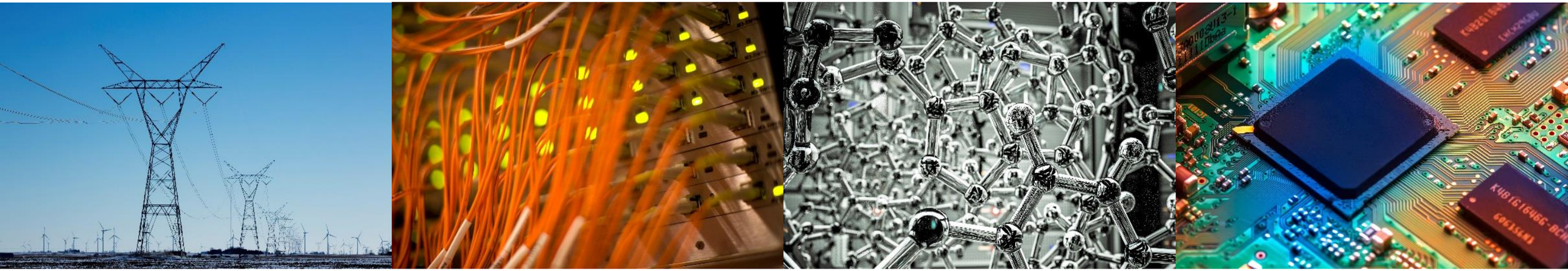
# Fast GPU Code for Graphs

**Wen-mei Hwu[1],** David Min, Vikram S. Mailthody[1], Zaid Qureshi[2], Carl Pearson[1], Mohammad Almasri[1], Omer Anjum[1] , Rakesh Nagi[3], Jinjun Xiong[4], Eiman Ebrahimi[5]

[1] ECE, [2] CS, [3] ISE, University of Illinois at Urbana-Champaign, [4] IBM Thomas J. Watson Research Center, [5] NVIDIA Research



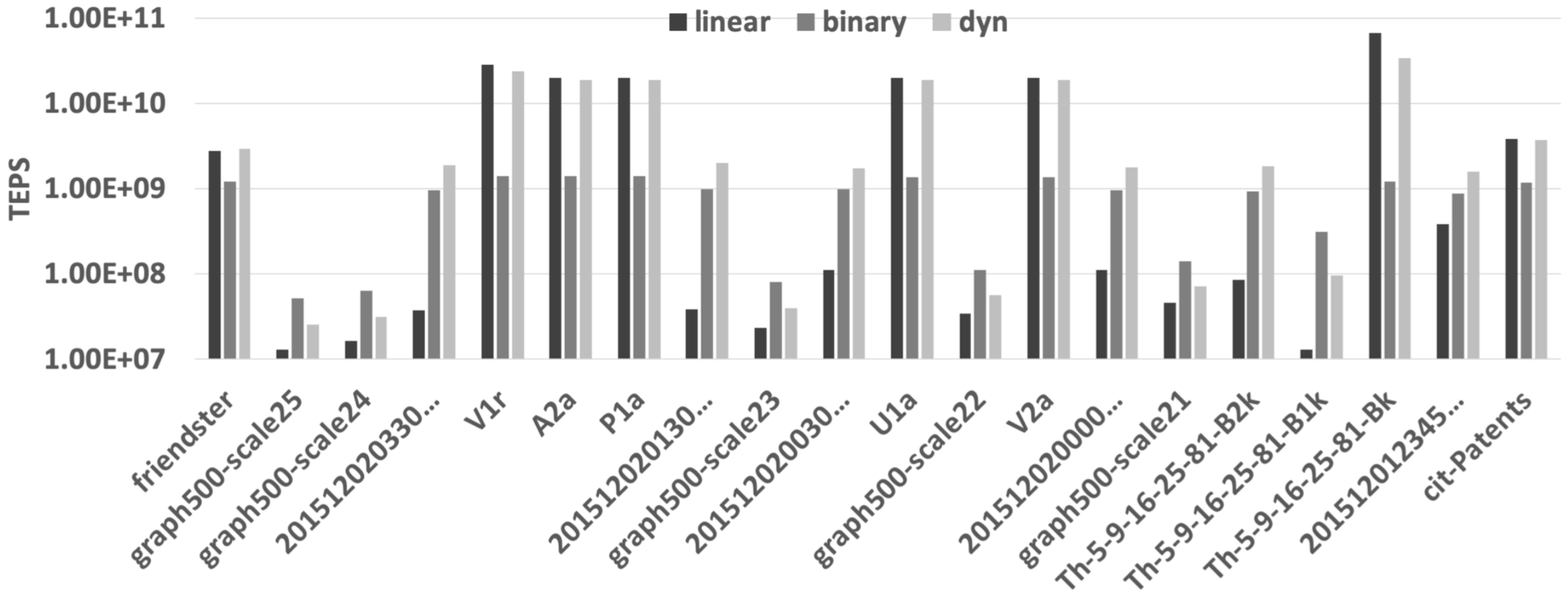
ILLINOIS
Electrical & Computer Engineering
COLLEGE OF ENGINEERING

IBM The IMPACT Research Group C3SR ISE

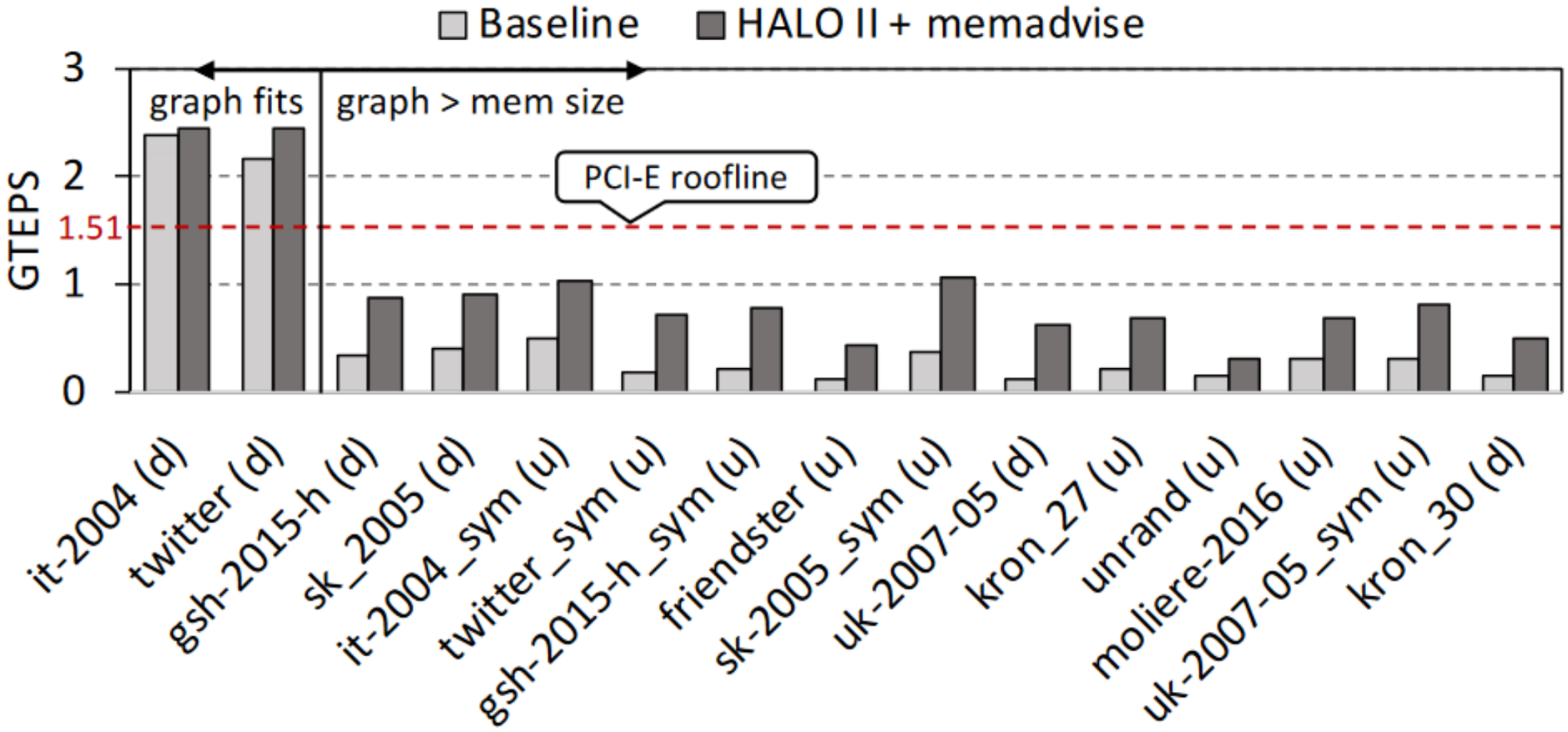
# Graph Usage Is Growing

- Modern NLU/NLP and recommendation systems are increasingly based on graphs data that capture the relations among entities being analyzed.
  - Keyword/phrase/concept extraction and role-determination
  - Name-entity resolution
  - Timeline resolution
  - Causal relation resolution
- Reviewer recommendations, citation recommendations, recruiting recommendations, …

# 2019 TC - Dynamic Algorithm Selection (P9+Volta)

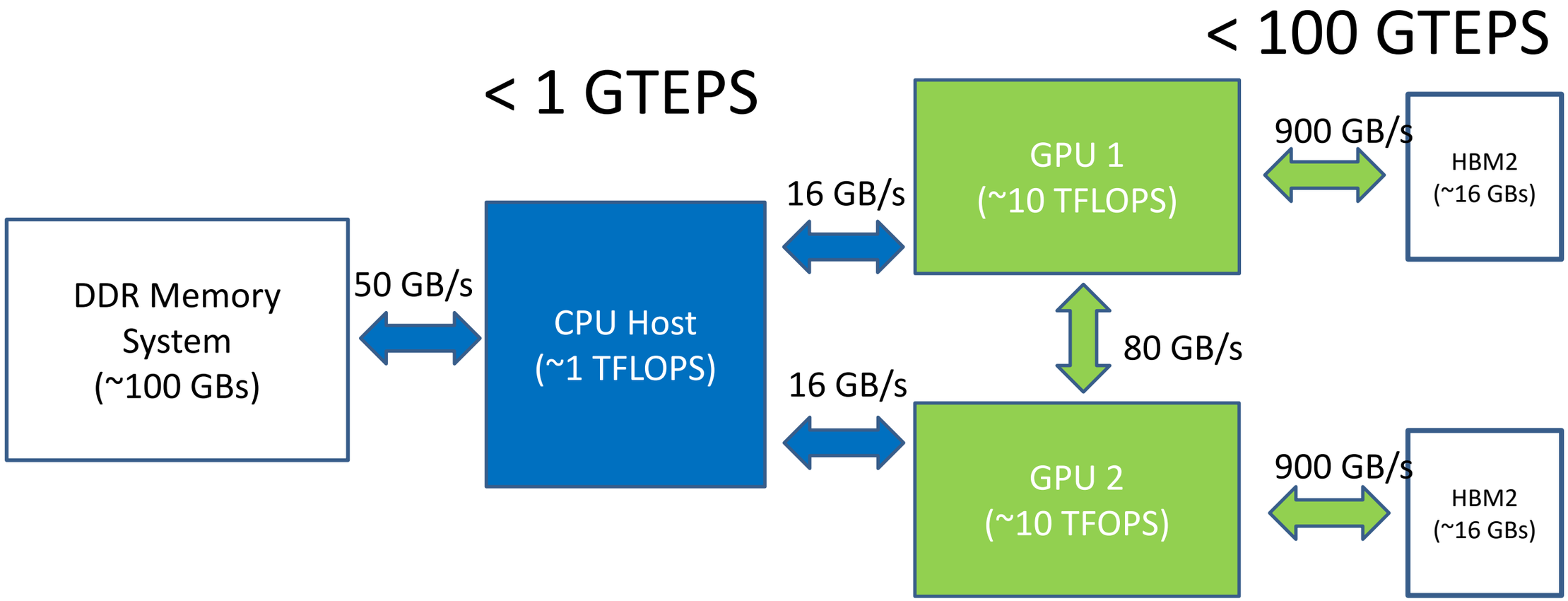


# 2020 state of large graph traversal (BFS, UVM)



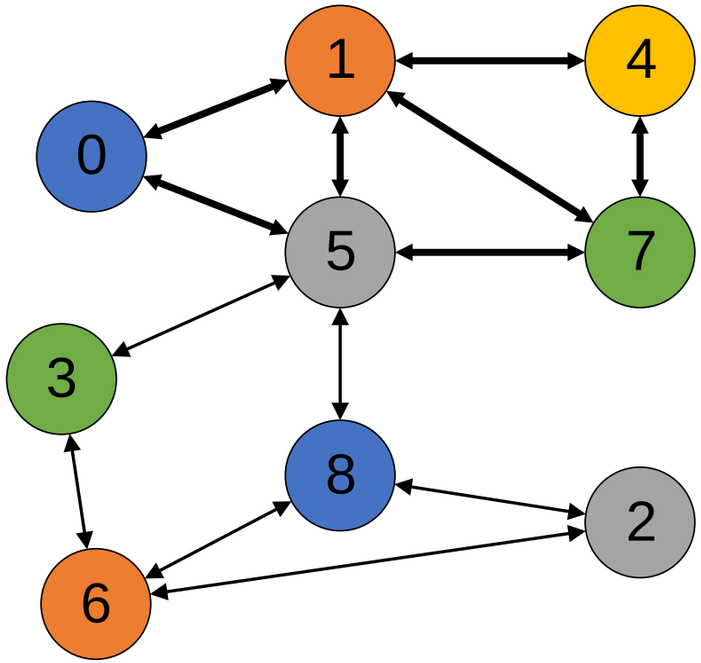

Prasun Gera, Hyojong Kim, Piyush Sao, Hyesoon Kim, and David Bader. 2020. Traversing large graphs on GPUs with unified memory. *Proc. VLDB Endow.* 13, 7 (March 2020), 1119–1133.

# A Simplified View of an X86+Volta PCIe System

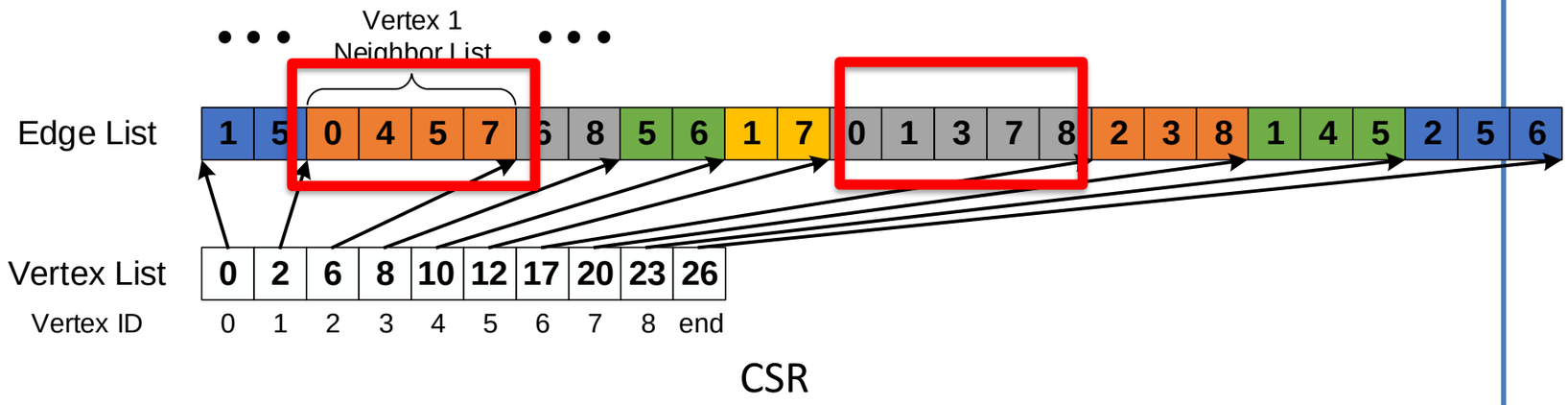


# Graph traversal

- Start from a certain vertex (or vertices) and keep traversing neighboring vertices iteratively (recursively)

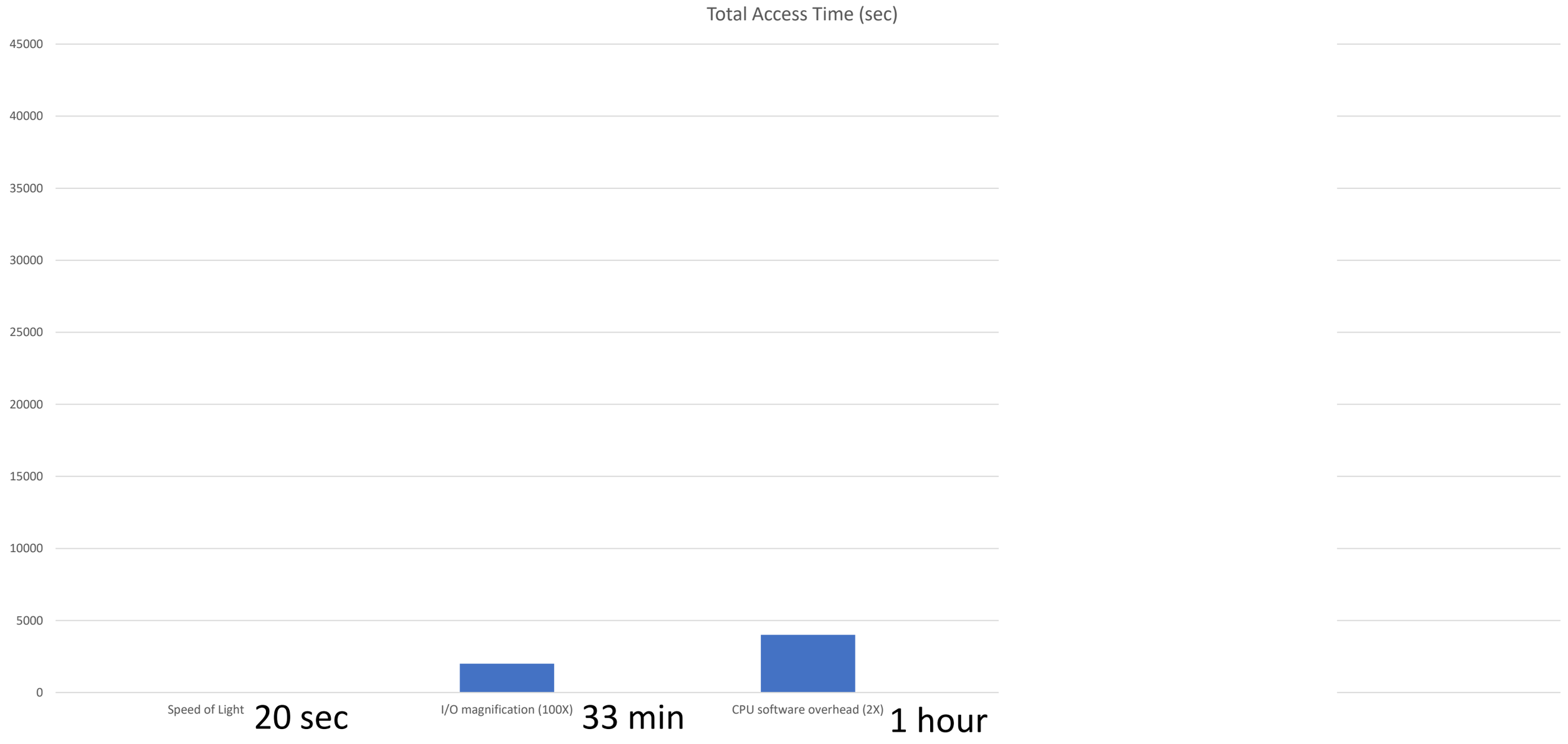



Graph




CSR

# Sparse Access to 1TB in Much Larger Data via 50GB/s PCIe Link



# Maybe forget about the data copy, just access it?

- GPU can also directly access the host memory
  - Also called as *Zero-copy*
- Can we make full use of PCIe only with zero-copy accesses?
- Conventional Wisdom says no – header overhead, latency, …
- Well… let's see

# Interconnects are gaining bandwidth

- PCIe Gen 3 - 16GB/s (12 GB/s usable)
- PCIe Gen 4 - 32GB/s (24 GB/s usable)
- PCIe Gen 5 - 64GB/s (48 GB/s usable)

- NVLink to GPU will have 300 GB/s in the same timeframe as PCIe Gen 4

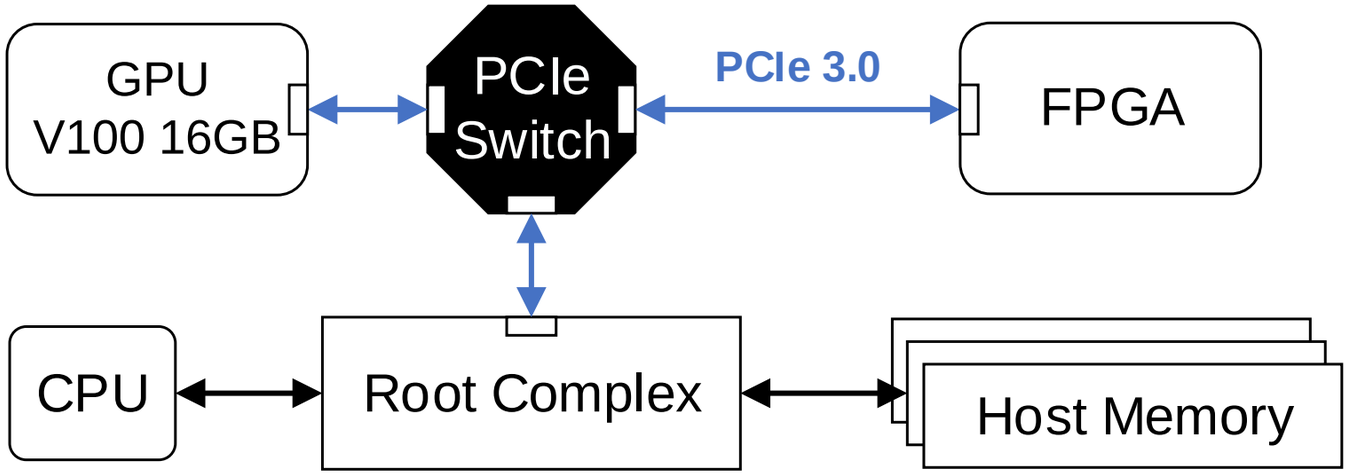
- Any application should have scalability in mind

# Challenges with UVM

- Large I/O Amplification
- Significant CPU paging software overhead
- Limited amount of parallelism in page fault handling

# Accessing Host Memory with Zero-Copy

- System setup



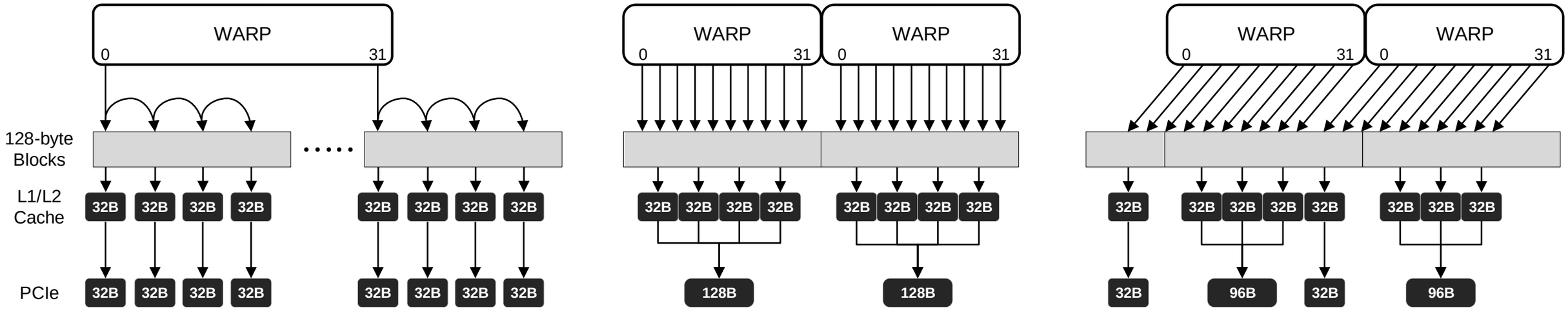

- Impersonate Host memory to the GPU to observe access patterns such as PCIe transaction size
- Measured latency roundtrip PCIe 1.0-1.4 µs

# Challenges with Zero-Copy Access

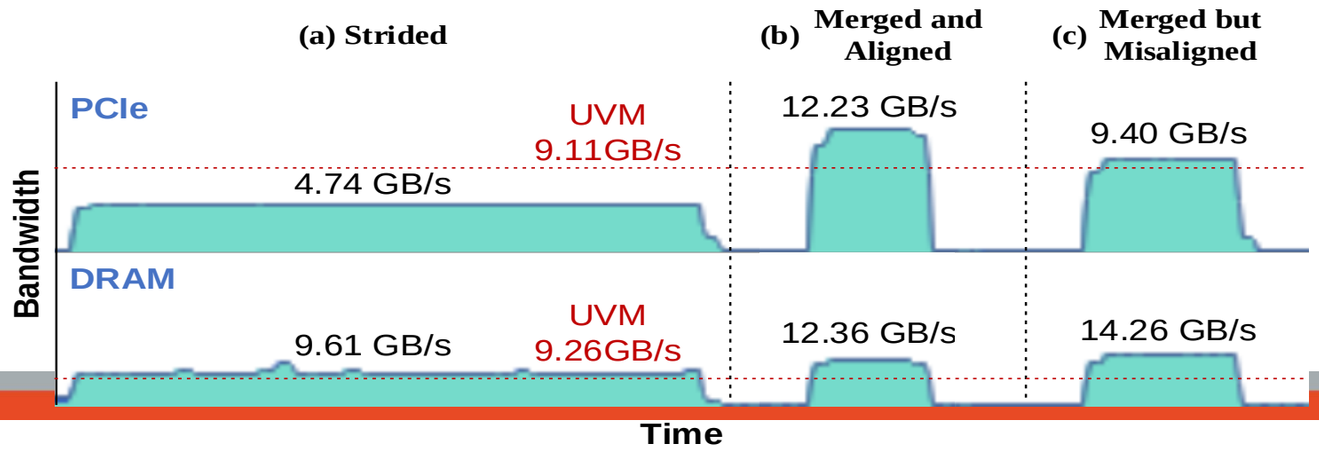
- Must have enough overlapping accesses to tolerate latency and fully utilize the PCIe link
  - Each 128B access only occupy the 16GB/s PCIe for 8ns.
  - To fully utilize the PCIe link must have at least 1000/8 = 125 overlapping accesses

- Must have well-coalesced accesses – there is a 18B Transaction Layer Packet header
  - The maximal effective bandwidth with 128B, 64B, and 32B accesses will be 14 GB/s, 12.48 GB/s and 10.24 GB/s

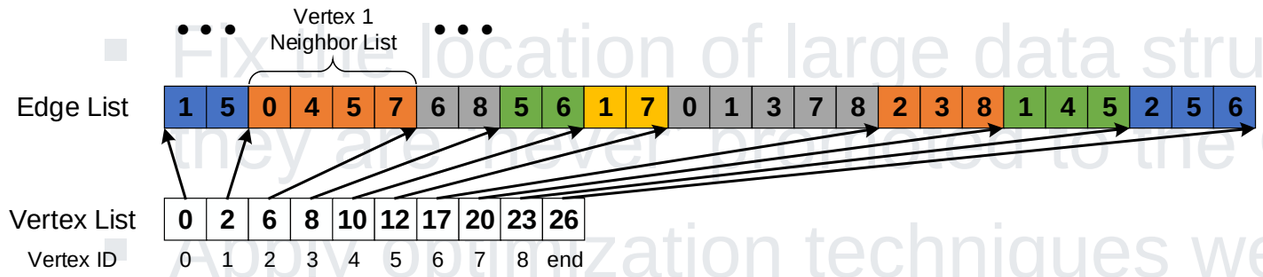
# Data Access Patterns for the Neighbor List




(a) Strided (b) Merged and Aligned (c) Merged but Misaligned



# EMOGI: Zero-copy graph traversal




**Listing 1: Uncoalesced Memory Access**

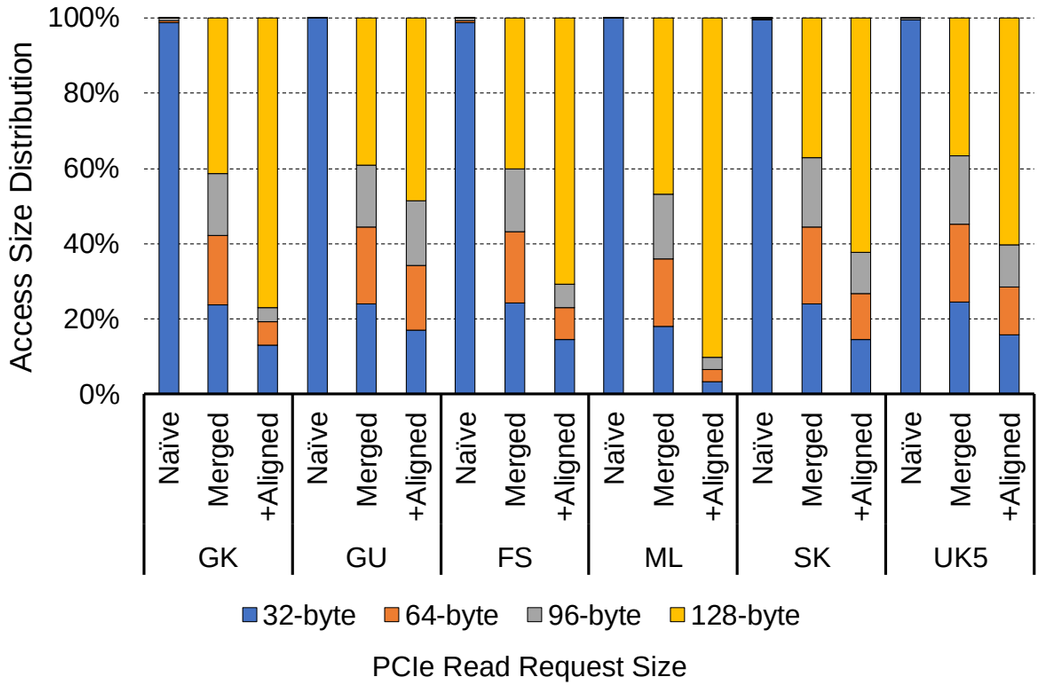
```
1  void naive(*edgeList, *offset, ...) {
2      thread_id = get_thread_id();
3      ...
4      start = offset[thread_id];
5      end = offset[thread_id + 1];
6
7      // Each thread loops over a chunk of edge list
8      for (i = start; i < end; i++) {
9          edgeDst = edgeList[i];
10         ...
11     }
12     ...
13 }
```

**Listing 2: Coalesced Memory Access (Merged + Aligned)**

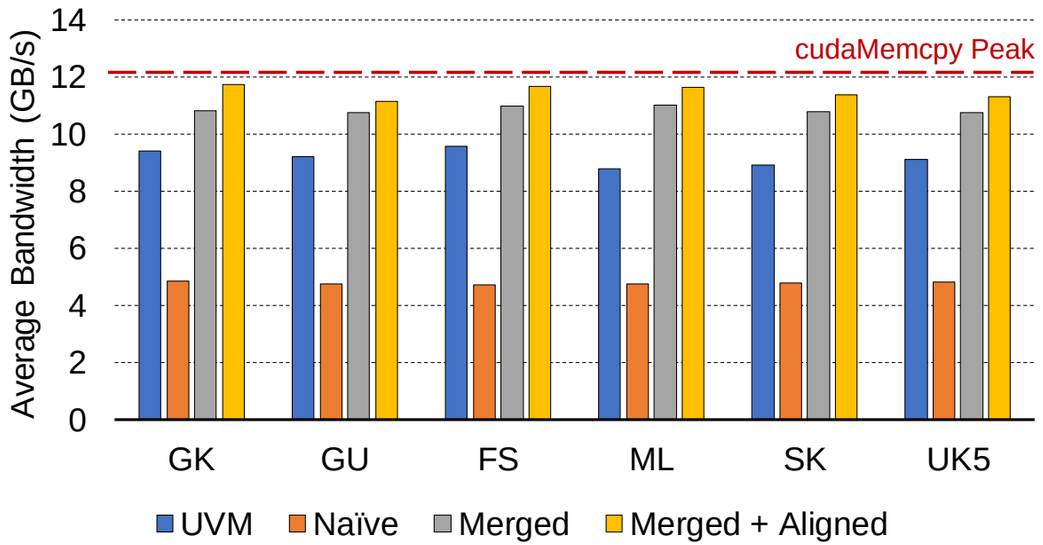
```
1  #define WARP_SIZE 32
2
3  void aligned(*edgeList, *offset, ...) {
4      thread_id = get_thread_id();
5      lane_id = thread_id % WARP_SIZE;
6      // Group by warp
7      warp_id = thread_id / WARP_SIZE;
8      ...
9      start_org = offset[warp_id];
10     // Align starting index to 128-byte boundary
11     start = start_org & ~0xF; // 8-byte data type
12     end = offset[warp_id + 1];
13
14     // Every thread in a warp goes to the same edgelist
15     for (i = start; i < end; i += WARP_SIZE) {
16         // Prevent underflowed accesses
17         if (i >= start_org) {
18             edgeDst = edgeList[i + lane_id];
19             ...
20         }
21     }
22     ...
23 }
```

# EMOGI: BFS case-study

| Sym. | Graph | Number |  | Size (GB) |  |
|---|---|---|---|---|---|
|  |  | $\|V\|$ | $\|E\|$ | $\|E\|$ | $\|w\|$ |
| GK | GAP-kron [10] | 134.2M | 4.22B | 31.5 | 15.7 |
| GU | GAP-urand [10] | 134.2M | 4.29B | 32.0 | 16.0 |
| FS | Friendster [52] | 65.6M | 3.61B | 26.9 | 13.5 |
| ML | MOLIERE_2016 [48] | 30.2M | 6.67B | 49.7 | 24.8 |
| SK | sk-2005 [12–14] | 50.6M | 1.95B | 14.5 | 7.3 |
| UK5 | uk-2007-05 [13, 14] | 105.9M | 3.74B | 27.8 | 13.9 |

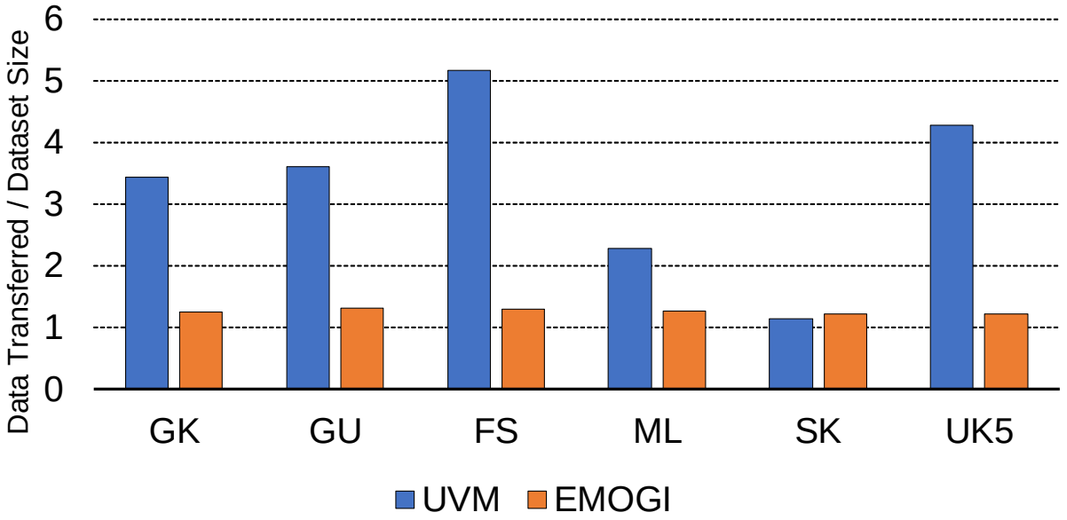



PCIe Gen 3Request Size Distribution

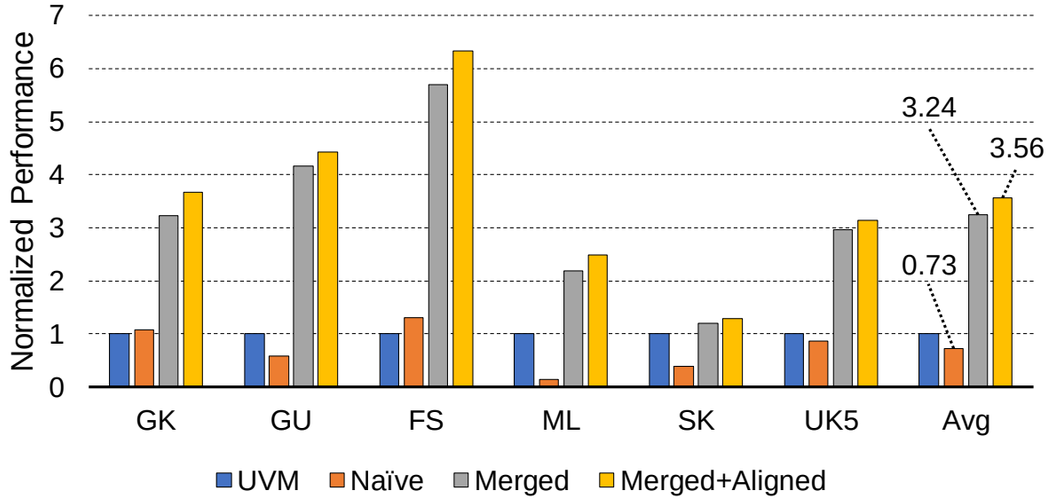



PCIe Gen 3 Bandwidth Utilization

# EMOGI: BFS case-study



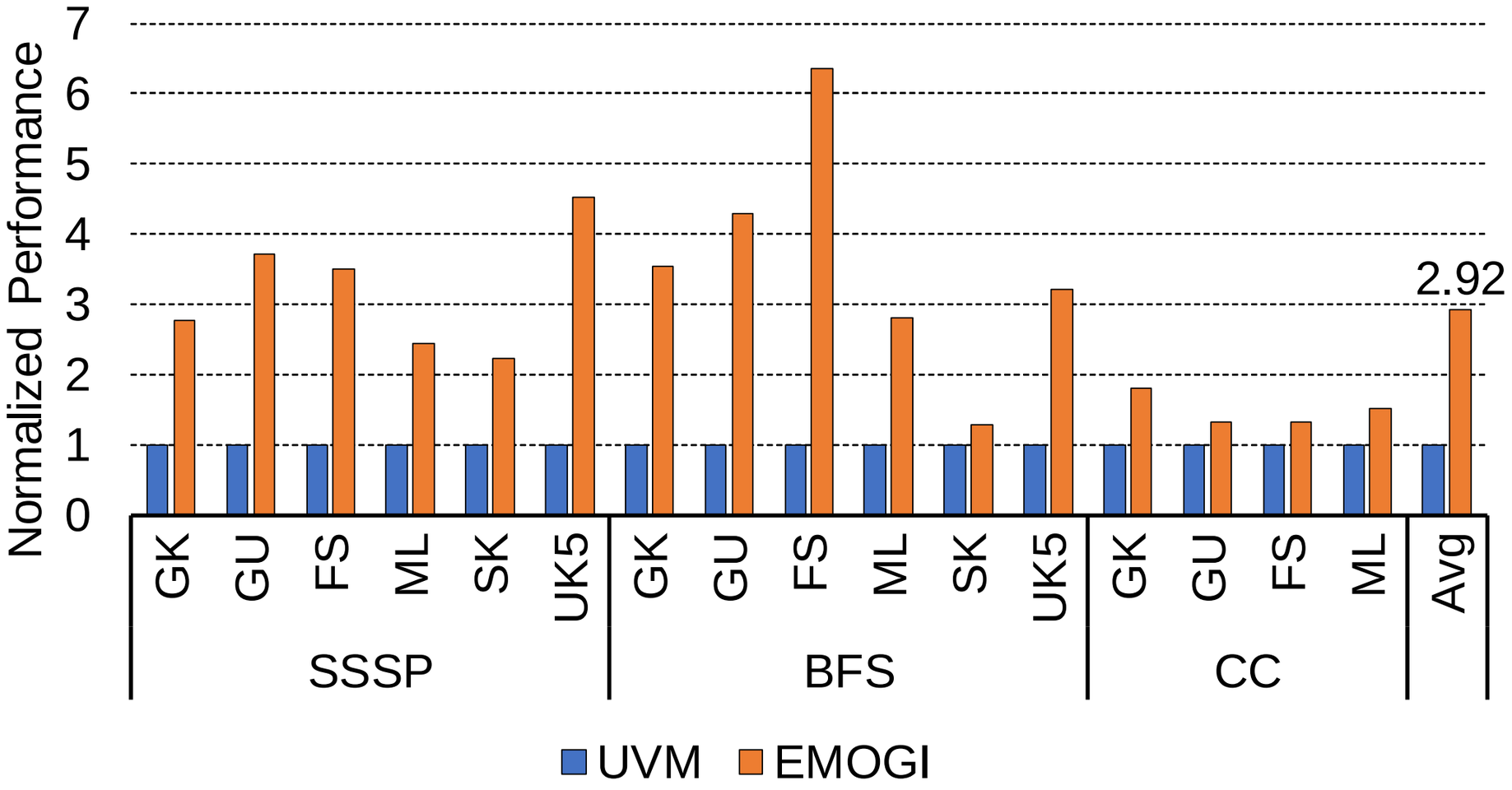

I/O Read Amplification




BFS Execution Speed

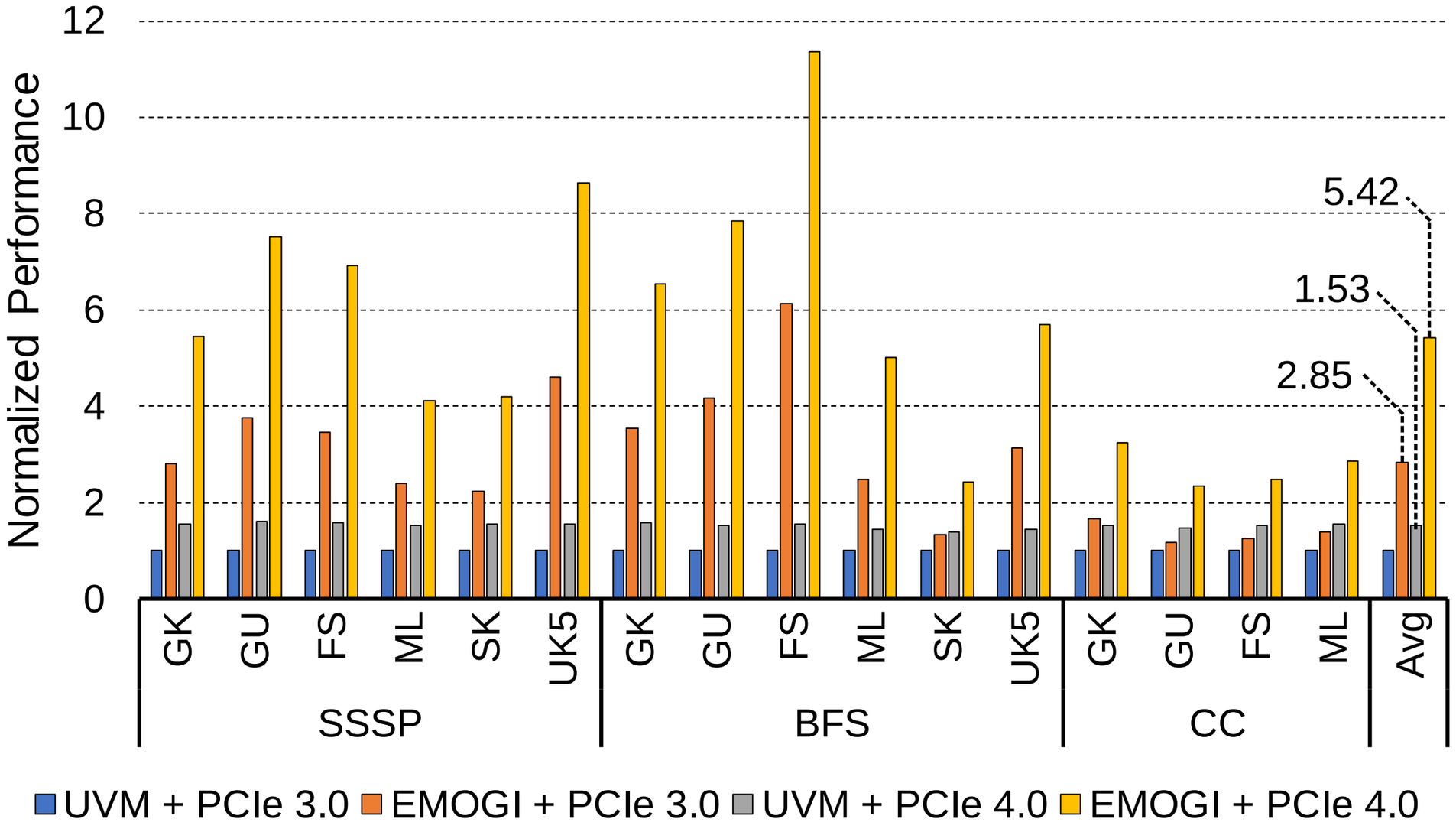
# EMOGI: Overall Performance Comparison

UVM vs. EMOGI
(V100 + PCIe 3.0)



# EMOGI: Overall performance comparison

UVM vs. EMOGI
(A100 + PCIe 3.0 / 4.0)



# Conclusion and Future Work

- Zero-Copy Access Merge+Aligned reduces I/O amplification and make full use of PCIe 3.0 and 4.0

https://arxiv.org/abs/2006.06890

- Workload balancing between long & short neighbor lists
- Utilizing unused GPU memory to enable reuse (e.g. TC)
  - Maintaining software cache
- Neighbor list compression / decompression