DESIGNING HIGH-PERFORMANCE IN-MEMORY KEY-VALUE OPERATIONS WITH PERSISTENT GPU KERNELS AND OPENSHMEM

Ching-Hsiang Chu (The Ohio State University, NVIDIA), Sreeram Potluri (NVIDIA), **Anshuman Goswami (NVIDIA)**, Manjunath Gorentla Venkata (Oak Ridge National Laboratory), Neena Imam (Oak Ridge National Laboratory), and Chris J. Newburn (NVIDIA)

OUTLINE

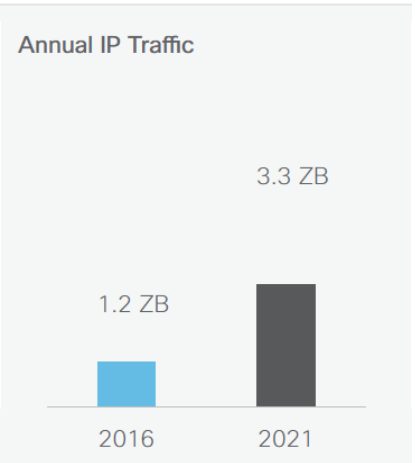
- Introduction
- Problem Statement & Challenges
- Background
- Proposed Solutions
- Evaluation
- Conclusion

HUGE DATA EXPLOSION

Predicted to continue

“Annual global IP traffic will reach 3.3 ZB (Zetta-bytes; 1000 Exa-bytes [EB]) by 2021”

source: Cisco Visual Networking Index: Forecast and Methodology, 2016-2021



“VNI Forecast Highlights Tool”, https://www.cisco.com/c/m/en_us/solutions/service-provider/vni-forecast-highlights.html

All kinds of indexed traffic

Video, Search, Maps…

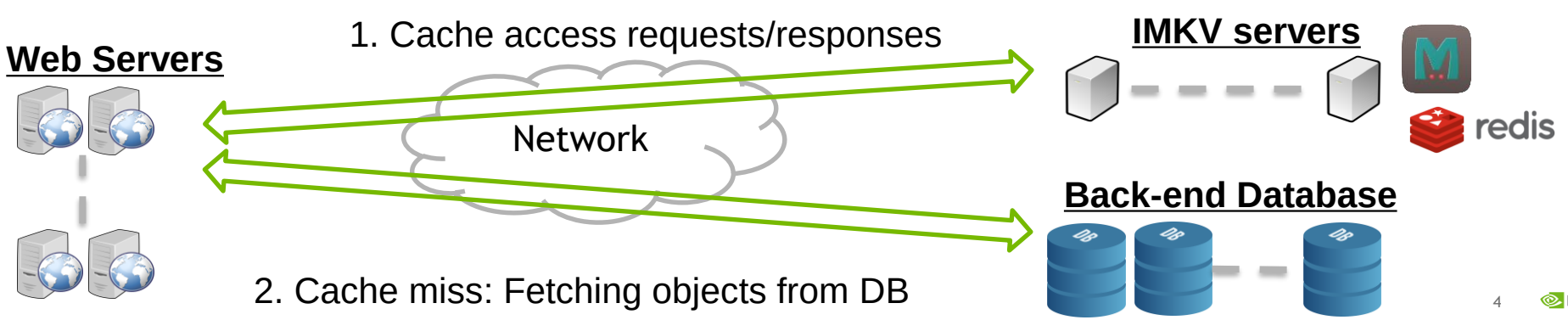
How to provide efficient access to such a large volume?

Cache!

IN-MEMORY KEY-VALUE STORES

What is it?

- In-Memory Key-Value (IMKV) Stores : Distributed in-memory data **caching** systems
  - Caching data and objects in system memory, i.e., RAM (Random Access Memory)
  - Querying the cache server(s) before retrieving data from database
  - Speeding up web applications by shortening query/response path



IN-MEMORY KEY-VALUE STORES (CONT')

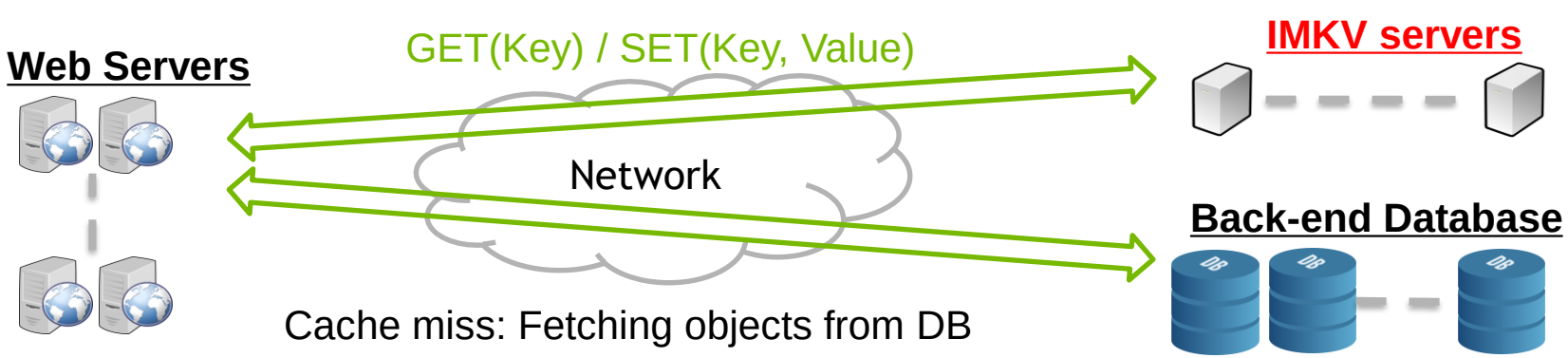
Why it becomes popular?

- **Faster for I/O-intensive applications (especially, read-heavy workloads)**
  - System memory (micro/nano-seconds) >>> database with persistent storage (milliseconds↑)
- **Resources are available:** Large system memory now common on cloud and HPC systems
  - Amazon EC2 X1e provides up to 4TB DDR4 Memory (https://aws.amazon.com/ec2/instance-types/)
  - Amazon ElastiCache offers Memcached/Redis-ready nodes equipped with up to 400 GB memory (https://aws.amazon.com/elasticache/)
  - Summit (#1 supercomputer): 512GB DDR4 per node

IN-MEMORY KEY-VALUE STORES (CONT')

How does it work?

- Fit hash table on system memory
  - Keys: 8Bytes – 250Bytes *(for memcached)*
  - Values: 8Bytes – 1Mbytes data objects *(for memcahced)*
- Basic Key-Value (KV) operations: **GET** (*the most commonly used*) & **SET**



LET'S MAKE IMKV SERVER FASTER

State-of-the-arts (*NOT a comparison*)

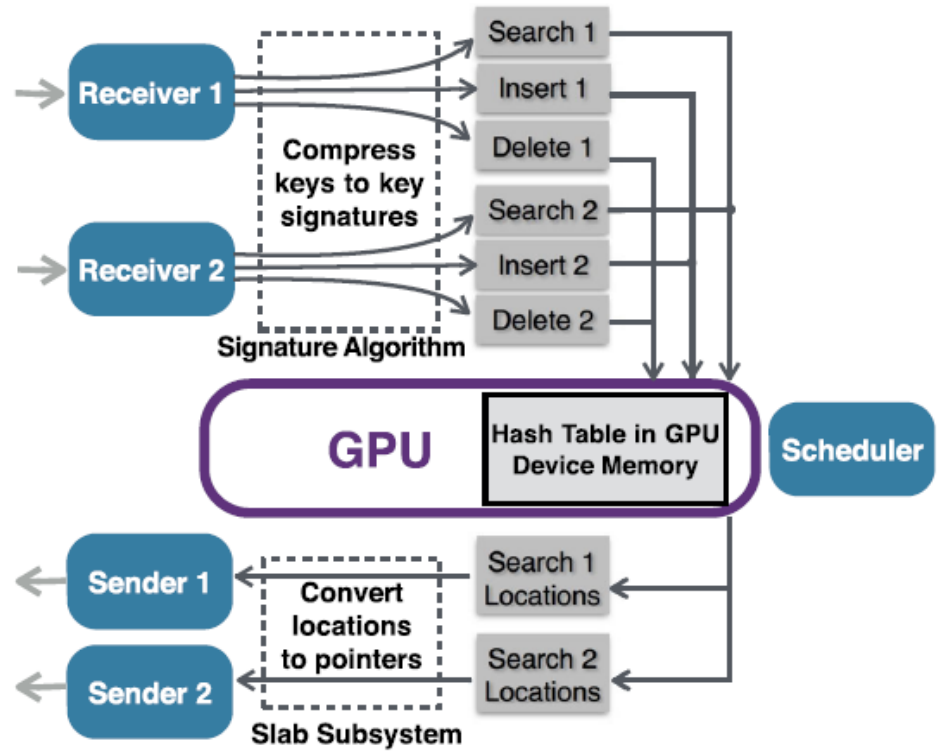
KVS	Highlights	Reported Peak throughput (MOPS)		
		Key HW	GET	PUT
Memcached	Traditional CPU based solutions, UDP	Intel 8-core Xeon CPU, One 40-GbE NIC	1.5	1.5
MICA[1]	Kernel bypass, Intel DPDK	24-core CPU, 12-port NIC	137	135
MemcachedGPU[2]	Offload packet parsing (UDP) and KV operations to GPU, GPUDirect	One Nvidia Tesla K20c, 10-GbE NIC	12.92	N/A
Mega-KV[3]	Offload only indexing operations to GPU, Intel DPDK (UDP)	Singe Node : Dual socket Intel Xeon 8-core CPUs, one Nvidia Tesla K40c GPU, one 40-GbE NIC per socket 4 nodes: Same config as above	166 623	171 N/A
SHMEMCache[4]	OpenSHMEM, RDMA, One-sided KV operations (CPU bypass)	10-core Intel Xeon CPU, IB ConnectX3 NIC 1024 nodes on Titan cluster	~0.9 ~50	~0.3 ~12
KV-Direct[5]	CPU Bypass, Offload KV operations to Programable NIC (FPGA)	Both are on a single server: One 40-GbE NIC, two PCIE Gen3 x8 link Ten 40-GbE NICs, one PCIE Gen3 x8 link per NIC	180 1220	114 610

[1] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky. MICA: a holistic approach to fast in-memory key-value storage. NSDI'14. USENIX Association, Berkeley, CA, USA, 429-444.
[2] T. H. Hetherington, Mike O'Connor, and Tor M. Aamodt. MemcachedGPU: scaling-up scale-out key-value stores. SoCC '15. ACM, New York, NY, USA, 43-57.
[3] K. Zhang, K. Wang, Y. Yuan, L. Guo, R. Lee, X. Zhang, B. He, J. Hu, and b. Hua. A distributed in-memory key-value store system on heterogeneous CPU---GPU cluster. The VLDB Journal 26, 5 (October 2017), 729-750.
[4] H. Fu, M. G. Venkata, A. R. Choudhury, N. Imam and W. Yu, "High-Performance Key-Value Store on OpenSHMEM," *2017 CCGRID*, Madrid, 2017, pp. 559-568.
[5] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang. 2017. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. SOSP '17. ACM, New York, NY, USA, 137-152.

GPU ACCELERATED IMKV SERVER

How Mega-KV works?

- KV operations are memory-intensive operations
  - GPU with HBM is a perfect fit!
  - But, GPU memory is not as BIG as system memory
- Mega-KV: Cache KVs in SysMem; Index KV on GPU
  - Introduce a new hash table in GPU memory
    - Compressed, fixed-width key
    - Value : location (in system memory)
  - Offload only indexing operations to GPUs
    - CPU threads complete KV operations



Kai Zhang, Kaibo Wang, Yuan , Lei Guo, Rubao Lee, and Xiaodong Zhang. 2015. Mega-KV: a case for GPUs to maximize the throughput of in-memory key-value stores. Proc. VLDB Endow. 8, 11 (July 2015), 1226-1237.

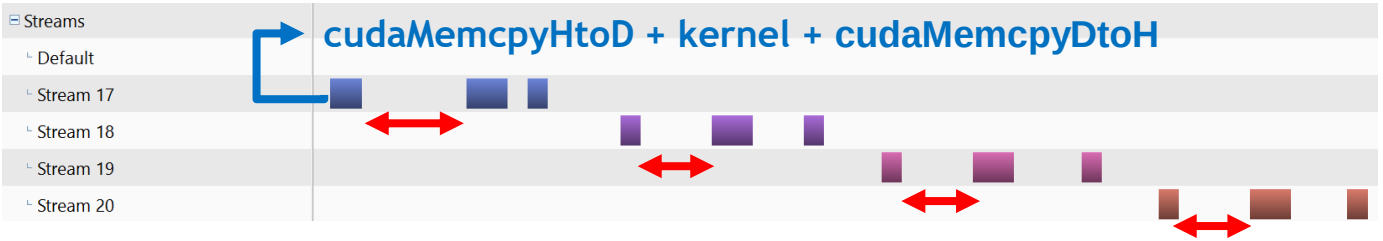
GPU ACCELERATED IMKV SERVER (CONT')

Can we do better? What is still missing in Mega-KV?

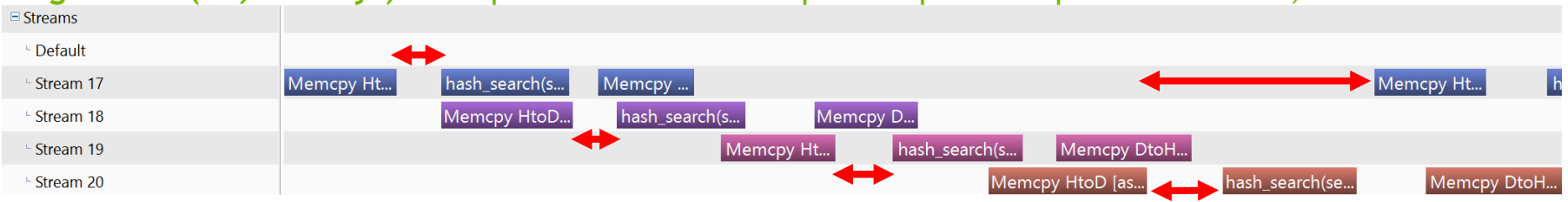
- Batching is required to achieve high-throughput
  - Data copy and CUDA kernel launch dominate indexing processes

→ longer response time for the small requests

Small batch (1,000 keys): Data copy & kernel launch take longer than actual work



Large batch (50,000 keys): Multiple CUDA streams help overlap data copies and kernels, but underutilization still present…



OUTLINE

- Introduction
- **Problem Statement & Challenges**
- Background
- Proposed Solutions
- Background
- Evaluation
- Conclusion

PROBLEM STATEMENT

Challenges

- Can we further improve the throughput of GPU-based IMKV server?
  - How to optimize data movement?
- Can we enable high-throughput indexing operations on GPU even without batching large amount of input requests?
  - How to reduce overhead when offloading operations to GPU?
  - How to maintain high GPU utilization?

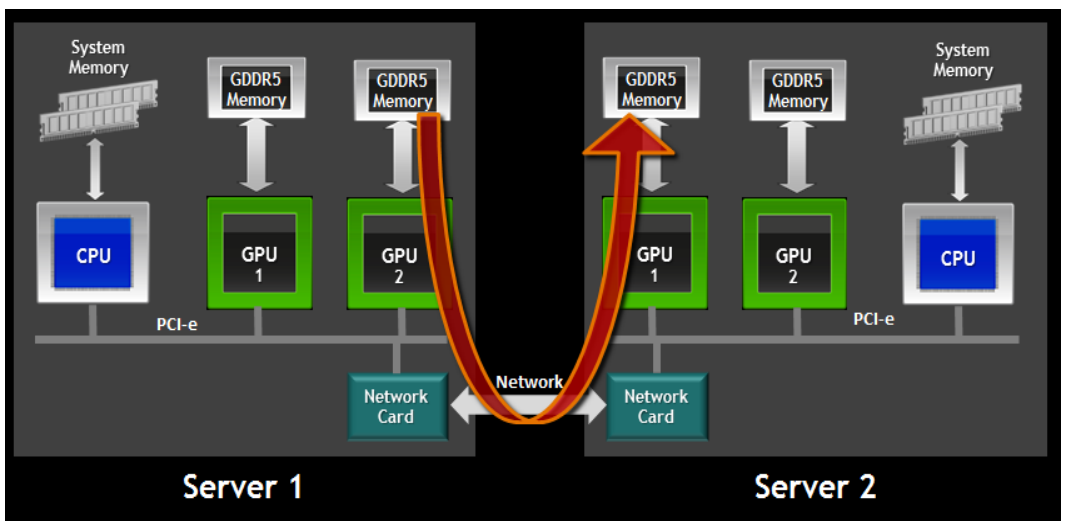
OUTLINE

- Introduction
- Problem Statement & Challenges
- **Background**
- Proposed Solutions
- Evaluation
- Conclusion

GPUDIRECT RDMA

Remote Direct Memory Access

- Introduced in CUDA 5, available on all Tesla GPUs starting from Kepler
  - Direct transfers between GPUs and NICs over PCIE
  - Eliminate CPU bandwidth and latency bottlenecks



https://developer.nvidia.com/gpudirect

OUTLINE

- Introduction
- Problem Statement & Challenges
- Background
- **Proposed Solutions**
- Evaluation
- Conclusion

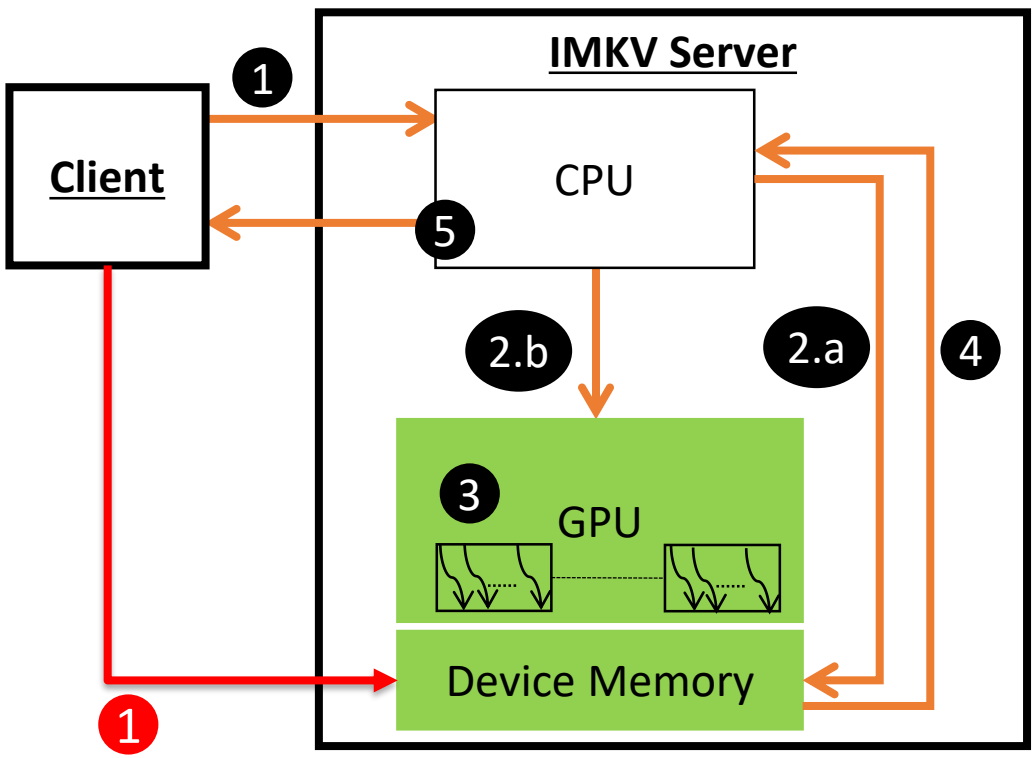
OVERVIEW

Challenges and Proposed Solutions

- Avoid copy engine overheads
  - SM-based copy (fused with processing kernel)
- Avoid copies from System Memory to Device Memory
  - Direct access to pinned system memory
  - Direct write to device memory using GPUDirect RDMA (GDR) through NVSHMEM
- Avoid kernel launch overheads
  - Persistent CUDA kernel(s)

AVOIDING COPIES

Data and control paths



1 Set/Get requests from clients

2.a Schedules memory copies

2.b Schedules GPU Kernel(s)

3 Performs indexing operations

Direct access pinned SysMem or SM-based copy

4 Copies result back to System Memory

5 Post-processing results and reply to client if required

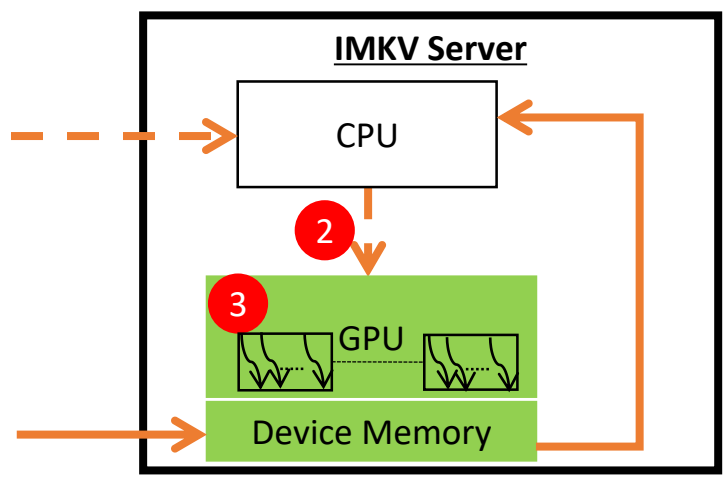
Put data directly to GPU memory using GDR technology through NVSHMEM

PERSISTENT CUDA KERNELS

Synchronization and Resource Use

- CPU thread(s) launch a persistent CUDA kernel when initializing the server
- CPU thread(s) assign and signal threadblocks/CTAs for new requests
- CTAs perform indexing operations and signal back upon completion
- CPU thread(s) check the completion and perform post-processing

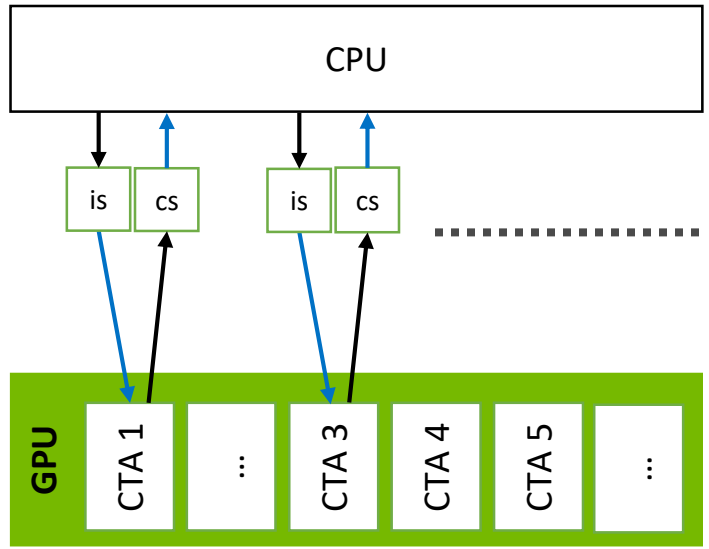
- Key design considerations
  - Low overhead signal mechanism
  - Efficient resource utilization



SIGNALING METHOD

How persistent kernel knows when and where to start processing?

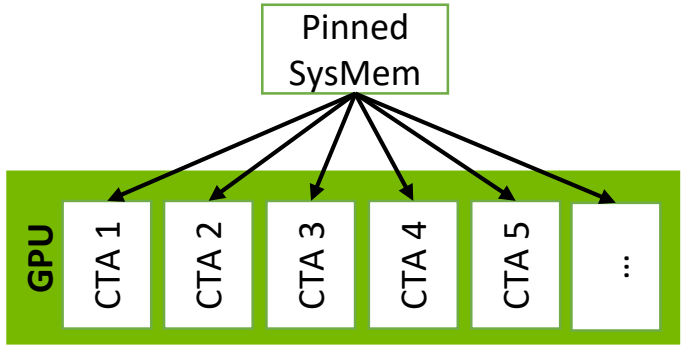
- What to signal?
  - From CPU to a particular CTA, that new work is assigned
  - Size of the assigned work
  - Pointers of input and output buffers
- How to signal?
  - Two counters and a work queue per CTA
  - issued_seqnum (is): CPU updates, CTA polls
  - completed_seqnum (cs): CTA updates, CPU polls



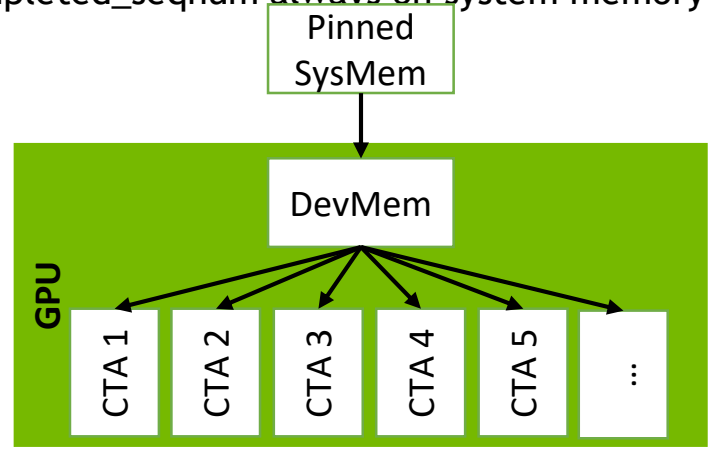
SIGNALING METHOD (CONT')

How persistent kernel knows when and where to start processing?

- Where to store the counters?
  - Pinned System Memory
    - Fast for CPU, slow for GPU to update and poll
  - Device Memory
    - Fast for GPU to poll and update but extra copies are required for CPU
    - Exploits low-latency GDRCopy [10] library which enables fast CPU update
    - Issued_seqnum on system memory or GPU memory; completed_seqnum always on system memory



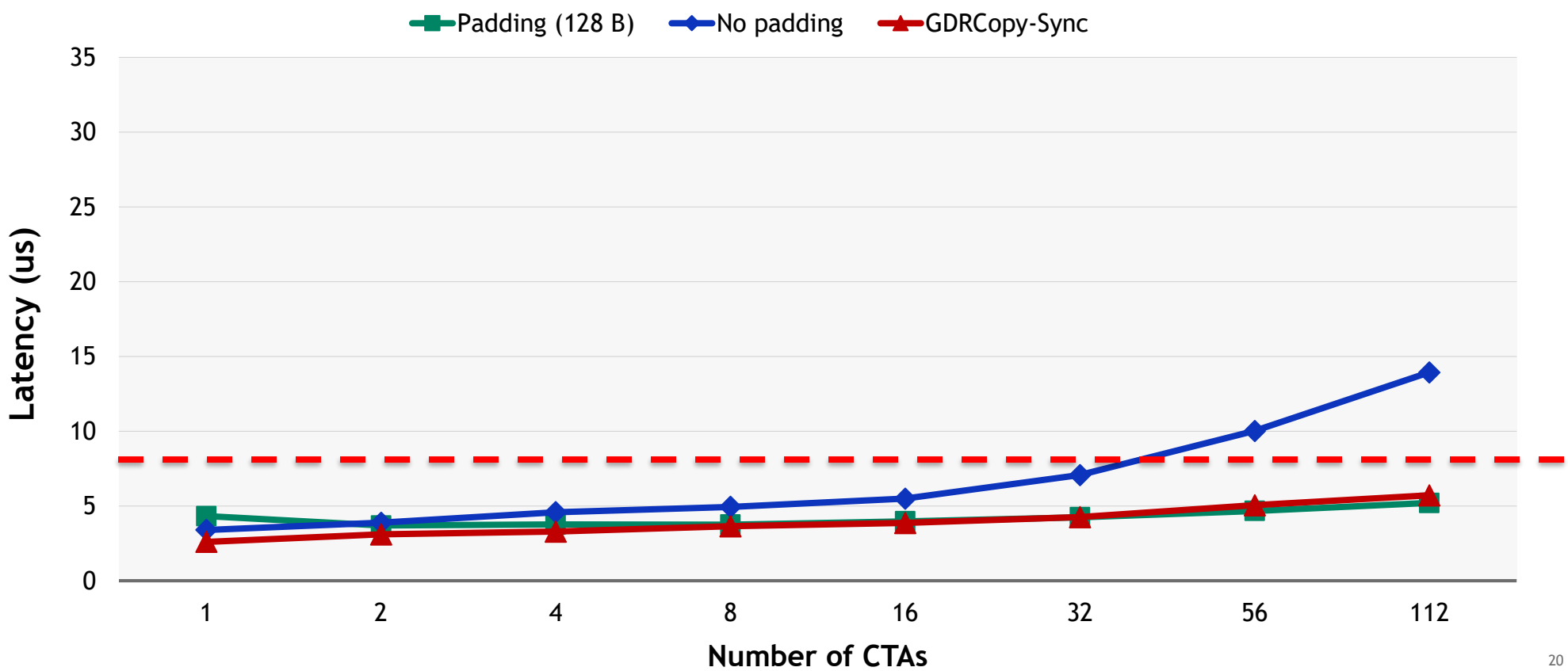
vs.



[10] https://github.com/NVIDIA/gdrcopy

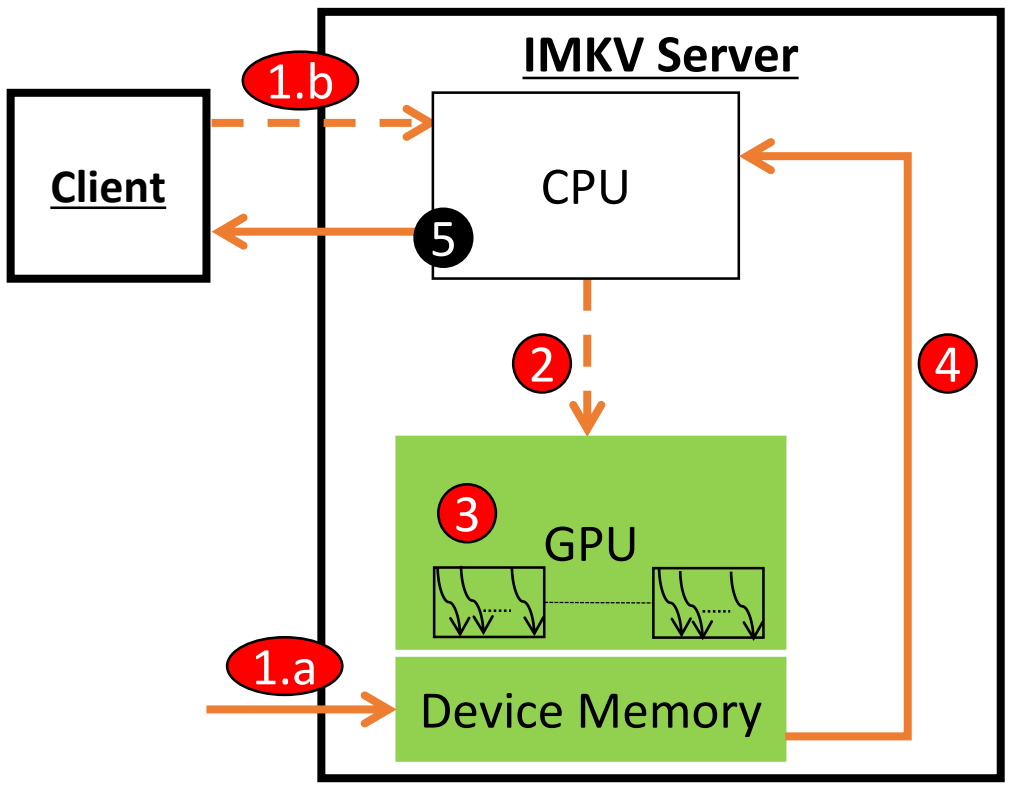
SIGNALING METHOD (CONT')

Signal overhead on Tesla P100 with 56 SMs



PERSISTENT KERNEL + GDR

Data and control paths



- 1.a Send SET/GET requests directly to GPU memory
- 1.b Client notifies server

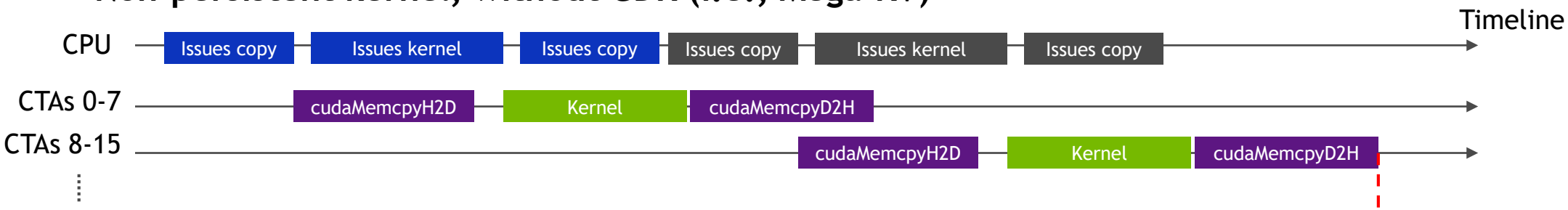
- 2 CPU signals available CTAs
- 3 Performs indexing operations
- 4 Copies result back to System Memory
  *(*SM-based or Copy Engine)*
- 5 Post-processing results and reply to client if required

BENEFITS OF PERSISTENT KERNELS + GDR

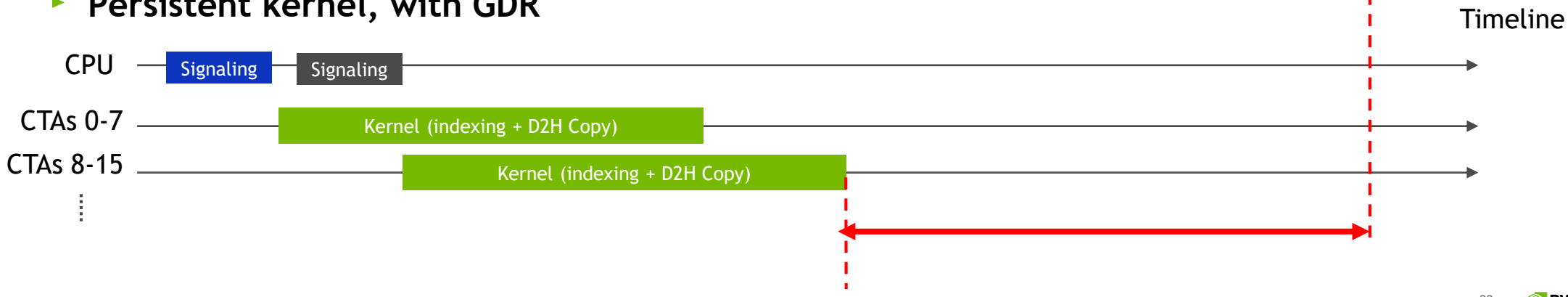
Better GPU Utilization & performance

Note: Assume processing two small batches of 1,000 keys & 8 CTAs required each batch

- **Non-persistent kernel, without GDR (i.e., Mega-KV)**



- **Persistent kernel, with GDR**



OUTLINE

- Introduction
- Problem Statement & Challenges
- Background
- Proposed Solutions
- **Evaluation**
- Conclusion

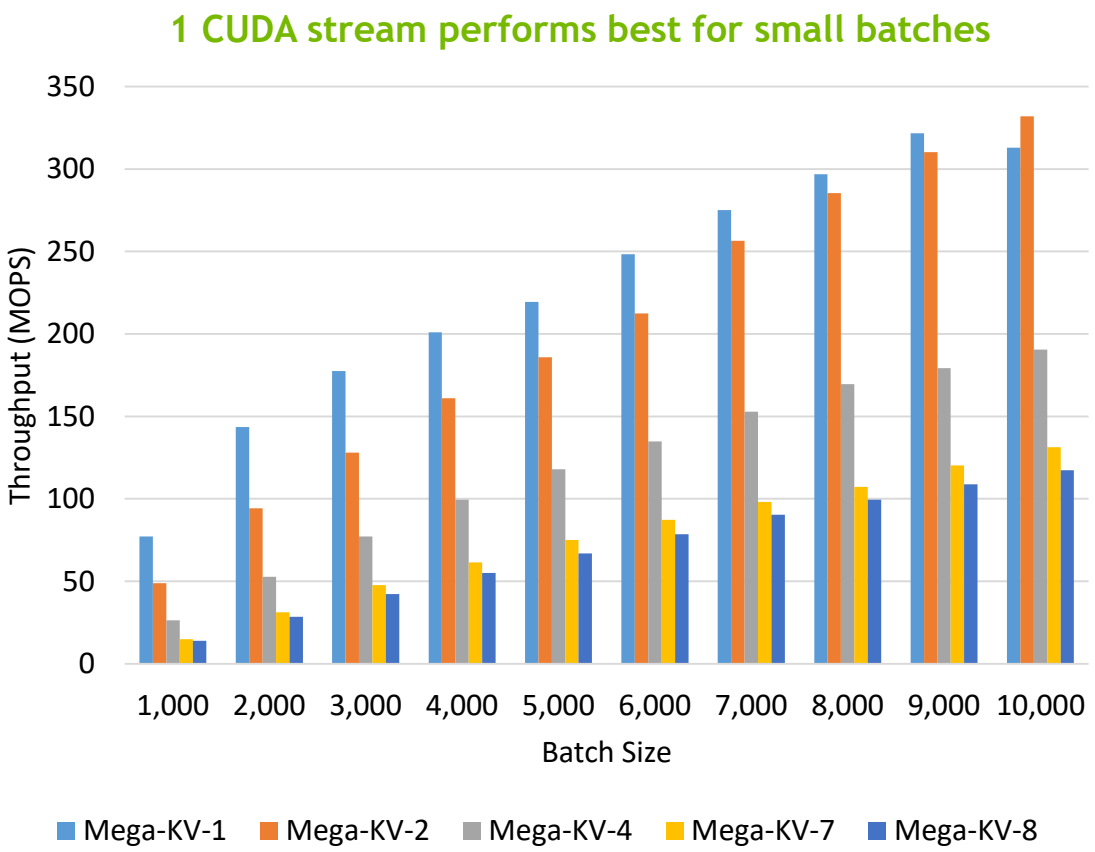
EXPERIMENTAL ENVIRONMENT

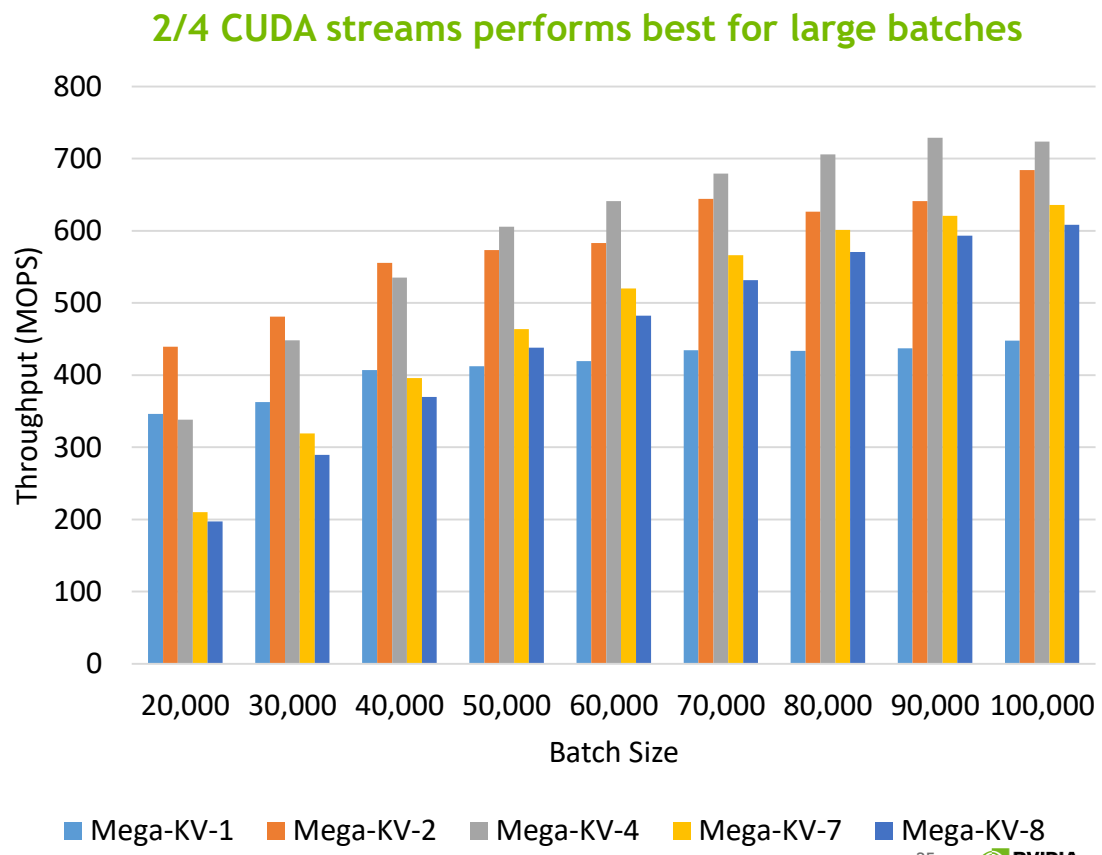
Pascal P100 GPU

- Intel Xeon dual-socket 16-core Haswell (E5-2698) CPU 2.3 GHz, 256 GB Host Memory
- InfiniBand Connect-IB (MT27600) FDR (56Gbps)
- NVIDIA Tesla P100 GPUs connected with PCIE Gen 3
  - 16 GB DDR5 memory, 56 SMs
  - CUDA 9.0.176
- Benchmark
  - Adopted from Mega-KV *(http://kay21s.github.io/megakv/)*
  - Metric: Million search Operations Per Second (MOPS)
    - i.e., throughput, higher is better

BASELINE

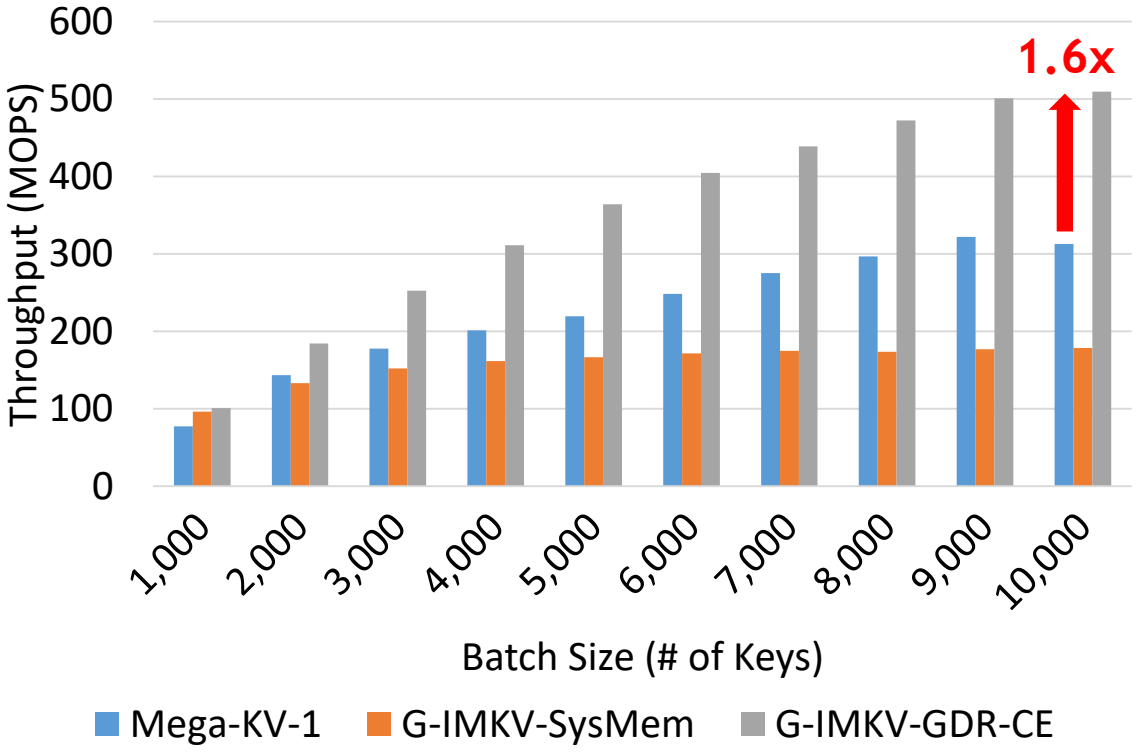
Mega-KV on P100: Tuning # of CUDA Streams

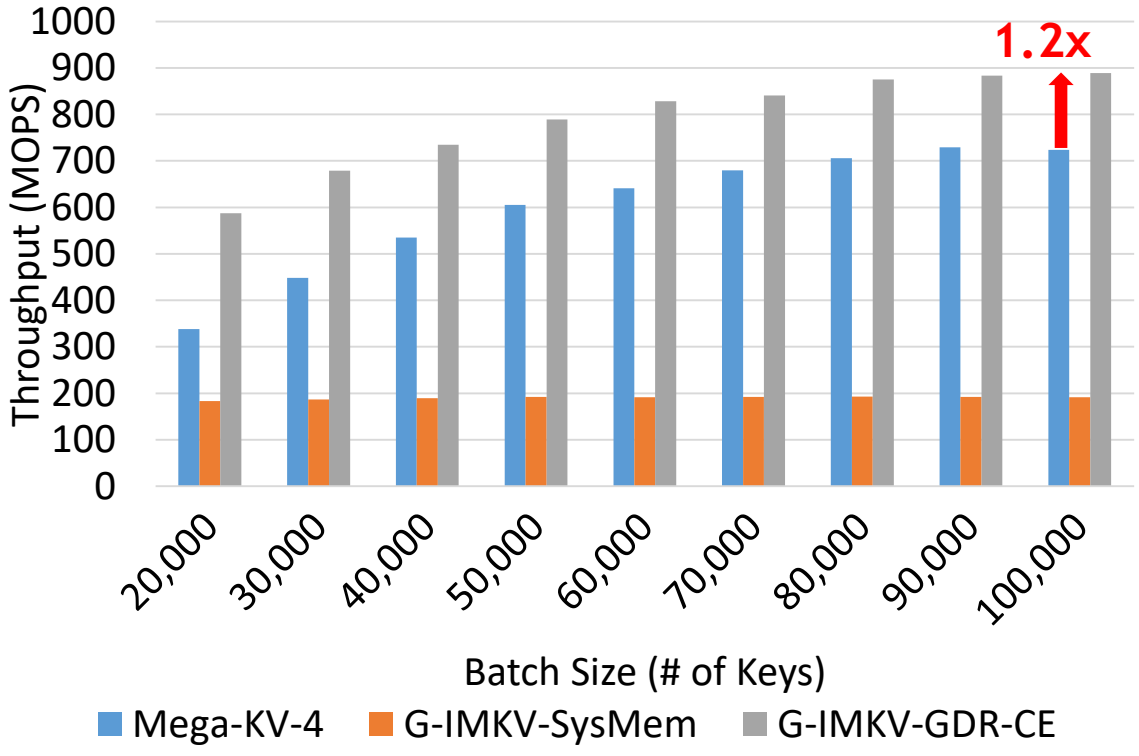




AVOID COPIES

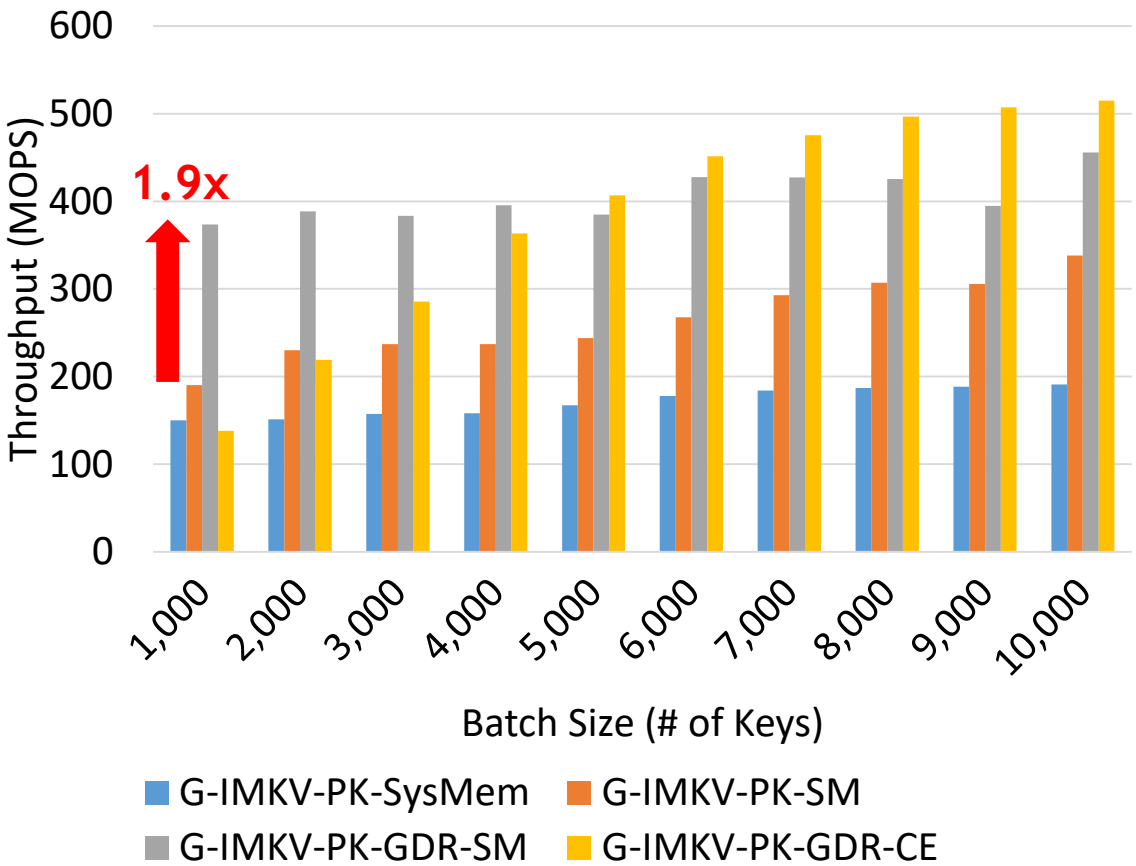
NVSHMEM with GDR improves throughput!

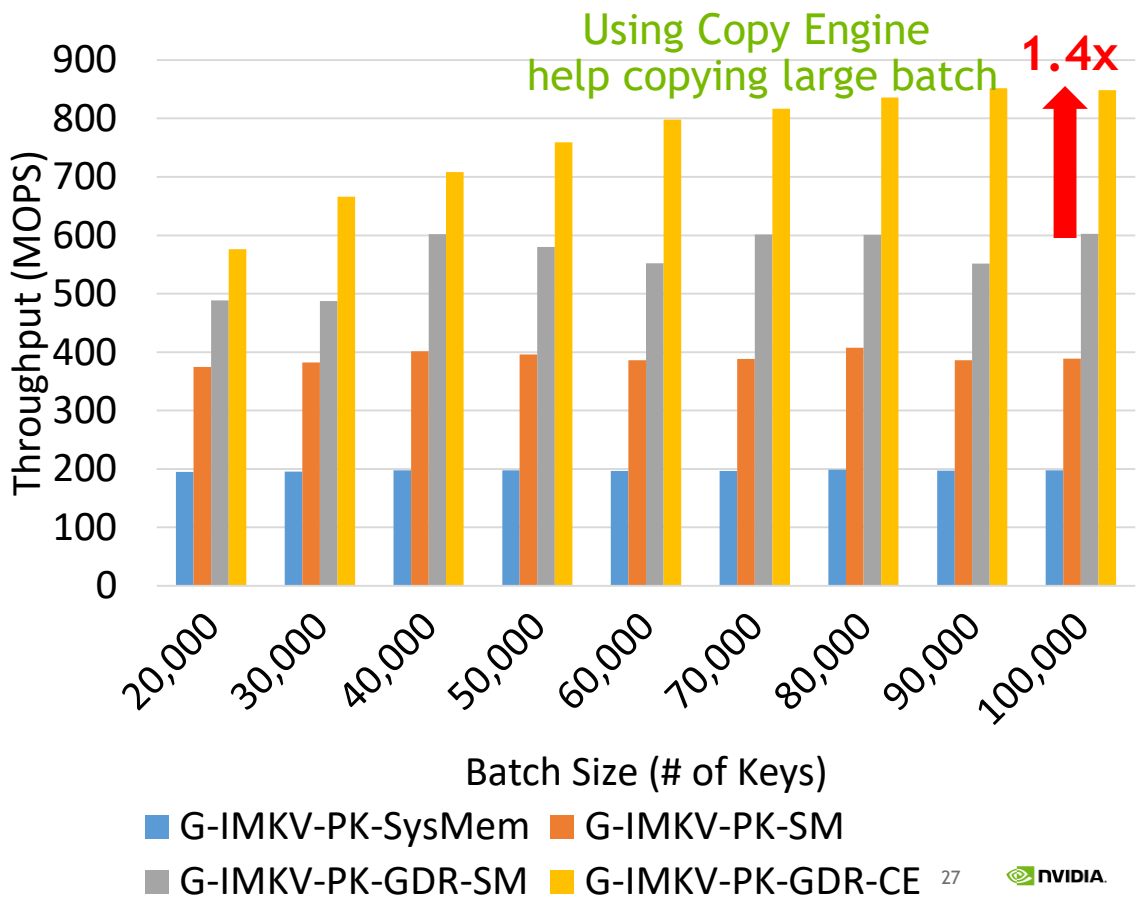




PERSISTENT KERNEL

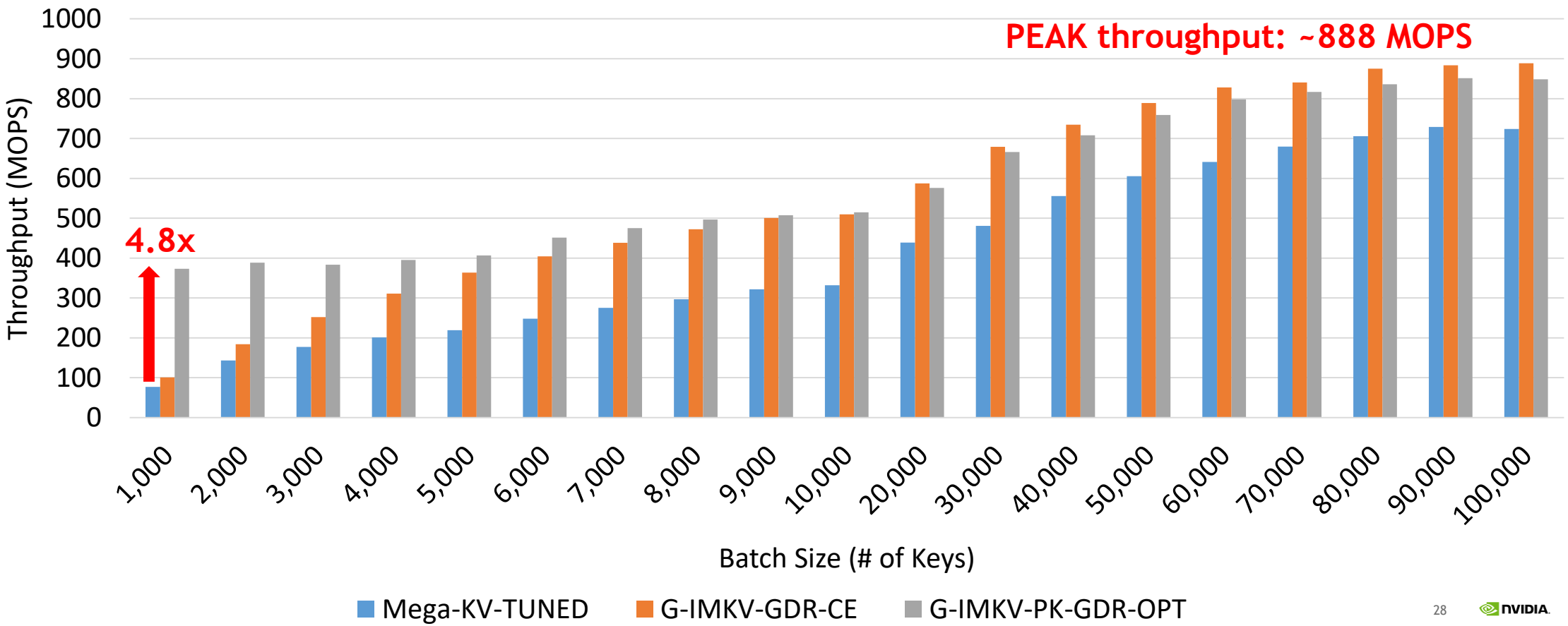
Choosing the appropriate copy back mechanism!





GDR, PERSISTENT KERNEL

vs. MegaKV



OUTLINE

- Introduction
- Problem Statement & Challenges
- Background
- Proposed Solutions
- Evaluation
- **Conclusion**

CONCLUSION & FUTURE WORK

Key takeaways

- Use of GPUDirect RDMA eliminates data transfer and copy command overhead
  - 1.2x speedup for large batches and 4.8x speedup for small batches compared to Mega-KV
- Persistent CUDA kernel mitigate kernel launch overhead
  - 3.7x higher throughput for small batch sizes compared to Mega-KV
- Future Work & Open issue
  - Evaluate scalability: IMKV server(s) with multiple GPUs and NICs
  - Extend to other streaming applications
  - Investigate CUDA level guarantees required for concurrent progress of persistent kernels and memory copies


NVIDIA