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

Tal Ben-Nun[†], **Michael Sutton**[†], Sreepathi Pai[‡], and Keshav Pingali[‡]

[†] The Hebrew University of Jerusalem, Israel

[‡]University of Texas at Austin

PPoPP 2017, Austin, TX

Outline

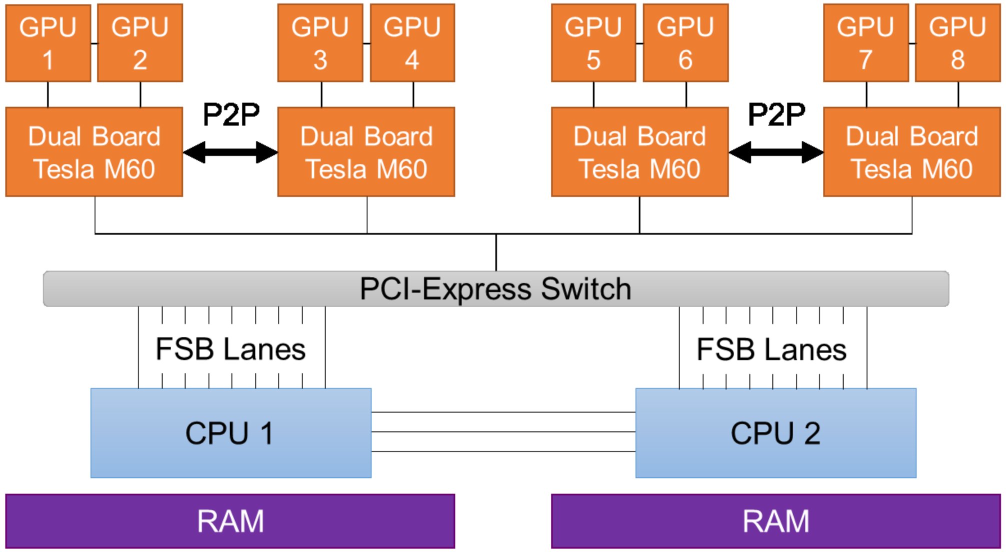
- Motivation
- Groute
  - Programming model
  - Implementation
- Experimental Evaluation
- Conclusions

Outline

- **Motivation**
- Groute
  - Programming model
  - Implementation
- Experimental Evaluation
- Conclusions

Multi-GPU Architecture

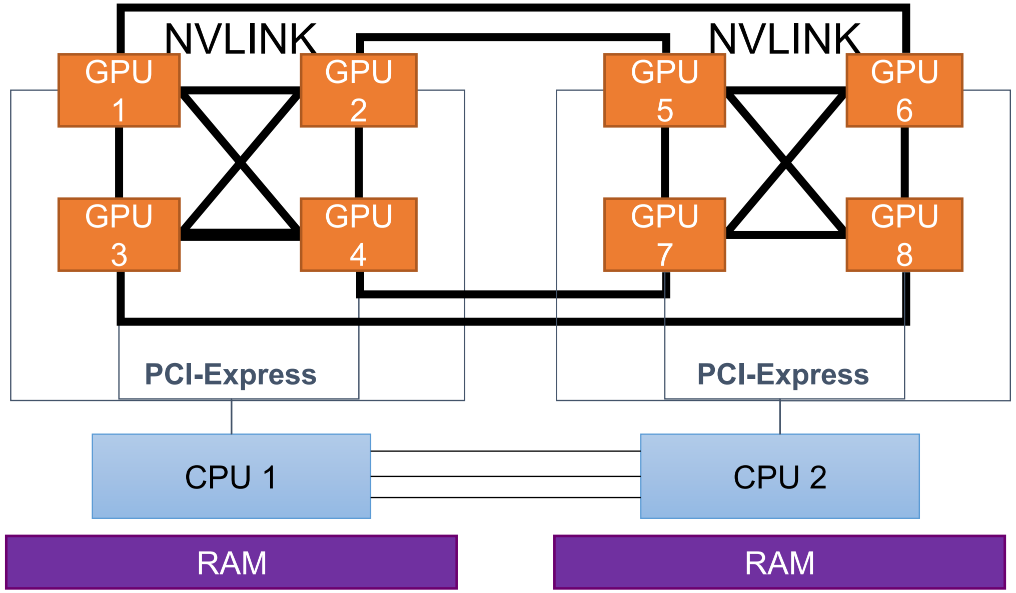
- Small number of devices (~1-25)
- Low-latency communication
- Large unified memory space
- Wide variation in topology



HUJI Cortex Cluster Node

Multi-GPU Architecture

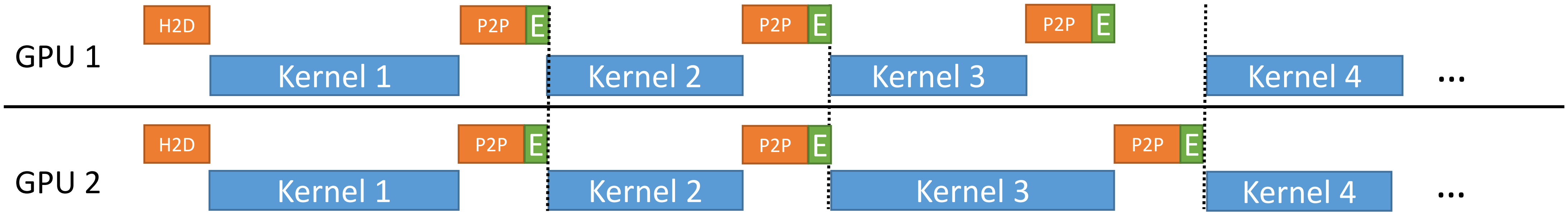
- Small number of devices (~1-25)
- Low-latency communication
- Large unified memory space
- Wide variation in topology



NVIDIA DGX-1

Multi-GPU Programming

- Two synchronization constructs: *streams* and *events*
- Current frameworks use BSP-style programming
  - Global synchronization
  - Peer memory transfers: MGPU, MAPS-Multi, Gunrock
  - Inter-GPU direct memory access: Back40Computing (B40C)

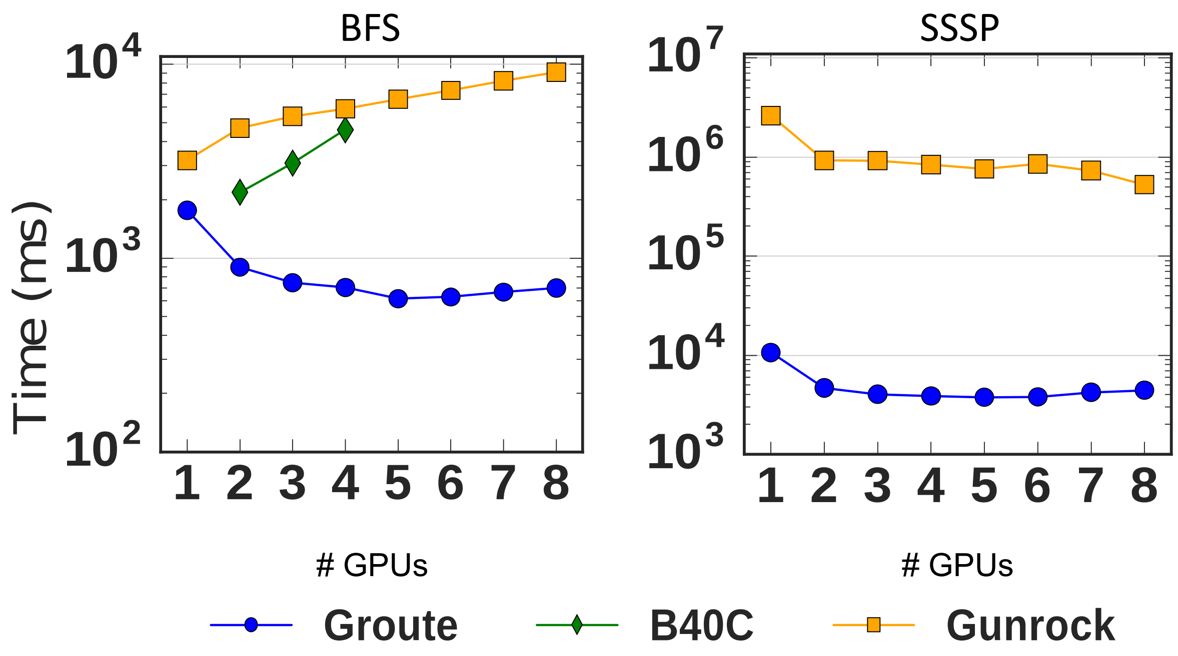


Irregular Applications

- Characterized by unpredictable memory transfers
- Global synchronization leads to utilization gaps
- Direct memory access does not scale well
    - See paper for micro-benchmark results

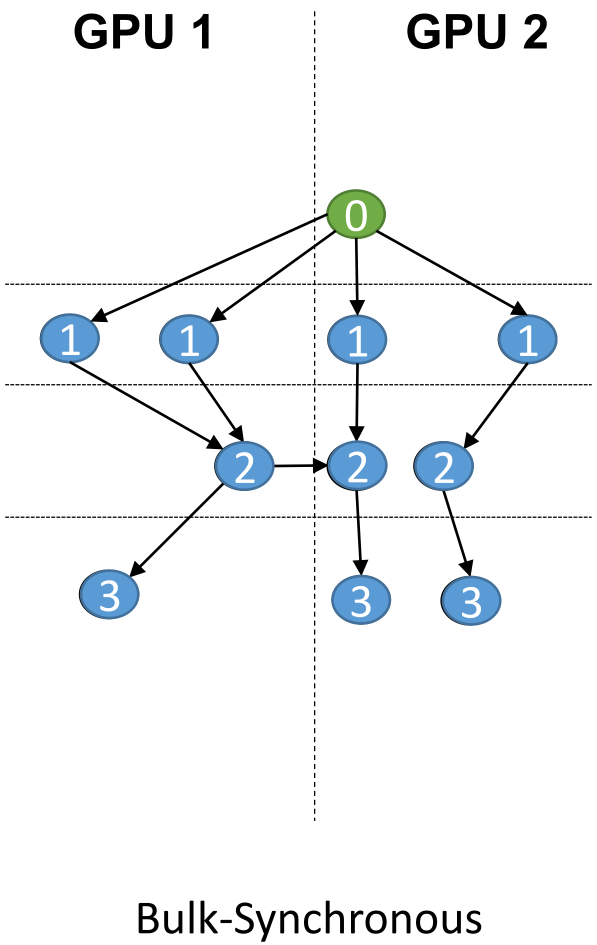
Asynchronous Multi-GPU Programming

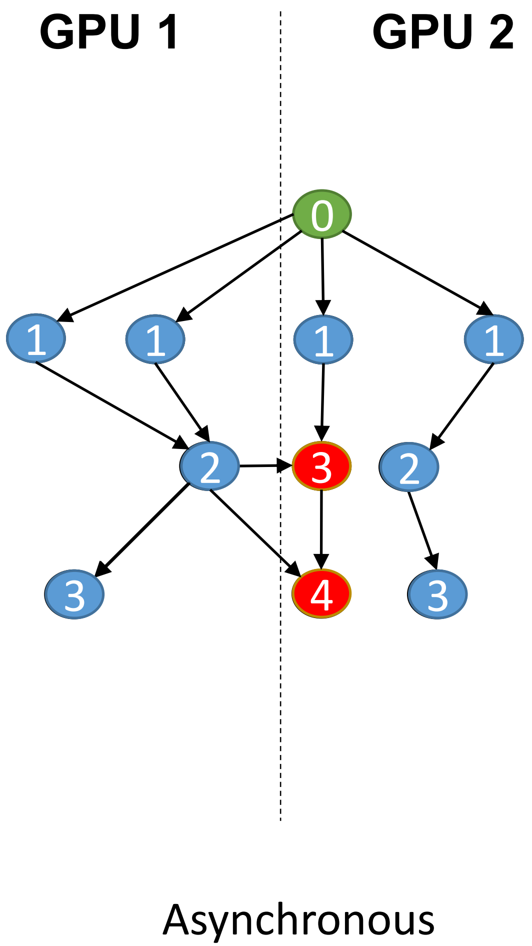
- Asynchronous multi-GPU programming can eliminate utilization gaps
- Requires:
  - Fine-grained synchronization
  - Interleaved communication
  - Device availability indication
- Entry barrier for asynchronous programming is very high, even for skilled developers



Asynchronous Scaling Challenges

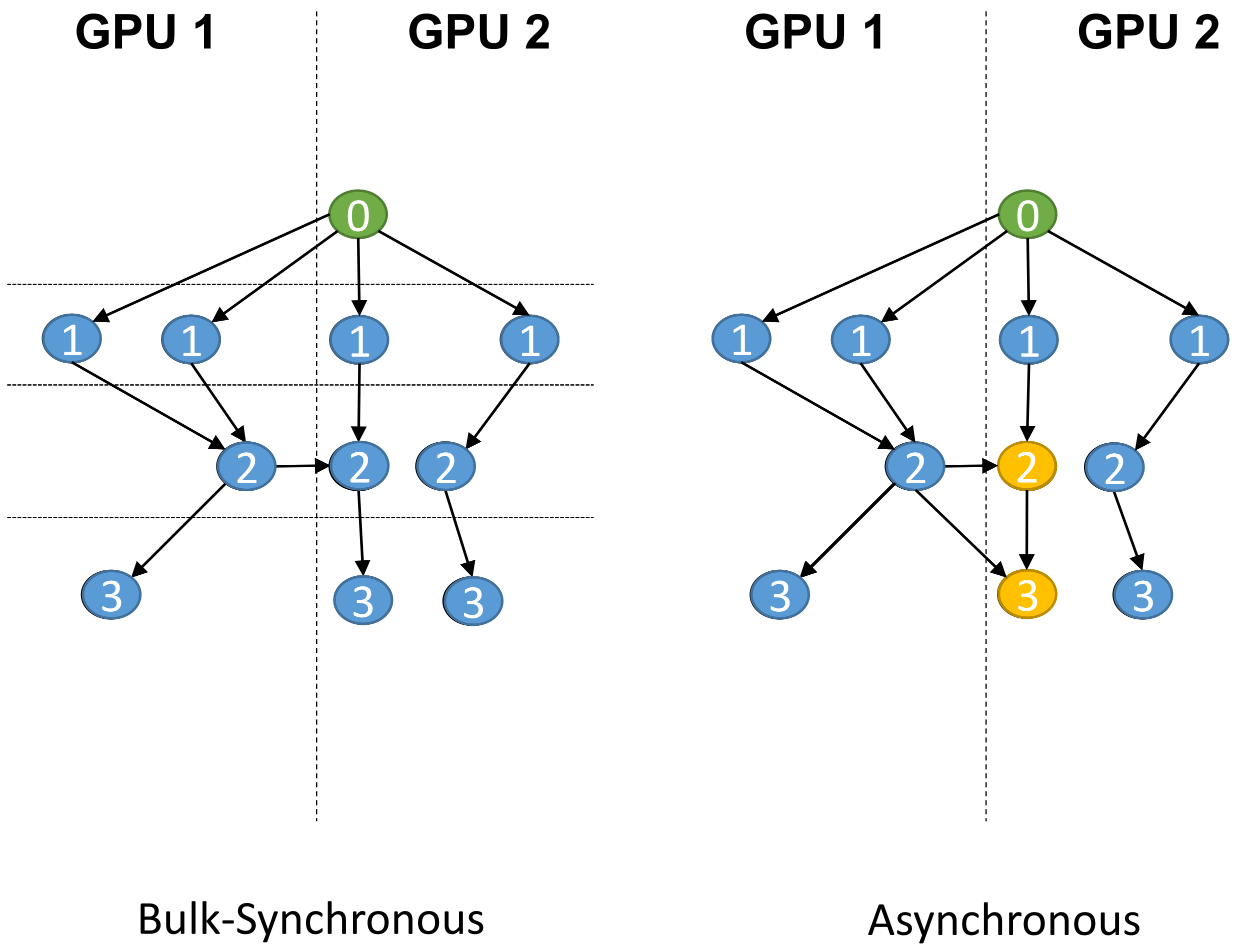
- Work explosion:
  Asynchronous processing can generate “useless” work
- Causes:
  - Lack of global state management
  - Some devices may be faster
  - Problem structure and partitioning
- May execute billions of additional work items





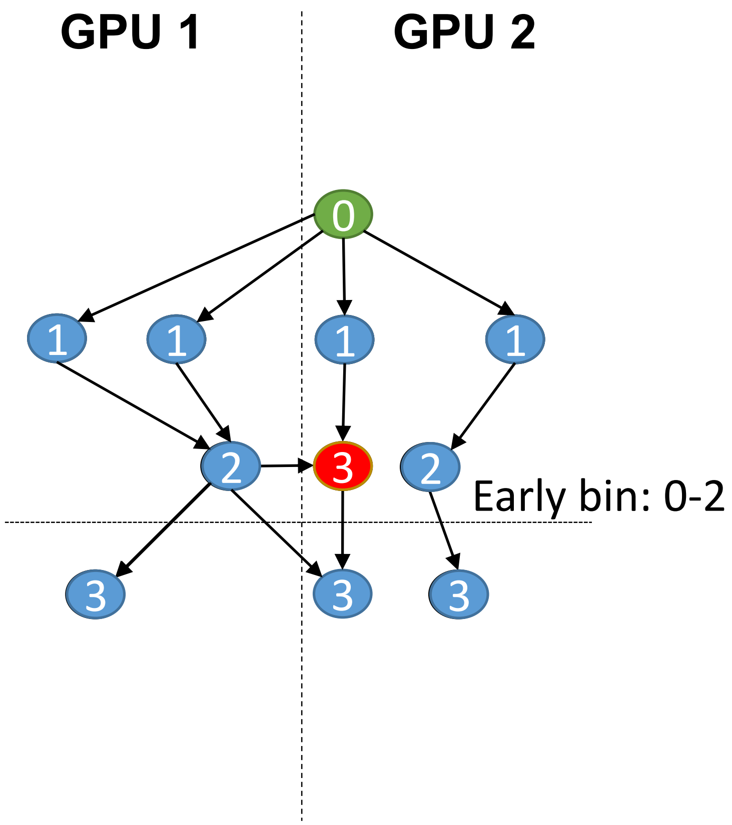
Asynchronous Scaling Challenges

- Work explosion: Asynchronous processing can generate "useless" work
- Causes:
  - Lack of global state management
  - Some devices may be faster
  - Problem structure and partitioning
- May execute billions of additional work items



Soft Priority Scheduling

- Addresses the work explosion problem
- Assign priorities to work-items
- Bin work-items by priority into early and late bins
- Complete early bin before advancing to late bin
  - Like SSSP delta-stepping
- Requires global consensus on advancing to late bin

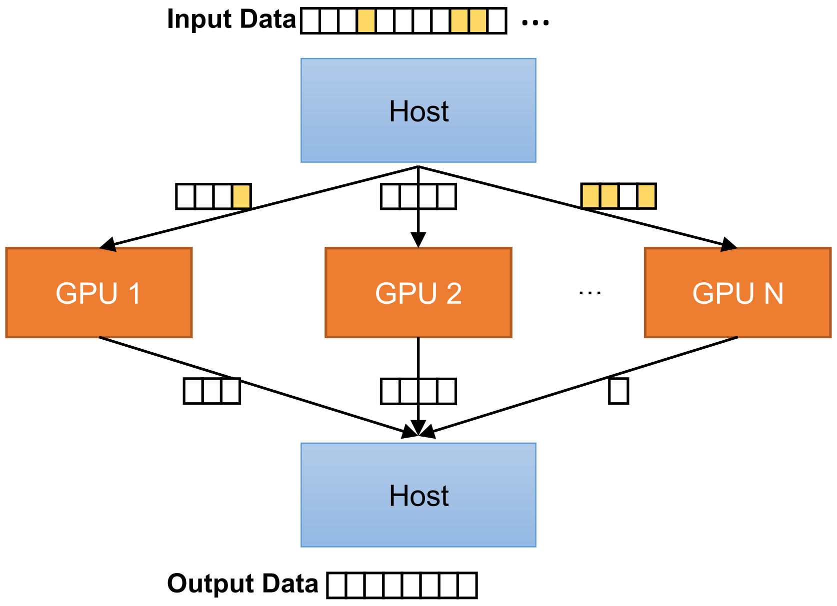


Outline

- Motivation
- **Groute**
  - Programming model
  - Implementation
- Experimental Evaluation
- Conclusions

Predicate-Based Filtering (PBF)

- Host scatters segments of input array to GPUs
- GPUs filter elements that match predicate
- Host gathers filtered data from GPUs
- Challenge: load balancing GPU work



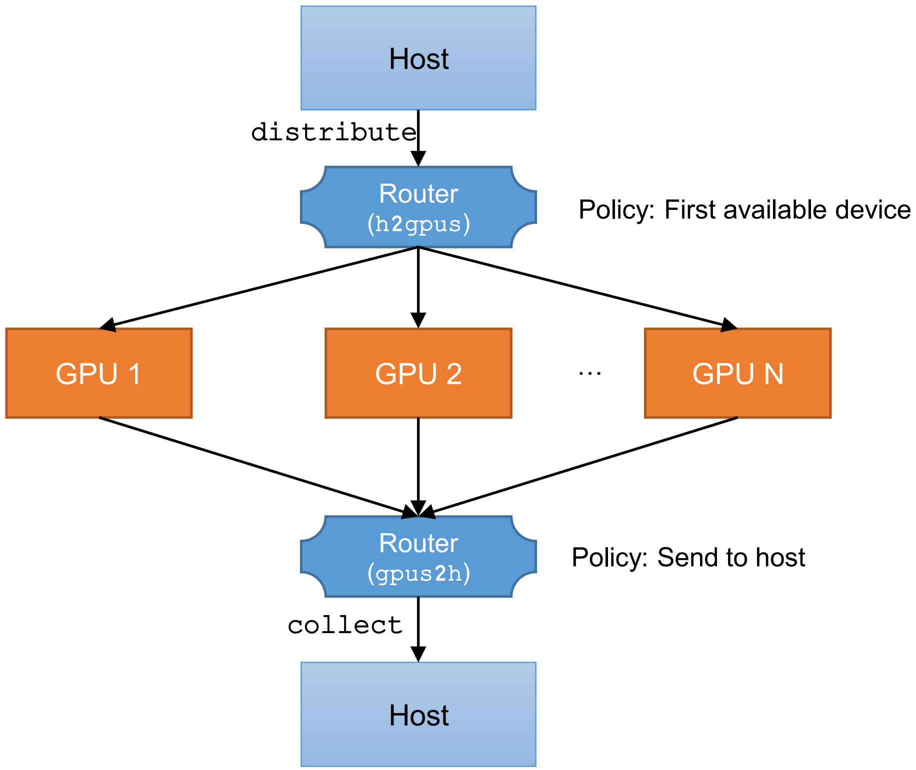
Predicate-Based Filtering Dataflow

Groute Programming Model

Groute programs consist of two parts:

- **Dataflow graph construction**
  - Create links and routers
  - Create management threads

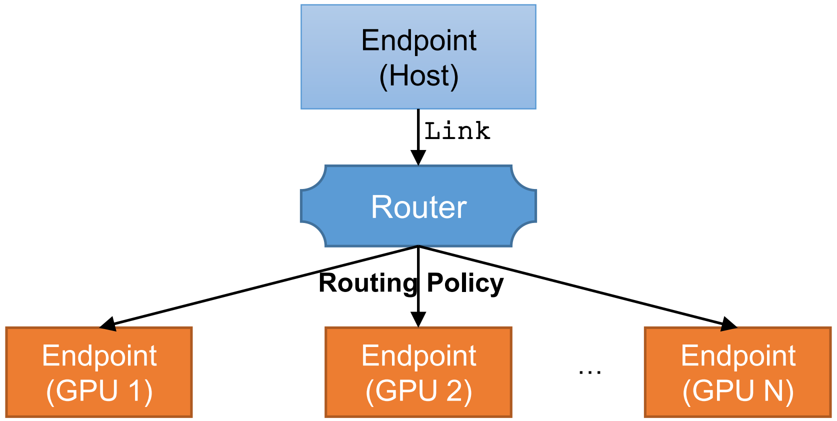
- **Asynchronous execution**
  - Send data through links
  - Process received data asynchronously



Predicate-Based Filtering

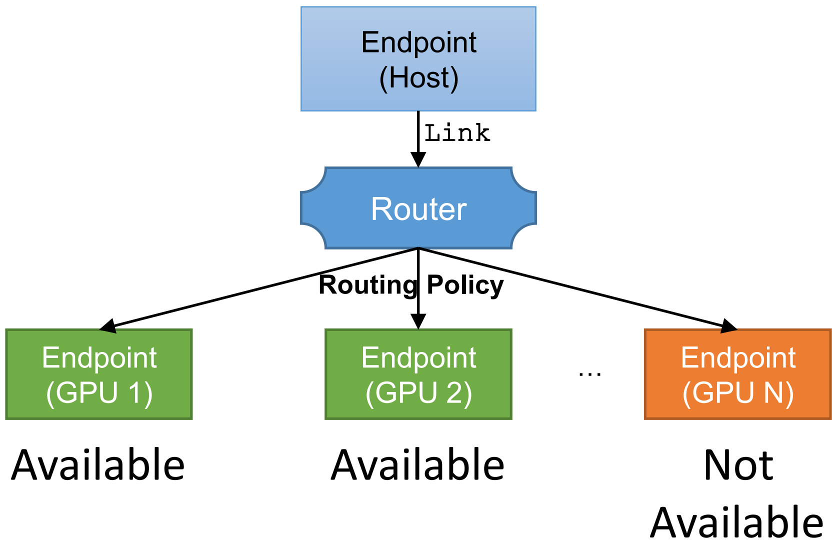
Groute Dataflow Graph Construction

- **Endpoint**: An entity that can communicate
- **Link**: Connects two Endpoints, data is transferred in pipelined packets
- **Router**: Connects multiple Endpoints for dynamic communication
- **Routing Policy**: Determines destination based on source and contents



Groute Routers

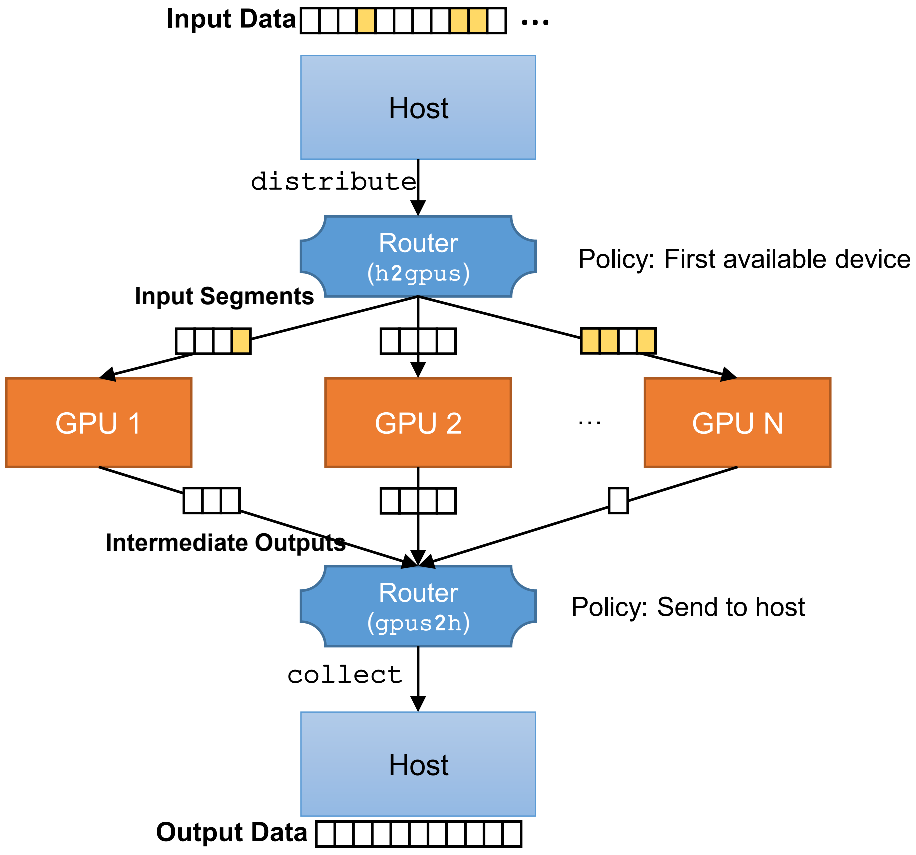
- Routers enable runtime decisions based on policy and availability
- Routing policies express different semantics:
  - Scatter, reduce, all-to-all
- Policy details can be tuned with respect to topology
- Load balancing in PBF implemented by routers



PBF: Asynchronous Execution

```
void HostThread() {
    std::vector<T> input = ...;
    std::vector<T> output;
    dist.Send(input, input_size);
    dist.Shutdown();
    while(true) {
        PendingSegment output_seg =
          collect.Receive().get();
        if(output_seg.Empty()) break;
        output_seg.Synchronize();
        append(output, output_seg);
        collect.Release(output_seg);
    }
}
```

```
void WorkerThread(device_t dev, Link in,
                  Link out) {
    Stream stream (dev);
    T *s_out = ...;
    int *out_size = ...;
    while(true) {
        PendingSegment seg =
          in.Receive().get();
        if(seg.Empty()) break;
        seg.Synchronize(stream);
        Filter<<<...,stream>>>(seg.Ptr(),
          seg.Size(), s_out, out_size);
        in.Release(seg, stream);
        out.Send(s_out, out_size, stream);
    }
    out.Shutdown();
}
```



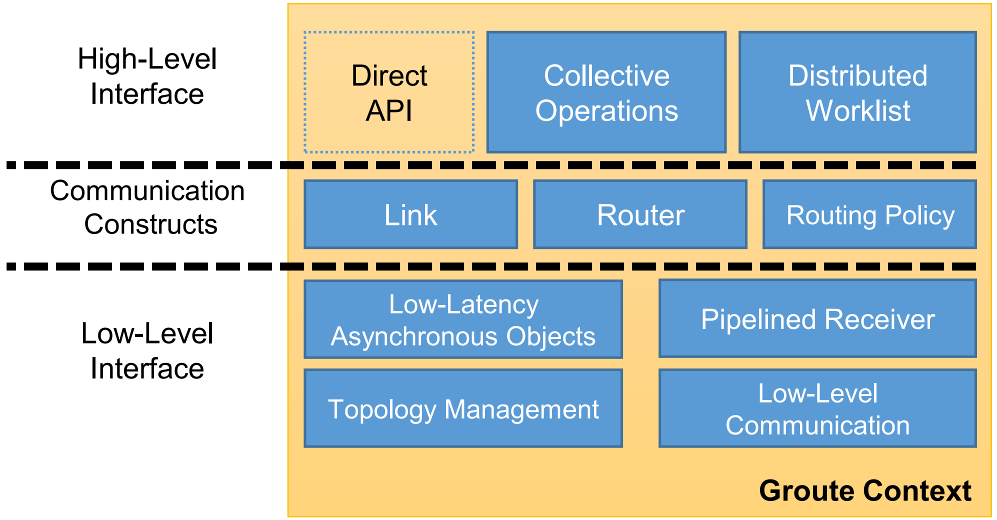
Predicate-Based Filtering

Groute Implementation

- Three-layered runtime environment
  - Each layer can be skipped for increased programmer control

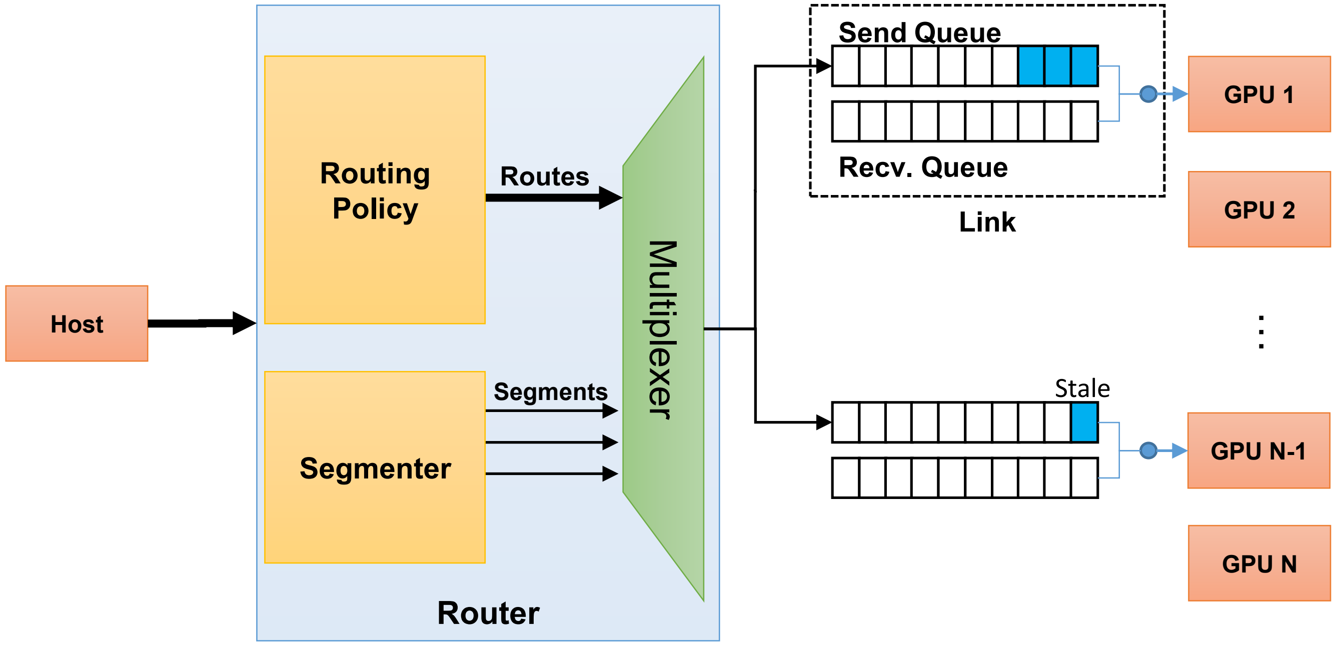
- Bus topology aware (e.g. PCIe, NVLINK)

- Standard C++11 over CUDA, open source



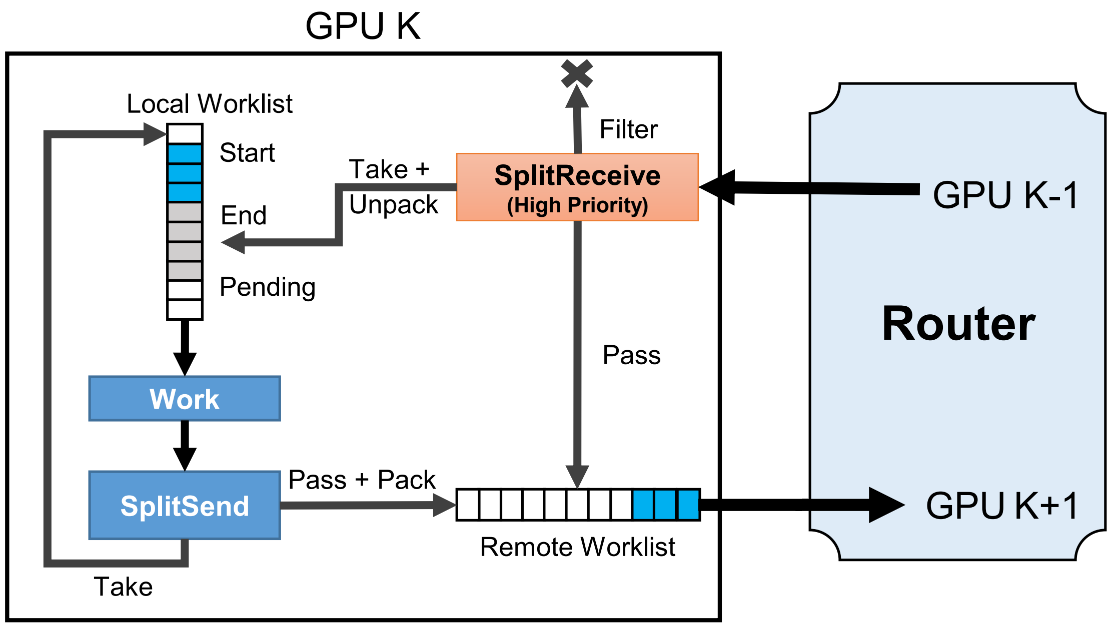
Router Implementation

- Links enable asynchronous memory transfers
  - Pipelining, workspace allocation, availability indication
- Router packetizes messages and determines destination devices
- Scalable implementation
  - No global locking



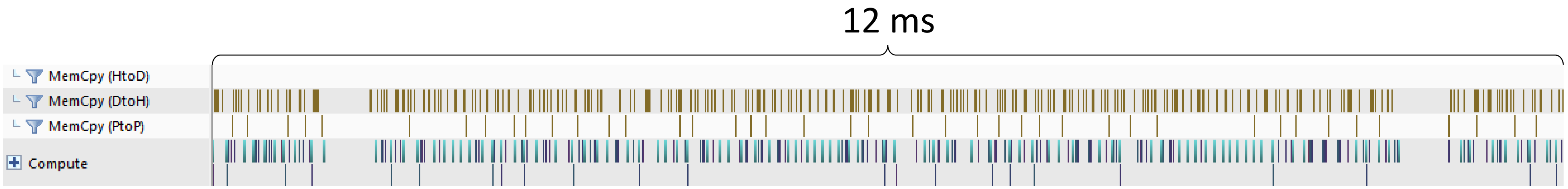
Worklist-Based Graph Algorithms

- Graph nodes partitioned between GPUs, halo added automatically
- Distributed worklist
  - Implemented over router using ring topology
  - Enables fine-grained all-to-all communication
  - Customized through programmer provided callbacks
  - Supports soft-priority scheduling

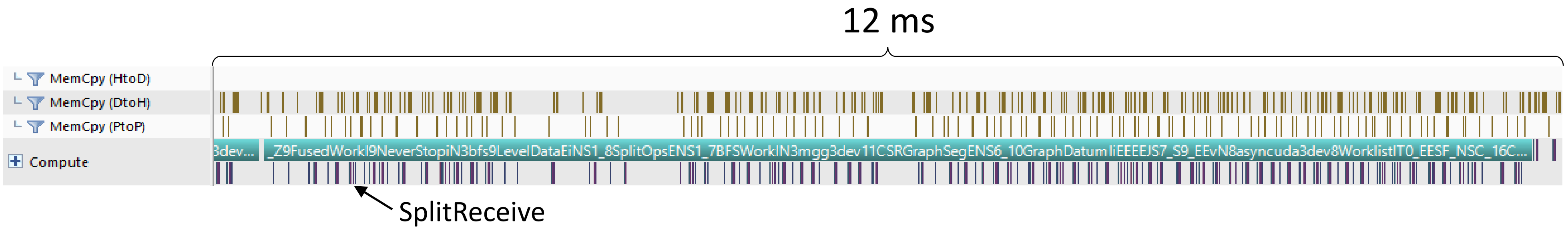


Distributed Worklist – Kernel Fusion

- Problem: Short kernel lifespan
  - Overhead incurred by kernel launches, CPU-GPU roundtrips



- Solution: Worker Fusion
  - Fuse control-flow, work-item processing and communication
  - One multiprocessor is kept free for receiving data



Outline

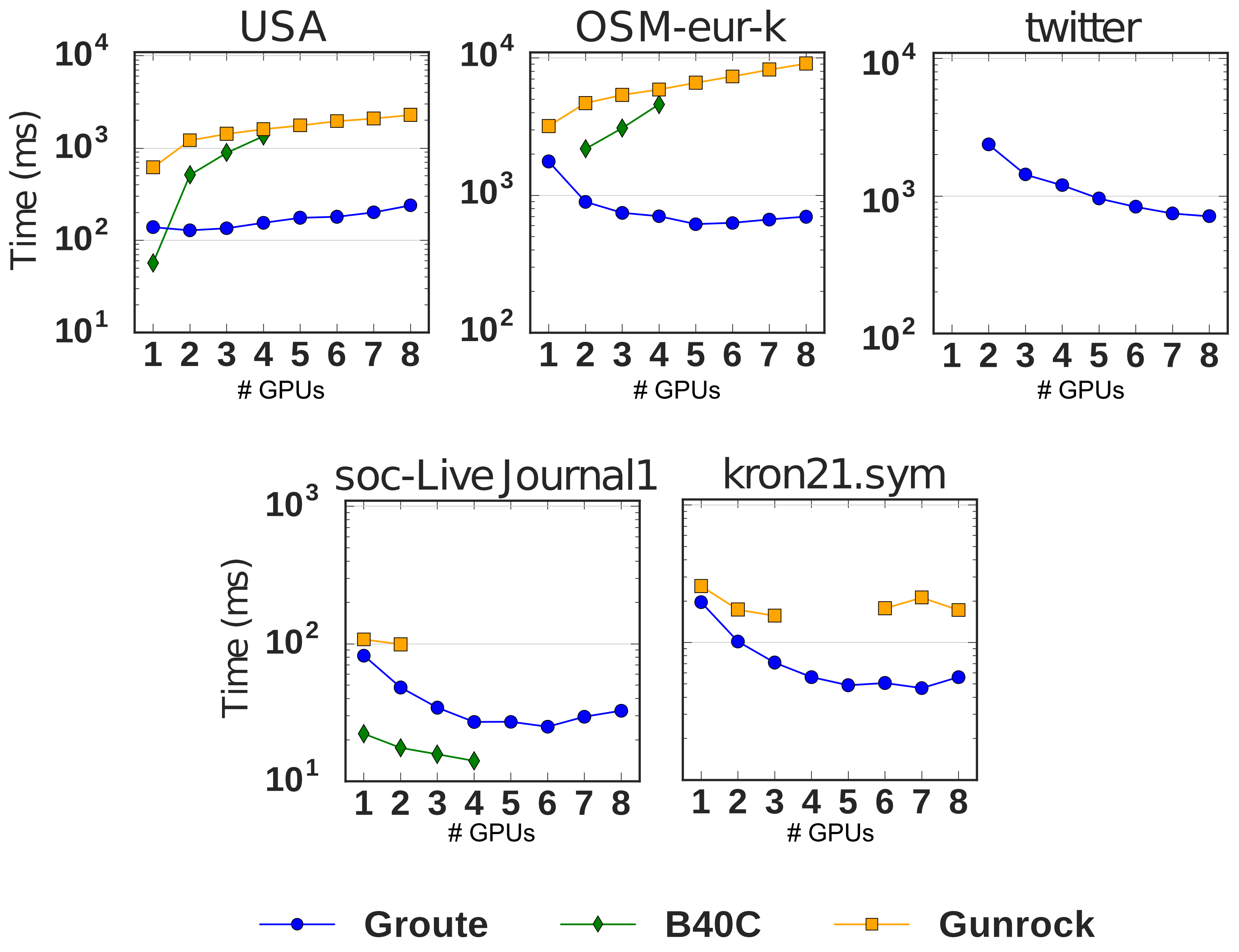
- Motivation
- Groute
    - Programming model
    - Implementation
- **Experimental Evaluation**
- Conclusions

Experimental Setup

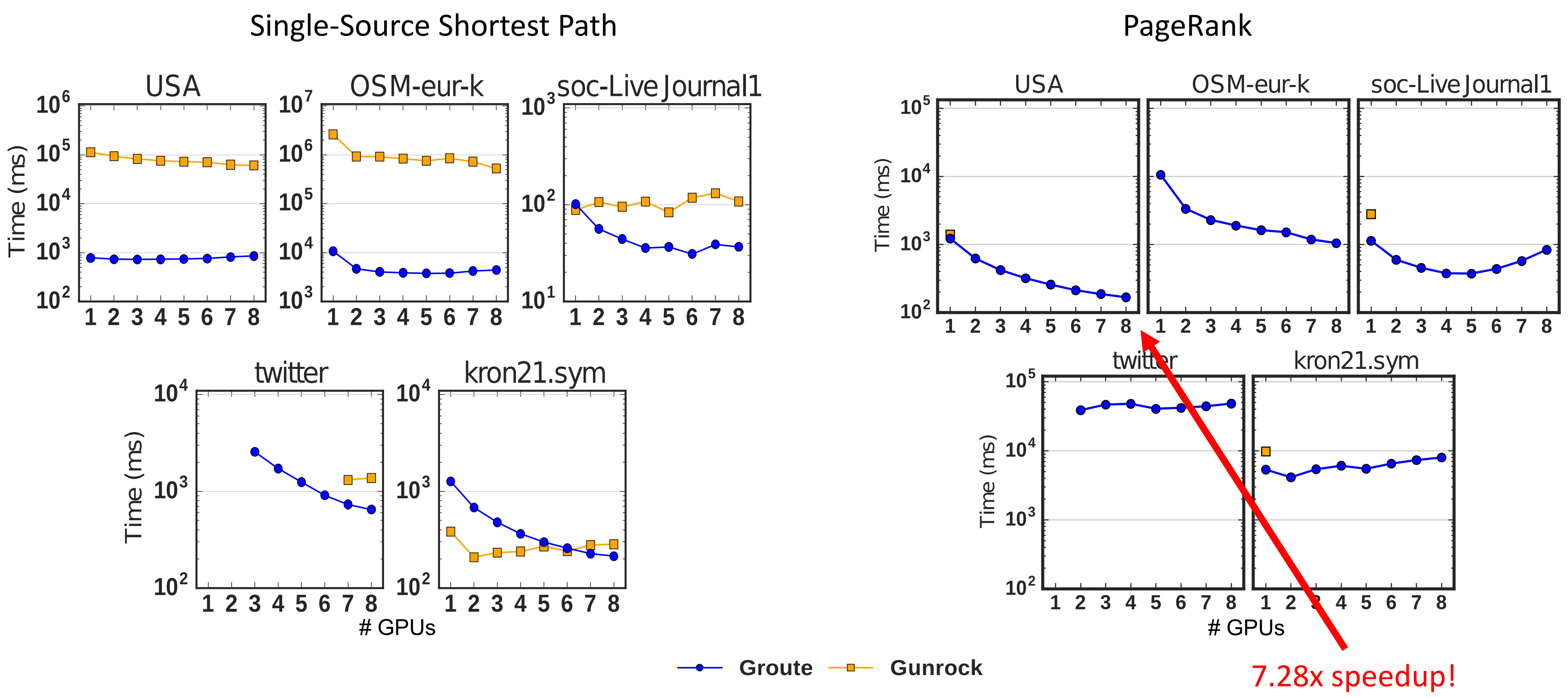
- Systems:
  - HUJI Cortex Cluster (8 Tesla M60 GPUs per node)
  - Heterogeneous node with a Quadro M4000 and a Tesla K40c
- Algorithms:
  - Breadth-First Search (data-driven)
  - Single-Source Shortest Path (delta-stepping)
  - PageRank (data-driven, push-based)
  - Connected Components (pointer-jumping)
  - Predicate-Based Filtering
- Datasets:
  - Road Networks: USA, OSM Europe
  - Social Networks: LiveJournal, Twitter
  - Synthetic Graphs: Kronecker Graph (logn21)

Breadth-First Search Performance

- Implementation is compared with:
  - Gunrock: Frontier-based bulk-synchronous
  - B40C: hardwired BFS implementation, uses direct memory access, limited to 4 GPUs

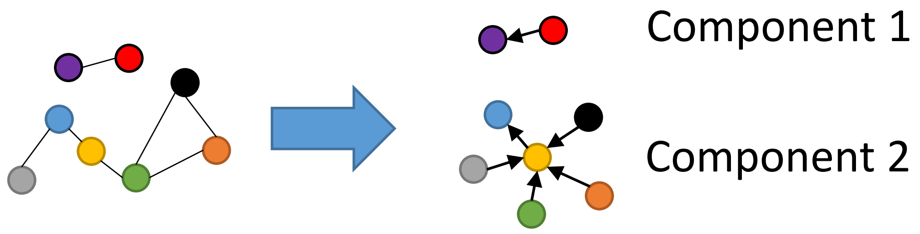

USA
OSM-eur-k
twitter
soc-LiveJournal1
kron21.sym
Time (ms)
GPUs
10^1
10^2
10^3
10^4
1 2 3 4 5 6 7 8
Groute
B40C
Gunrock

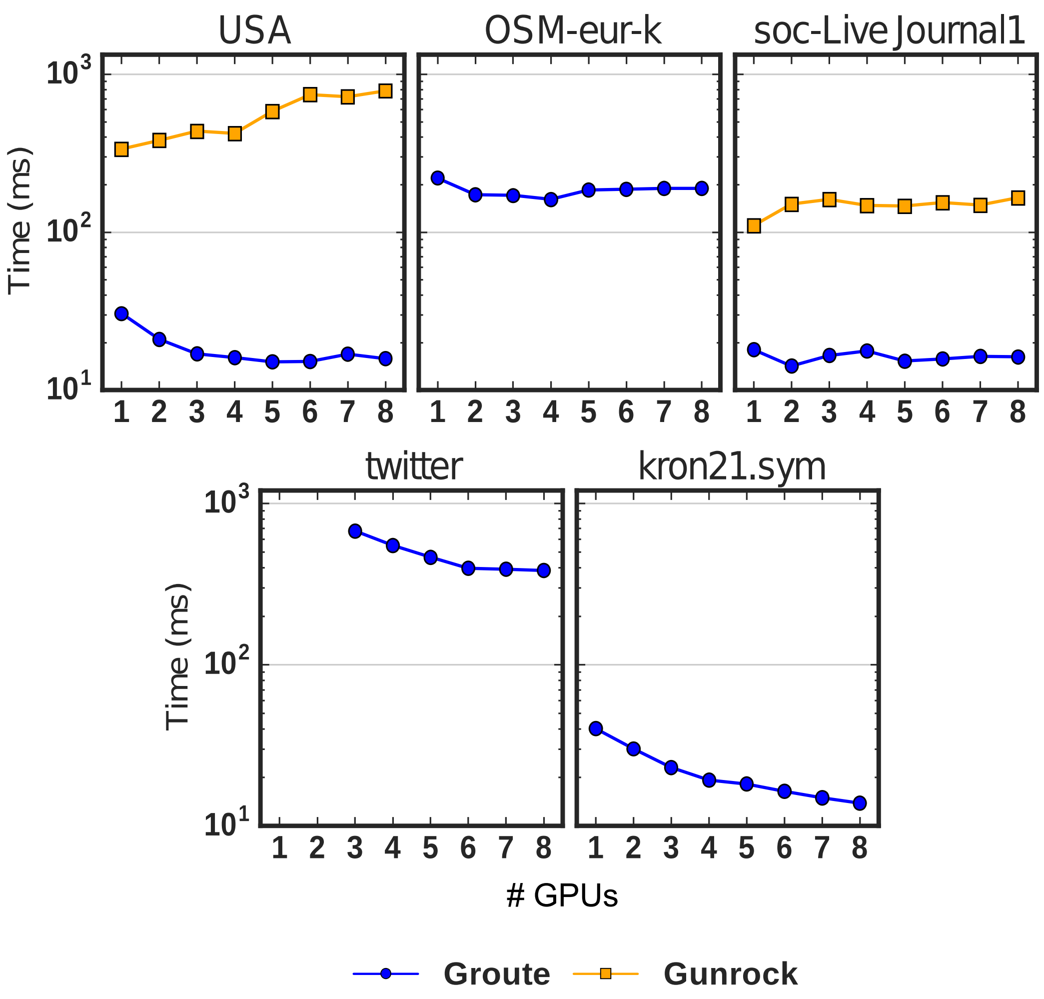
SSSP and PageRank Performance



Connected Components Performance

- Pointer-jumping algorithm
  - *Not* a vertex program
  - Topology-driven, highly irregular
- Based on "compression" of graph to depth-1 trees:





Performance Summary

Best Runtime:

Graph	BFS [ms]			SSSP [ms]		PR [ms]		CC [ms]	
	Gunrock	B40C	Groute	Gunrock	Groute	Gunrock	Groute	Gunrock	Groute
USA	617.85	**56.83**	128.38	60,656.91	**725.93**	1,394.25	**167.89**	335.65	**15.11**
OSM-eur-k	3,191.78	2,177.1	**616.4**	874,083.5	**3,513.29**	—	**1,045.33**	—	**160.96**
soc-LiveJournal1	99.11	**14.07**	24.96	83.36	**30.98**	2,782.06	**371.71**	110.05	**14.19**
Twitter	—	—	**713.6**	1,310.7	**649.2**	—	**38,549.27**	—	**384.13**
Kron21.sym	156.68	—	**46.55**	**208.43**	213.92	9,800.43	**5,342.73**	—	**13.86**

Best Groute Scaling:

Graph	BFS	SSSP	PR	CC
USA	1.08x	1.07x	**7.28x**	1.93x
OSM-eur-k	2.86x	2.83x	3.19x *(over 2 GPUs)*	1.36x
soc-LiveJournal1	3.28x	3.27x	3.05x	1.27x
Twitter	3.32x *(over 2 GPUs)*	3.93x *(over 3 GPUs)*	0.95x *(over 2 GPUs)*	1.75x *(over 3 GPUs)*
Kron21.sym	**4.21x**	**5.93x**	1.29x	**2.90x**

Outline

- Motivation
- Groute
  - Programming model
  - Implementation
- Experimental Evaluation
- **Conclusions**

Conclusions

- Groute programming model is simple and expressive
- Framework hides per-platform tuning
- High-level constructs overcome asynchronous programming pitfalls
- Exceeds state-of-the-art performance

Future Research

- Programming model application to distributed systems
- Dynamic vertex ownership in traversal algorithms for load balancing
- Complete device-level communication management

G route

Groute is open-source and available at:
http://www.github.com/groute/groute

Thank you

Questions?

This research was supported by the German Research Foundation (DFG) Priority Program "Software for Exascale Computing" (SPP-EXA), research project FFMK; by NSF grants 1218568, 1337281, 1406355, and 1618425; and by DARPA BRASS contract 750-16-2-0004.