Fast GPU Code for Graphs

Wen-mei Hwu1, David Min, Vikram S. Mallthody1, Zaid Qureshi2, Carl Pearson1, Mohammad Almasri1, Omer Anjum1, Rakesh Nagi3, Jinjun Xiong4, Eiman Ebrahimi5

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

A Simplified View of an X86+Volta PCIe System

CPU Host (~1 TFLOPS)

DDR Memory System (~100 GBs)

GPU 1 (~10 TFLOPS)

GPU 2 (~10 TFLOPS)

HBM2 (~16 GBs)

80 GB/s

16 GB/s

16 GB/s

900 GB/s

900 GB/s

< 1 GTEPS

< 100 GTEPS

50 GB/s

16 GB/s

900 GB/s

Electrical & Computer Engineering
Graph traversal

- Start from a certain vertex (or vertices) and keep traversing neighboring vertices iteratively (recursively)
Sparse Access to 1TB in Much Larger Data via 50GB/s PCIe Link

Total Access Time (sec)

- 20 sec
- 33 min
- 1 hour

Factors:
- Speed of Light
- I/O magnification (100X)
- CPU software overhead (2X)
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

128-byte Blocks
L1/L2 Cache
PCIe
(a) Strided
128B 128B
(b) Merged and Aligned
(c) Merged but Misaligned

WARP
32B 32B 32B 32B
0 31 0 31
WARP
96B
32B 32B 32B
WARP
96B
32B 32B 32B

Bandwidth
PCIe
DRAM
(a) Strided
UVM 9.11GB/s
4.74 GB/s
9.61 GB/s
(b) Merged and Aligned
12.23 GB/s
12.36 GB/s
UVM 9.26GB/s
(c) Merged but Misaligned
9.40 GB/s
14.26 GB/s
EMOGI: Zero-copy graph traversal

- Fix the location of large data structures into the host memory, and they are 'never' promoted to the GPU memory
- Apply optimization techniques we learned with the toy example EMOGI: Zero-copy graph traversal

---

Listing 1: Uncoalesced Memory Access

```c
void naive(*edgeList, *offset, ...) {
    thread_id = get_thread_id();
    ...  
    start = offset[thread_id];
    end = offset[thread_id + 1];

    // Each thread loops over a chunk of edge list
    for (i = start; i < end; i++) {
        edgeDst = edgeList[i];
        ...
    }
    ...
}
```

Listing 2: Coalesced Memory Access (Merged + Aligned)

```c
#define WARP_SIZE 32

void aligned(*edgeList, *offset, ...) {
    thread_id = get_thread_id();
    lane_id = thread_id % WARP_SIZE;

    // Group by warp
    warp_id = thread_id / WARP_SIZE;
    ...  
    start_org = offset[warp_id];

    // Align starting index to 128-byte boundary
    start = start_org & ~0xF; // 8-byte data type
    end = offset[warp_id + 1];

    // Every thread in a warp goes to the same edgelist
    for (i = start; i < end; i += WARP_SIZE) {
        // Prevent underflowed accesses
        if (i >= start_org) {
            edgeDst = edgeList[i + lane_id];
            ...
        }
    }
    ...
}
```
EMOGI: BFS case-study

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

- **PCIe Read Request Size Distribution**
- **Average Bandwidth (GB/s)**
- **cudaMemcpy Peak**

Electrical & Computer Engineering
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)

Normalized Performance

SSSP BFS CC

UVM + PCIe 3.0 EMOGI + PCIe 3.0 UVM + PCIe 4.0 EMOGI + PCIe 4.0

5.42 1.53 2.85
Conclusion and Future Work

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

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