Groute: An Asynchronous Multi-GPU Programming Model for Irregular Computations

Tal Ben-Nun, Michæl Sutton, Streepathi Pai, Keshav Pingali

Presented by Jæmin Choi

November 2, 2018
Overview

1. Motivation and Goals
2. Multi-GPU Architecture and Communication
3. Groute Programming Model
4. Implementation Details
   ▶ Distributed Worklists
   ▶ Soft Priority Scheduling
   ▶ Kernel Fusion
5. Performance Evaluation
6. Conclusion
Motivation

- Prevalent method of multi-GPU programming: **Bulk Synchronous Parallel (BSP)**
  - Local computation $\rightarrow$ global communication
  - Underutilization particularly for irregular applications
  - Due to load imbalance and unpredictable communication
- **Asynchronous** programming models to the rescue
  - Processors can compute and communicate autonomously
  - Overlap computation and communication
  - But requires in-depth knowledge of underlying architecture and network
Goals

- Asynchronous programming model + runtime environment
- Provide communication constructs to efficiently express both regular and irregular programs
- Promote load balancing for heterogeneous GPUs
- Outperform existing state-of-the-art implementations (Gunrock, B40C)
Multi-GPU Architecture

Figure: Multi-GPU Node Schematic

1All figures were taken from the paper
Inter-GPU Communication

- **Peer transfer**
  - Host-initiated
  - Executed explicitly

- **Direct access (DA)**
  - Device-initiated
  - Implemented with virtual addressing
  - Performance sensitive to alignment, coalescing, order of access
  - May not be available between all pairs of GPUs
Packetization

- GPUs can only transmit to one destination at a time
- Hinders *responsiveness* of an asynchronous system, especially with large buffers
- Divide messages into **packets**
- Also used in collective communication
- But overhead exists
Packetization

(c) Packetized transfer rate  
(d) Peer broadcast performance

Figure: Inter-GPU Memory Transfer Benchmarks
Groute Programming Model

- Dataflow graph construction + asynchronous computation
- **Endpoint**: a physical device (CPU/GPU) or a router
- **Router**: connects endpoints for dynamic communication
- **Link**: connects two endpoints
- **Routing policy** determines how routers behave
Example: Predicate-Based Filtering

- Filter data based on some condition (i.e. predicate)
- E.g. With a number of particles as input data, give me all the particles whose mass is larger than some threshold
Example: Predicate-Based Filtering

Figure: PBF Dataflow Graph
Example: Predicate-Based Filtering

```
std::vector<T> input = ...;
std::vector<T> output;
int packet_size = ...;

Context ctx;
auto all_gpus = ctx.devices();
int num_gpus = all_gpus.size();

Router h2gpus(1, num_gpus, AnyDevicePolicy);
Router gpus2h(num_gpus, 1, AnyDevicePolicy);

Link dist (HOST, h2gpus, packet_size, 1);
Link collect (gpus2h, HOST, packet_size, 2);

for (device_t dev : all_gpus) {
    std::thread t(WorkerThread,
        Link(h2gpus, dev, packet_size, 2),
        Link(dev, gpus2h, packet_size, 2);
        t.detach();
    )

dist.Send(input, input_size);
dist.Shutdown();

while (true) {
    PendingSegment output_seq = collect.Receive().get();
    if(output_seq.Empty()) break;
    output_seq.Synchronize();
    append(output, output_seq);
    collect.Release(output_seq);
}

//-------------------------------------------
```

Figure: PBF Pseudocode
Distributed Worklists

- Global list of computations (work-items) to process
- Each item may generate new items
- Requires all-to-all communication
Distributed Worklists

- **SplitReceive**
  - Controlled by receive thread
  - High priority for responsiveness
  - Filter/take/pass

- **Local worklist**
  - Controlled by worker thread
  - Lock-free circular buffer
  - Newly generated work → local worklist or remote worklist

- **Remote worklist**: send to next GPU
Soft Priority Scheduling

- Stale information may be propagated due to asynchrony
- Can generate additional intermediate work
- Example: Asynchronous BFS
  - Path with least number of edges is located on a lagging device
  - "Incorrect" path will be used to traverse the graph
  - After the lagging device completes, all traversed values will be recomputed
- Solution: assign soft priorities to each work-item
  - Defer items suspected to generate "useless work"
  - Decreases amount of intermediate work
Kernel Fusion

- Small kernels cause underutilization and increases communication overhead
- Augment worker kernel to include entire control flow and communication with host and other GPUs
  - Includes
    - Determining work-item priorities
    - Processing a batch of work-items in local worklist
    - Running SplitSend
  - Decreases kernel launch overhead in high-diameter graphs
  - Reduces CPU-GPU roundtrips
Performance Evaluation

1. Breadth-First Search (BFS)
2. Single-Source Shortest Path (SSSP)
3. PageRank (PR)
4. Connected Components (CC)

▶ Compared to Gunrock and Back40Computing (B40C)
  ▶ Gunrock: multi-GPU graph analytics library using BSP
  ▶ B40C: state-of-the-art hardcoded BFS

▶ Evaluated on multiple graphs
Graphs

Figure: Graph Properties

- Avg/max degrees vary significantly
- Partitioned using METIS, except *kron21.sym* and *twitter*
Evaluation Environment

Figure: Multi-GPU Node Schematic

- 8-GPU server of 4 dual-board NVIDIA Tesla M60 cards
- 2 8-core Intel Xeon E5-2630 v3 CPUs
- 2 QPI links per CPU for PCI-E switch
**Strong Scaling**

- In communication intensive algorithms (BFS, SSSP), bus topology starts to affect performance when more than the single 4-GPU quadruplet is used.

- Groute mitigates these issues but can still be seen in high-degree graphs such as *soc-LiveJournal1*.
Breadth-First Search (BFS)

**Figure:** BFS Execution Time

- **B40C**
  - Requires direct memory access to all GPUs
  - No METIS partitioning
  - Failed on *twitter* and *kron21.sym*

- **Gunrock**
  - Ran out of memory on *twitter*
  - Produced incorrect results on *kron21.sym* and *soc-LiveJournal1*
Breadth-First Search (BFS)

- Groute significantly outperforms Gunrock in road networks due to kernel fusion
- B40C is faster on soc-LiveJournal1 as it contains a hybrid implementation that switches between kernels
  - Not implemented by Groute due to its highly specialized nature
Single-Source Shortest Path (SSSP)

Figure: SSSP Execution Time

- Groute outperforms Gunrock in all cases except *kron21.sym*
- Asynchrony causes an inflation in number of atomic operations
PageRank (PR)

- Computationally intensive, unlike BFS and SSSP
- Groute outperforms Gunrock on all graphs
- Best scaling achieved with a high ratio of computation to communication (low-degree graphs)

**Figure:** PR Execution Time
Connected Components (CC)

- Topology-driven, not worklist-driven
- Outperforms Gunrock on all counts
- Less memory consumption from not using worklists
- Gunrock runs out of memory

Figure: CC Execution Time
Conclusion

- A robust asynchronous multi-GPU programming model coupled with a runtime environment
- Expressive set of communication primitives capable of expressing both regular and irregular applications
- Outperforms existing graph analytics frameworks
Comments & Discussion

- Requires the programmer to explicitly implement threading (as in pseudocode)
- Limited to a single shared-memory node
- Lacks comparison to CPU-based implementations
- Will the ring topology be scalable in a distributed memory setting?
- All-to-all communication for distributed worklists likely to be a scalability bottleneck
- Soft priority scheduling: how do we know if items are likely to generate "useless work"?
- Load balancing policy described in the paper is basically ‘first available device’; how does Groute adapt to changing load during runtime? (E.g. imbalance in the number of generated work-items per GPU)
Thank You