A Distributed Multi-GPU System for Fast Graph Processing

Zhihao Jia et al. (2018)

Presented by Edward Fan
Another graph processing framework

- Frameworks we’ve seen so far:
  - Shared memory / disk-based:
    - Ligra, GraphChi, X-Stream
  - Distributed:
    - Pregel, PowerGraph, GraphX
  - Single-machine GPU:
    - Garaph, CuSha, MapGraph
Another graph processing framework - Lux

- Lux: a distributed multi-GPU framework

- Three interesting components:
  - Execution model: push vs pull
  - Use of GPU-specific memory hierarchy
  - Dynamic load balancing based on runtime performance

Figure 1: Multi-CPU node architecture.
Programmer interface

- *init, compute, update*

- Somewhat similar to Pregel's gather-apply-scatter

```java
interface Program(V, E) {
    void init(Vertex v, Vertex v^old);
    void compute(Vertex v, Vertex u^old, Edge e);
    bool update(Vertex v, Vertex v^old);
}
```

*Figure 3: All Lux programs must implement the state-less init, compute and update functions.*
Push execution

- Maintains frontier of vertices to compute on
- Used by many distributed systems - minimizes work

Algorithm 2: Pseudocode for generic push-based execution.

1: while $F \neq \emptyset$ do
2:     for all $v \in V$ do in parallel
3:         init($v, v^{old}$)
4:     end for
5:     end for
6:     for all $u \in F$ do in parallel
7:         for all $v \in N^1(u)$ do in parallel
8:             compute($v, v^{old}$, $(u, v)$)
9:         end for
10:     end for
11:     $F = \emptyset$
12:     for all $v \in V$ do in parallel
13:         if update($v, v^{old}$) then
14:             $F = F \cup \{v\}$
15:         end if
16:     end for
17: end while
Pull execution

- Processes all vertices and edges at each iteration
- Faster on GPUs (except for very sparse updates)

Algorithm 1 Pseudocode for generic pull-based execution.

1: while not halt do
2:   halt = true  \( \triangleright \) halt is a global variable
3:   for all \( v \in V \) do in parallel
4:     init(\( v, v^{old} \))
5:     for all \( u \in N^{-}(v) \) do in parallel
6:       compute(\( v, v^{old}, (u, v) \))
7:   end for
8:   if update(\( v, v^{old} \)) then
9:     halt = false
10: end if
11: end for
12: end while
GPU memory hierarchy

- Three major types of memory:
  - Zero-copy memory: pinned region of DRAM that can be accessed directly
  - GPU device memory: main GPU memory
  - GPU shared memory: small cache shared by all threads (think L1, but if shared by CPU cores)
GPU memory hierarchy

- Goal is to:
  - Minimize transfers from zero-copy memory to device memory
  - Use shared memory as much as possible

- Two optimizations:
  - Load and update vertices only once per iteration
  - Pull execution can put all updates in shared memory

*Figure 9: Data flow for one iteration.*
- **Coalesced memory access**
  - When multiple GPU threads access consecutive addresses, the hardware combines them into one range.
- Next section: assigning consecutive vertices to each GPU means that accesses are consecutive.
- Additional optimization: copy a block to shared memory using coalescing.
Dynamic load balancing

- To start: simple edge partitioning (assign roughly equal number of edges to each GPU; sequentially pick boundary vertices through CSR)

- During each iteration: observe actual runtime to see how much work is in each partition
  - Then, run model to see if inter-node or local repartitioning is worthwhile
  - Seems to converge quickly

Figure 8: The estimates of $f$ over three iterations. The blue squares indicate the actual execution times, while the red circles indicate the split points returned for a partitioning among 4 GPUs at the end of each iteration.
Performance

- Pretty good! Outperforms single-CPU and multi-CPU systems
  - Competitive against single-GPU when run on just 1 GPU
- Arguably, deck is stacked against CPU systems- similar “cost efficiency” numbers, but lots more hardware for Lux
Performance

(a) PR.
(b) CC.

Figure 19: Per iteration runtime on TW with 16 GPUs.

(a) Pull-based executions (PR).
(b) Push-based executions (CC).

Figure 20: Performance model for different executions.
Questions?

Thanks!