Understanding and Tackling the Hidden Memory Latency for Edge-based Heterogeneous Platform

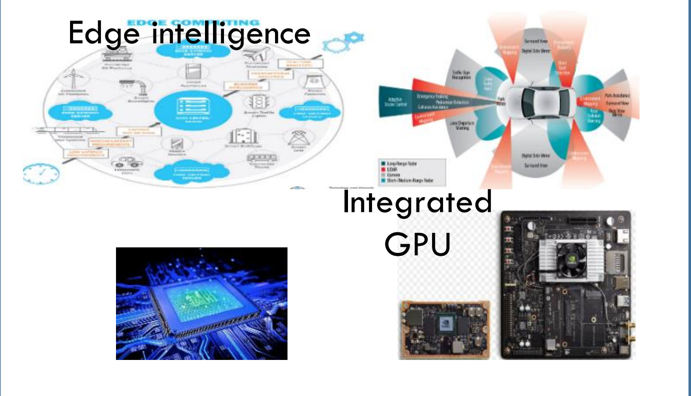
Zhendong Wang, Zhen Wang, Cong Liu, and Yang Hu

Presented by ***Zhendong Wang***

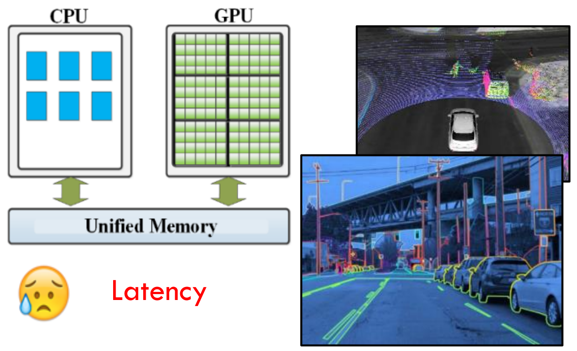
HotEdge 2020
Jun. 25, 2020

Pervasive and Emerging Architecture Research Lab, UT Dallas

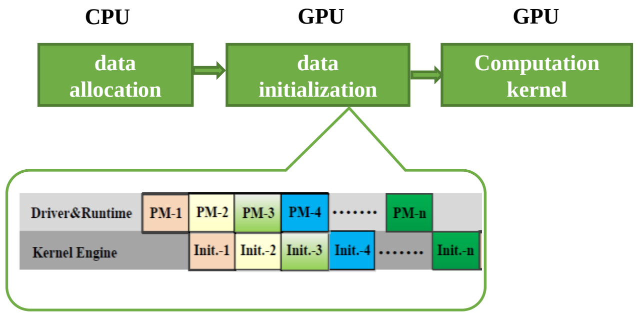
1. Background



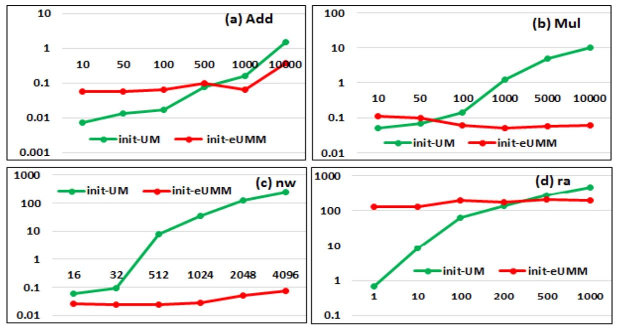
2. Motivation and Challenges



3. Proposed design



4. Evaluation and Conclusion



1. Background

ML/DNN enables a series of edge applications

Widely deployed in integrated CPU/GPU(iGPU) platform

camera
Detects traffic lights, oncoming vehicles and other obstacles
Lidar
A rotating sensor on the roof scans 200ft in all directions to create a 3D map of its surroundings
Radar
Three sensors at the front and one at the back work out the positions of nearby objects

backed by

Integrated GPU

Size 😉

Weight 😉

Power 😉

Deployment in iGPUs are stymied

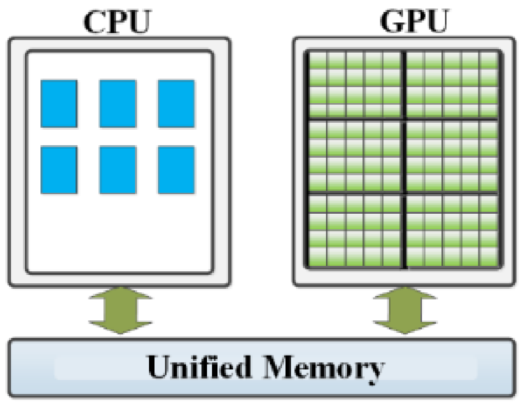
(1) Limited memory space
e.g., TX2: 8GPU, AGX:16GB

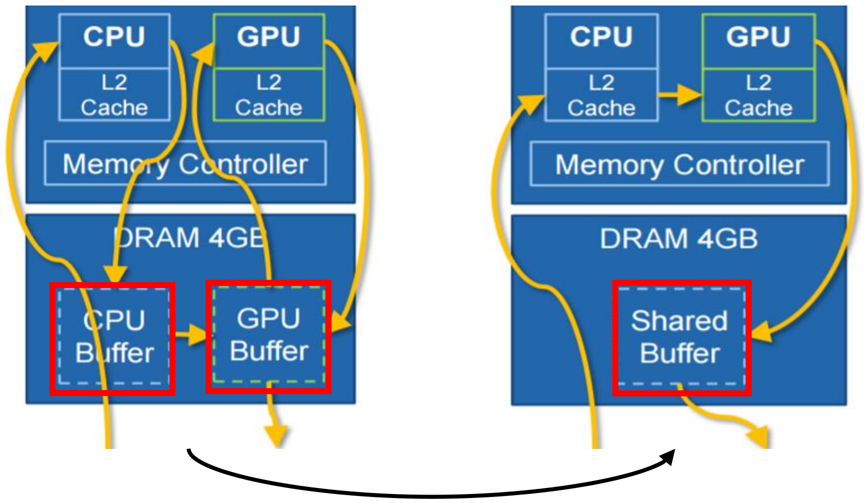
(2) Application stringent latency requirements
e.g., driving automation is safety-critical and latency-sensitive

Rigorous requirements of memory footprint and processing latency
for iGPU platform 😢

1. Background

Unified Memory (UM) Management model has relieved the situation





(1) Ease memory management

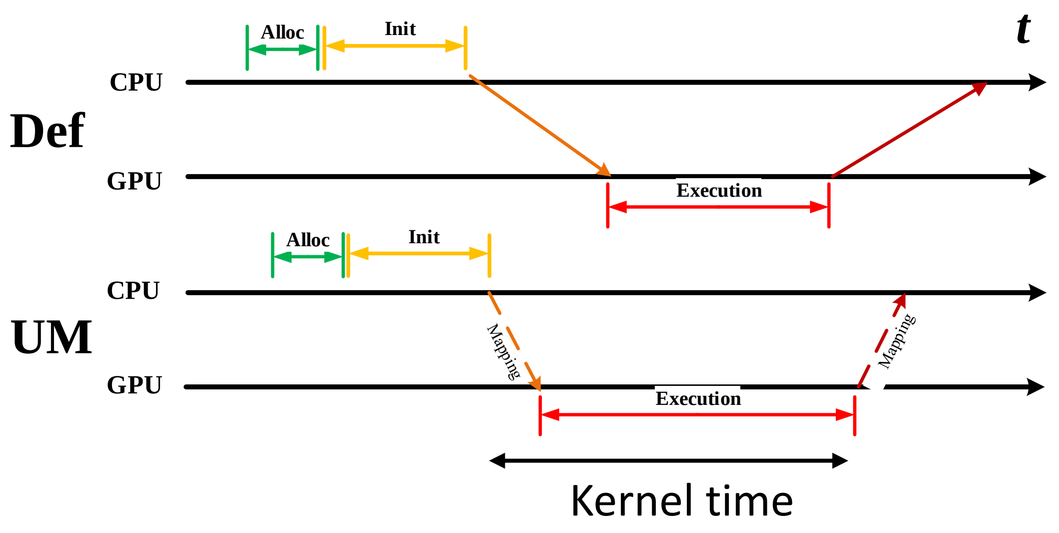
(2) Save memory footprint

CUDA: cudaMallocManaged()

Is current Unified Memory (UM) model good enough?

2. Motivation

Limits of current Unified Memory (UM) model – hidden latency



Data processing flow under Def. and UM memory model

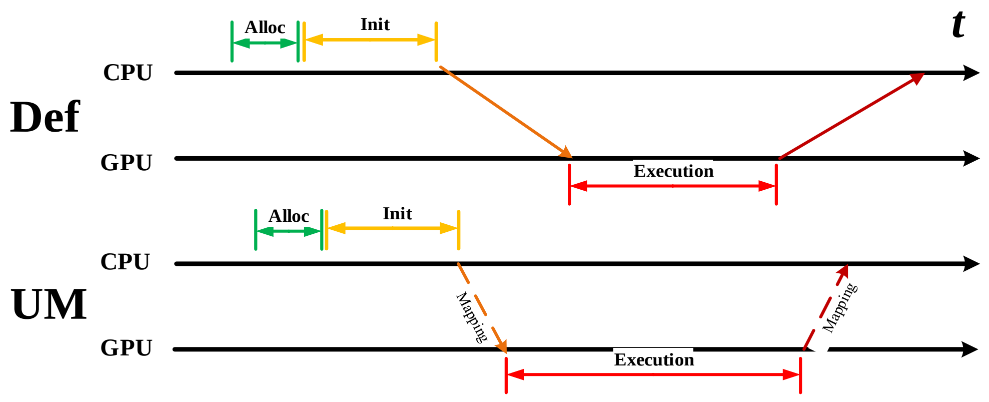
Def: copy-then-execute memory model
UM: unified memory model

Autonomous driving workloads – large matrix operation scale (M.O.S.) --- Matric Addition and Matric Multiplication

DNN	YOLO2	YOL03	SSD	DAVE-2
M.O.S	49K	81K	10K	250K

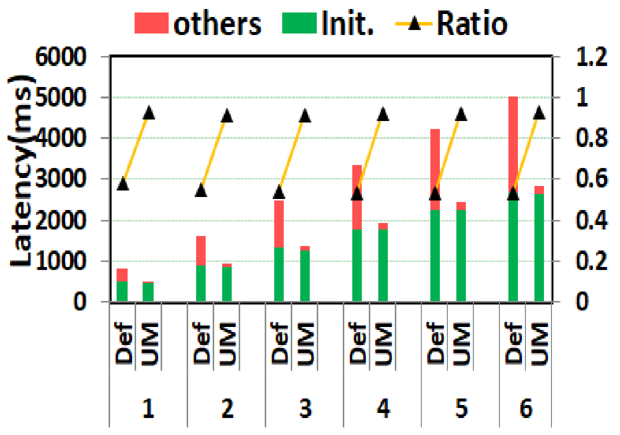
2. Motivation

Limits of current Unified Memory (UM) model – hidden latency

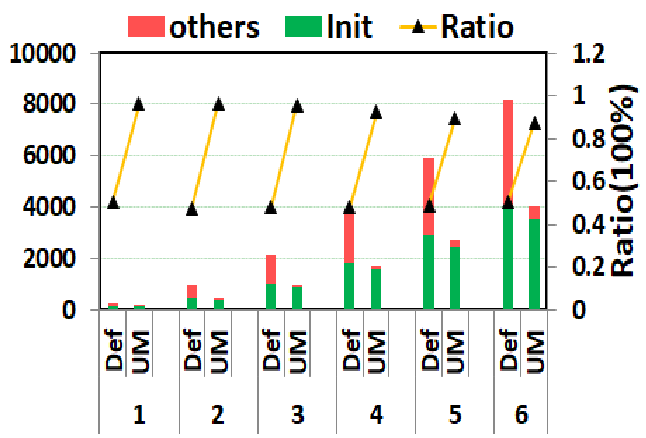


1. Def: copy-then-execute memory model
Others = H2D copy + D2H copy + kernel time
Init.: data initialization

2. UM: unified memory model
Others = kernel time (No copy)
Init.: data initialization



matrix addition

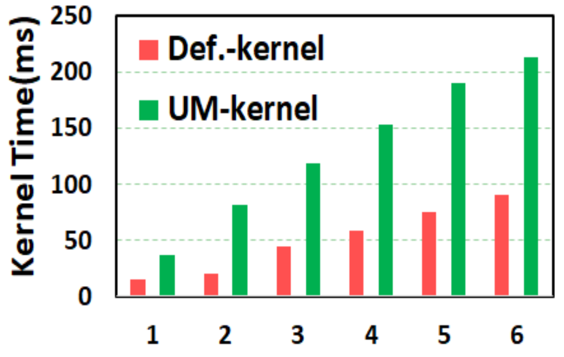


matrix multiplication

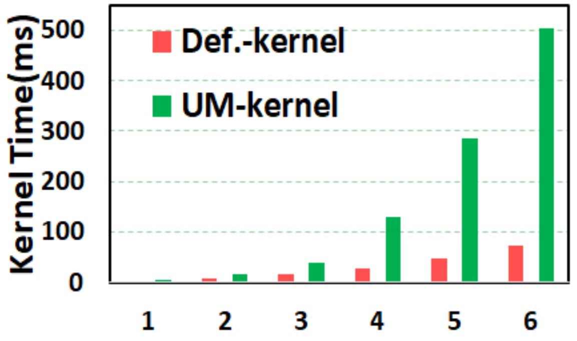
UM still spends excessive time on initialization
Def: init ~50% latency
UM: init ~90% latency

2. Motivation

Limits of current Unified Memory (UM) model – hidden latency



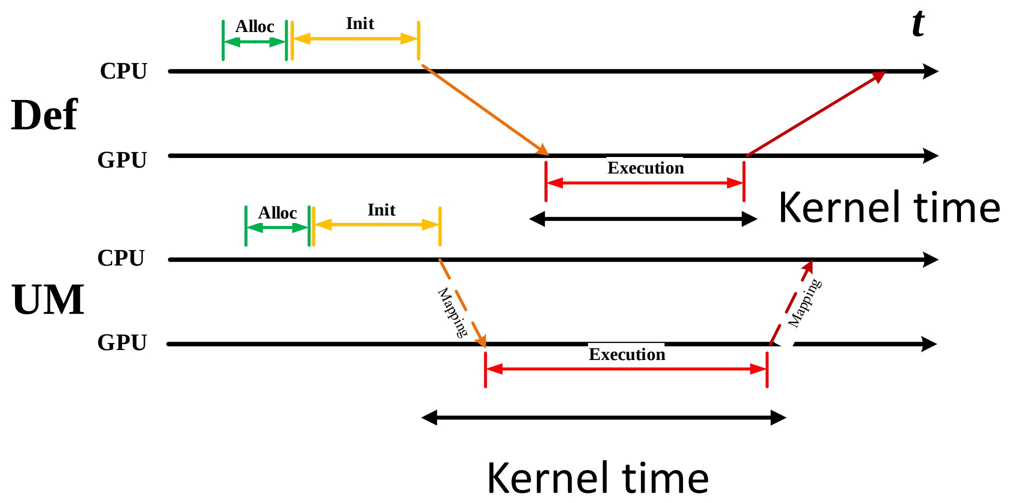
matrix addition



matrix multiplication

UM also slows down the computation kernel

1. Def kernel = kernel execution
2. UM = kernel execution + mapping latency

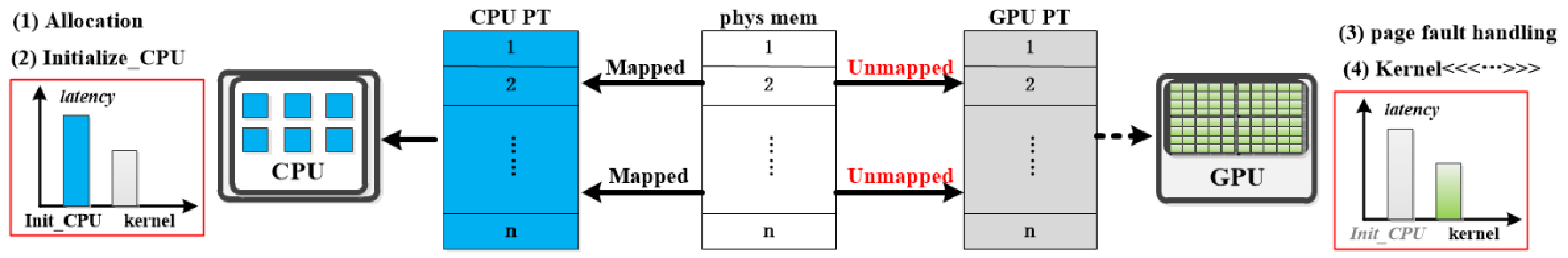


Observations: Unnecessary initialize data in CPU side
(1) Save initialization latency
(2) Benefit kernel /overall application response performance

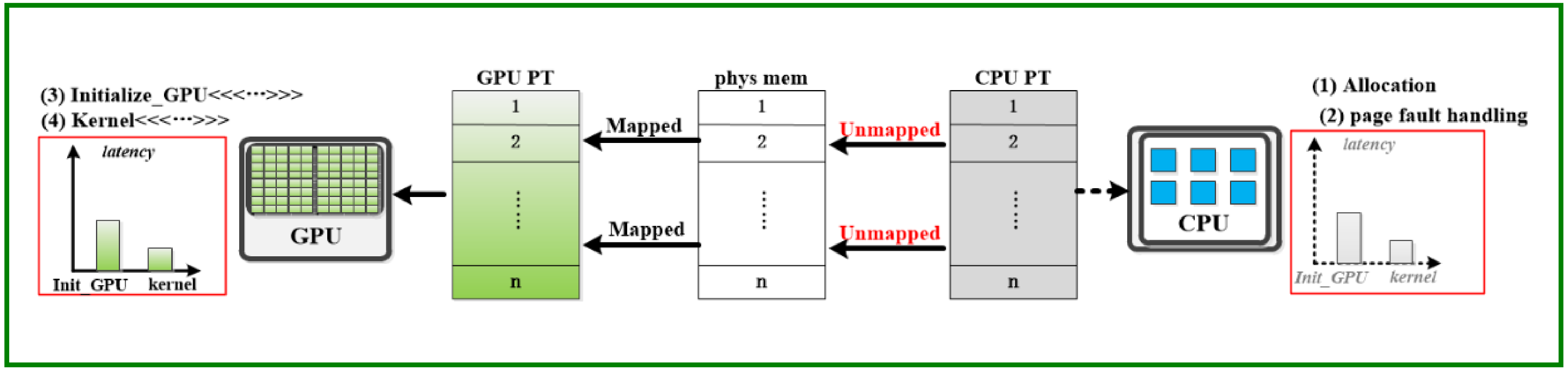
3. Proposed design

Enhanced Unified Memory Management (eUMM)

(1) Initializing data in GPU side



Existing mechanism of legacy Unified Management model

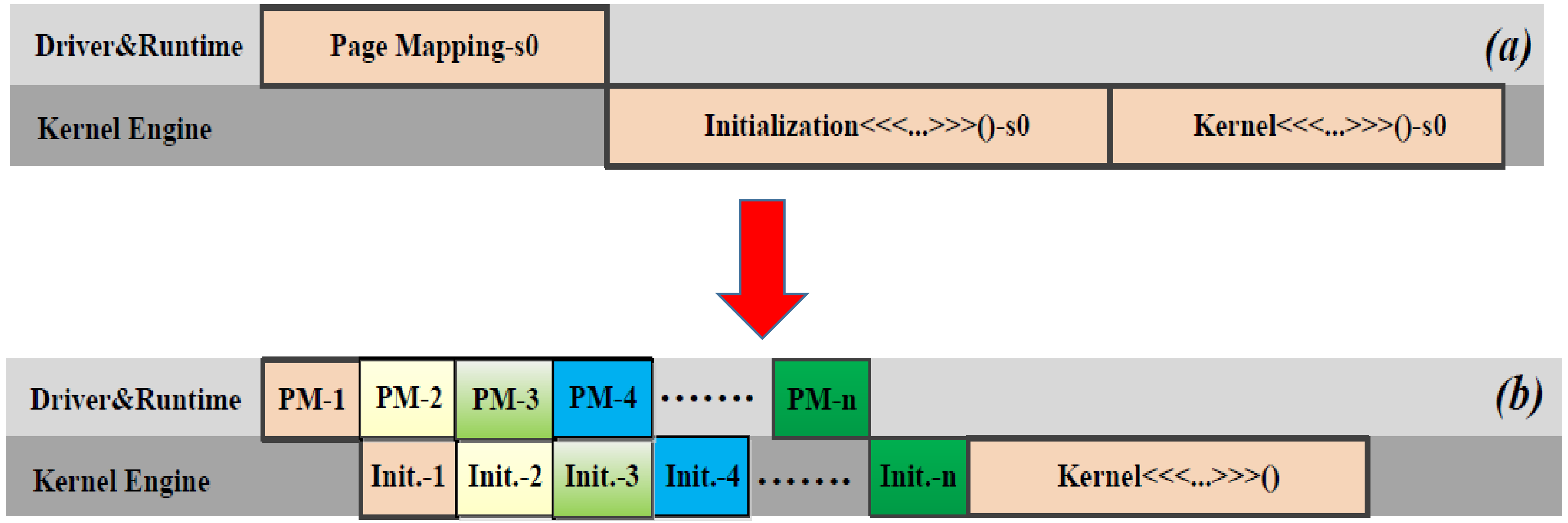


GPU-side data initialization in eUMM

3. Proposed design

Enhanced Unified Memory Management (eUMM)

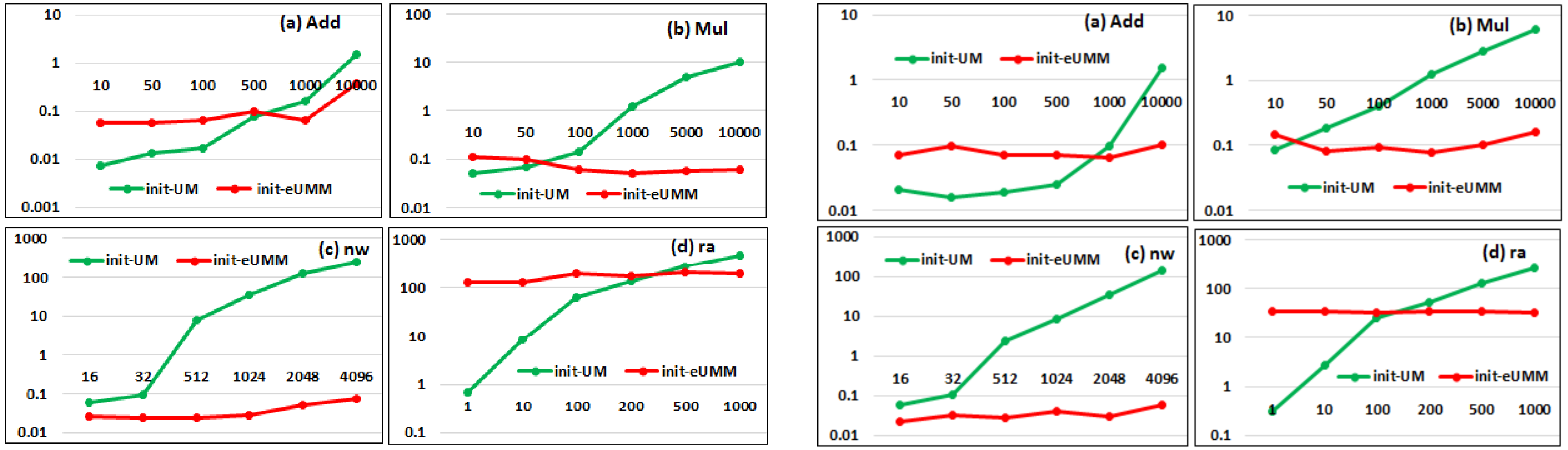
(2) Prefetch-enhanced GPU-Init performance



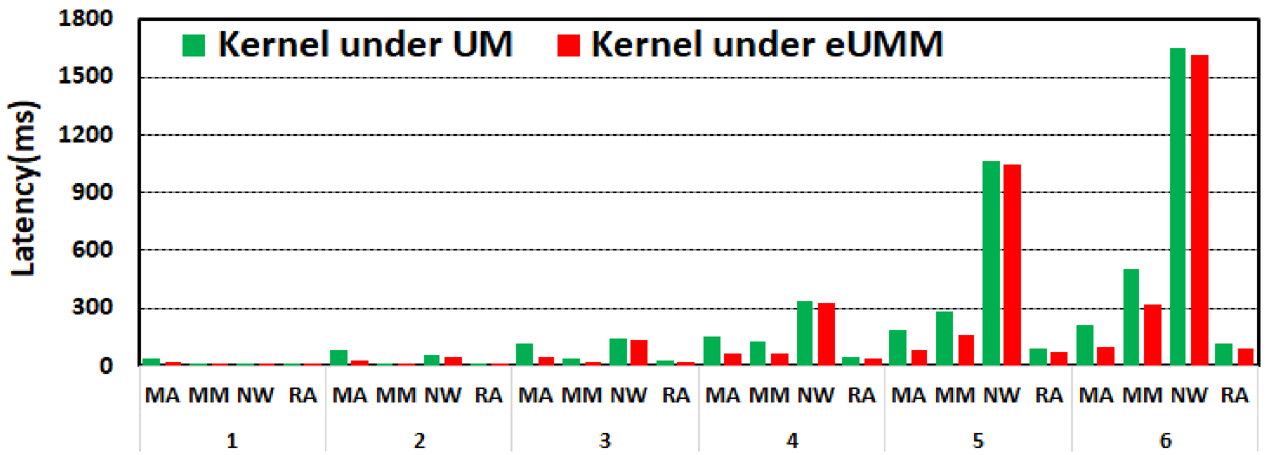
4. Evaluation

Platform: Jetson TX2, Xavier AGX Benchmark: matrix addition, matrix multiplication, Needleman-Wunsch (NW), random access (RA)

Faster data initialization



Computation kernel is not longer slowed down



5. Conclusion

- Characterization of legacy unified memory management
    - ◆Initialization latency
    - ◆Kernel launch latency
- An enhanced data initialization model based on Unified Memory management (eUMM)
    - ◆Initializing data in GPU side
    - ◆Overlapping page mapping with data initialization to further reduce latency

Prospect & Future work

- Extend eUMM to a broad spectrum of workloads
    - ◆Autonomous driving workloads (object detection, object tracking)
- Reduce the inherent overhead of GPU-side data initialization
    - ◆GPU-side data initialization does not outperform when data size is small
- GPUDirect
    - ◆Bypass CPU to accelerate the communication between GPU and peripheral storage

Thank You

If you have any questions, please contact **zhendong.wang@utdallas.edu**