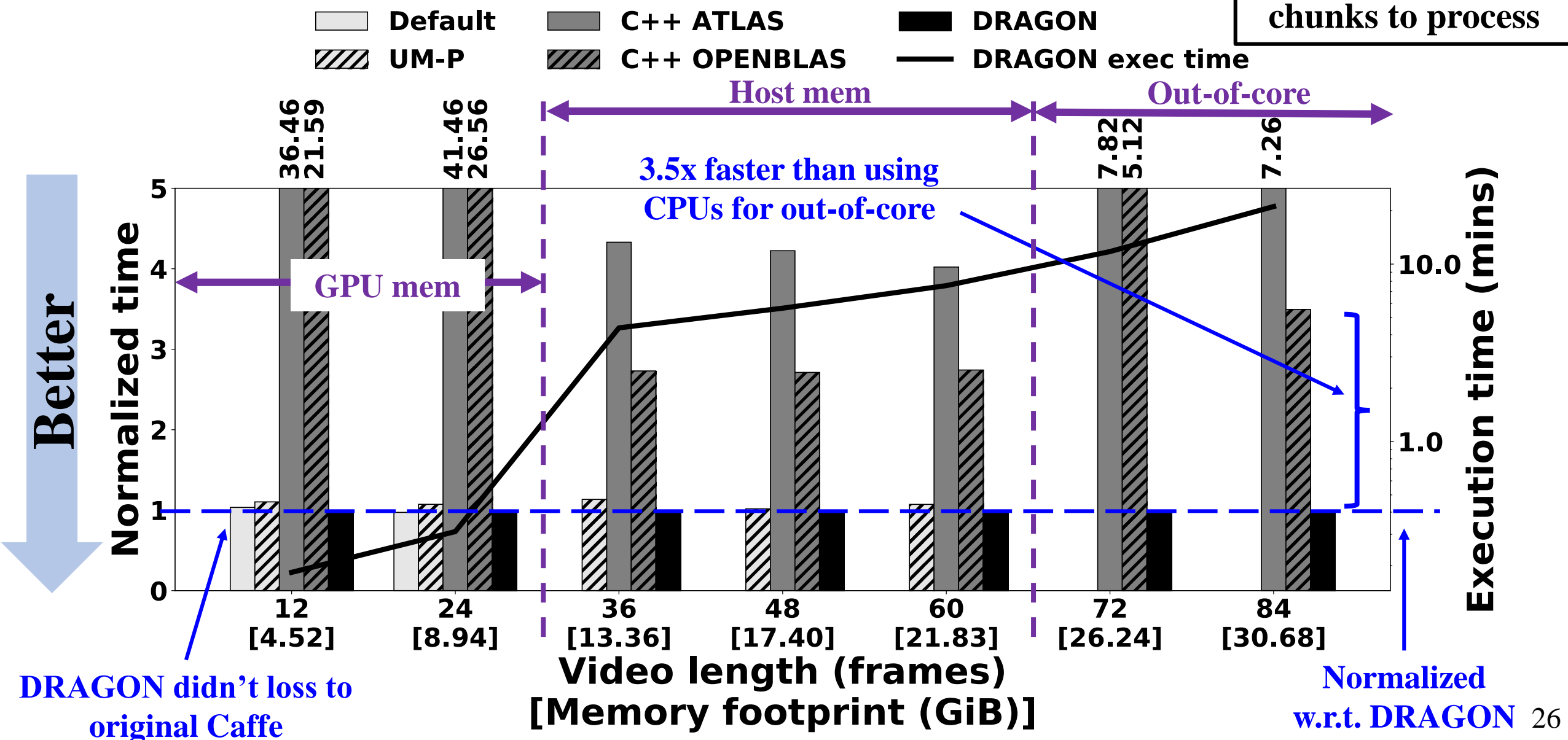
- Modified Caffe to support UM-P and DRAGON
- Changed only data movement methods on host; **No change to CUDA kernels**
- Varied memory footprint size by varying the problem size and input parameters
- Measured execution time and normalized w.r.t. DRAGON



(video action recognition; 3D Conv; batch size 1; ran 30 batches)

# Case Study: C3D-UCF101Net (Caffe)

Enable complex algorithms that require large data chunks to process





# Conclusion

## DRAGON:

- **Efficiently and transparently maps files on NVMs to GPUs**
  - No need to modify CUDA kernels
- **Enables transparent off-chip memory access for large data** with no need to do manual buffering nor modify GPU algorithms
- **Significantly improves application execution times** with
  - **Direct page-cache access** and intermediate data handling optimization
  - **Implicitly overlapping computation with data transfer** with read-ahead and write-back

Available at <https://github.com/pakmarkthub/dragon>



# Acknowledgement

- This research was supported in part by an appointment to the Oak Ridge National Laboratory ASTRO Program, sponsored by the U.S. Department of Energy and administered by the Oak Ridge Institute for Science and Education.
- This work was also partially supported by JST CREST Grant Numbers JPMJCR1303 (EBD CREST) and JPMJCR1687 (DEEP CREST), and performed under the auspices of Real-World Big-Data Computation Open Innovation Laboratory (RWBC-OIL), Japan.

