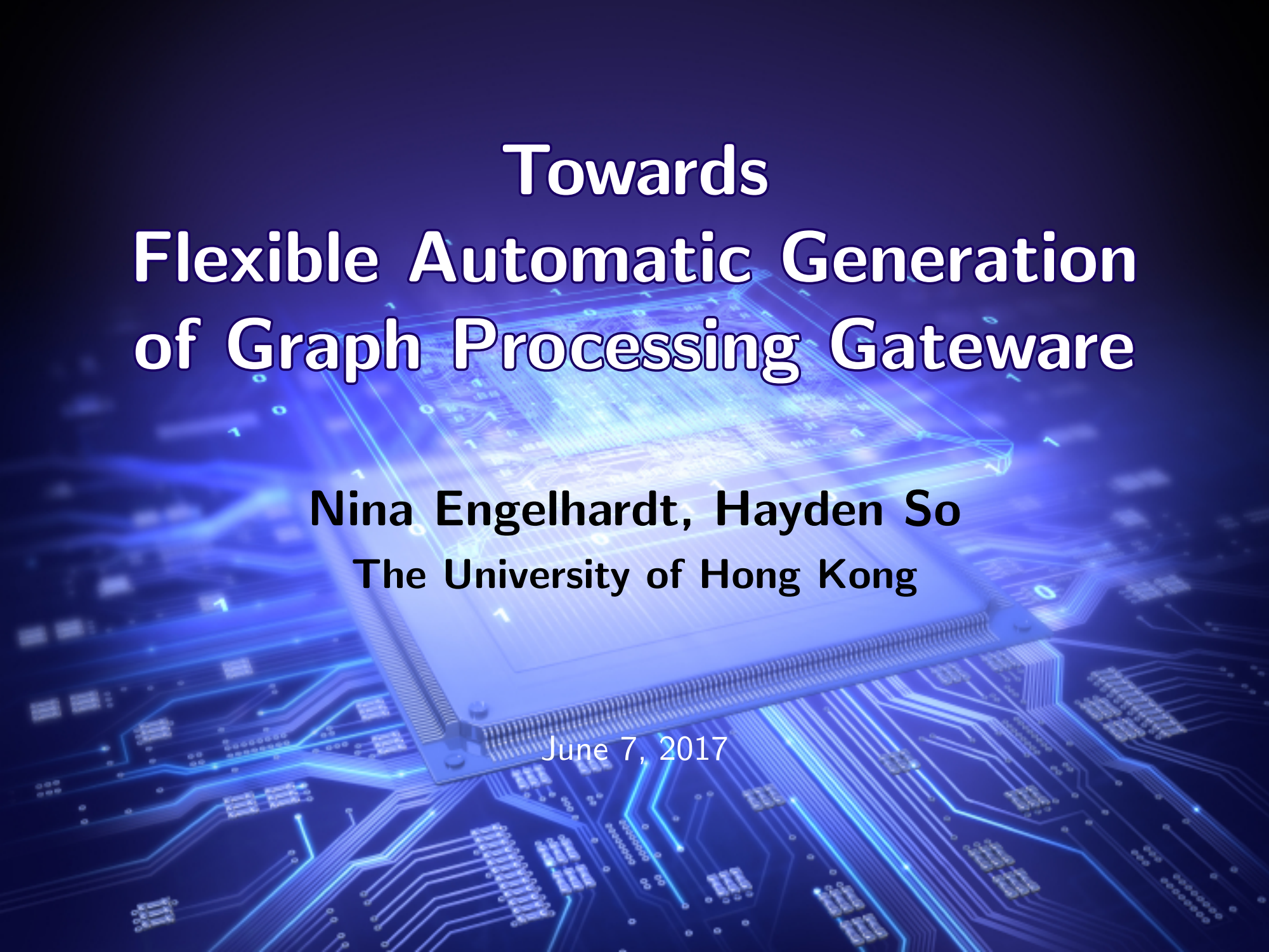


Towards Flexible Automatic Generation of Graph Processing Gateware

Nina Engelhardt, Hayden So

The University of Hong Kong

June 7, 2017

GraVF Graph Processing Framework

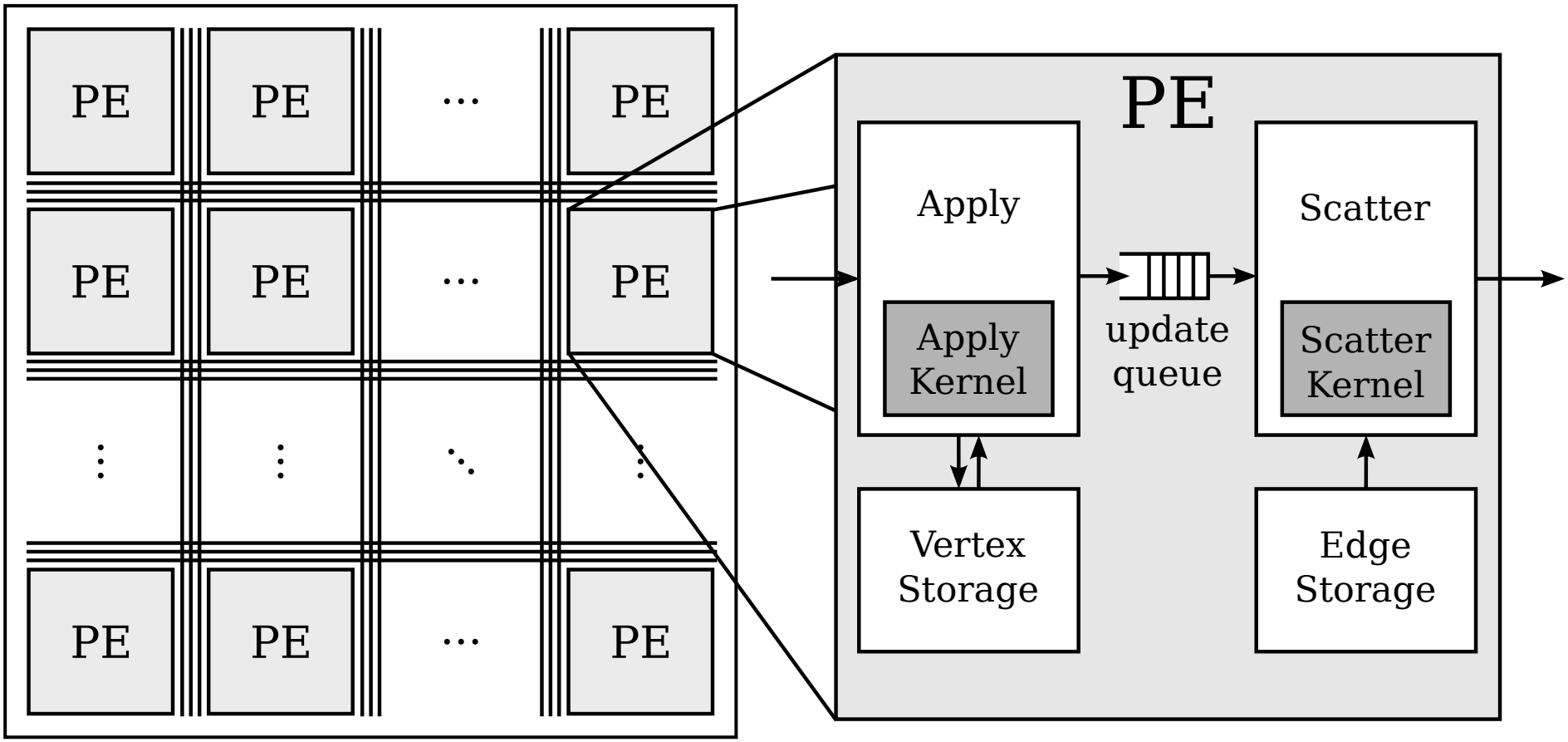
- Introduced last year [HEART 2016, FPL 2016]
- Several improvements now:
  - Off-Chip Memory (HMC)
  - Cleaner Synchronization/Network Protocol
  - Better Balancing/Partitioning

GraVF Overview

Synchronous Vertex-Centric Programming Model

- Algorithm is a *vertex kernel* operating from the POV of a vertex
- Computation is organized in *supersteps*
- Each superstep, all active vertices execute the vertex kernel concurrently, followed by global barrier
- Vertices exchange messages along (outgoing) edges
- Messages sent during superstep n will be received during superstep $n + 1$
- In GraVF: kernels divided in two phases *apply* and *scatter*

Architecture



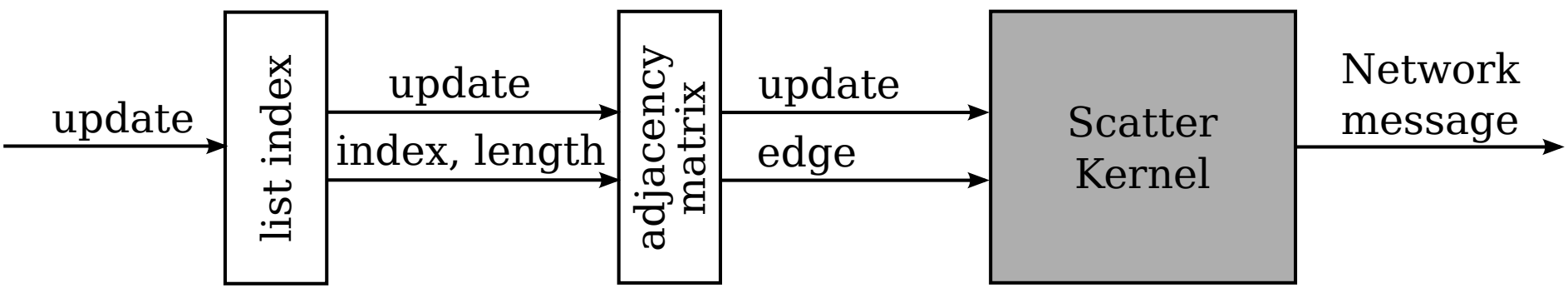
Dark Gray: user-provided algorithm-specific module

Using off-chip HMC memory to increase capacity

Approach to using off-chip memory

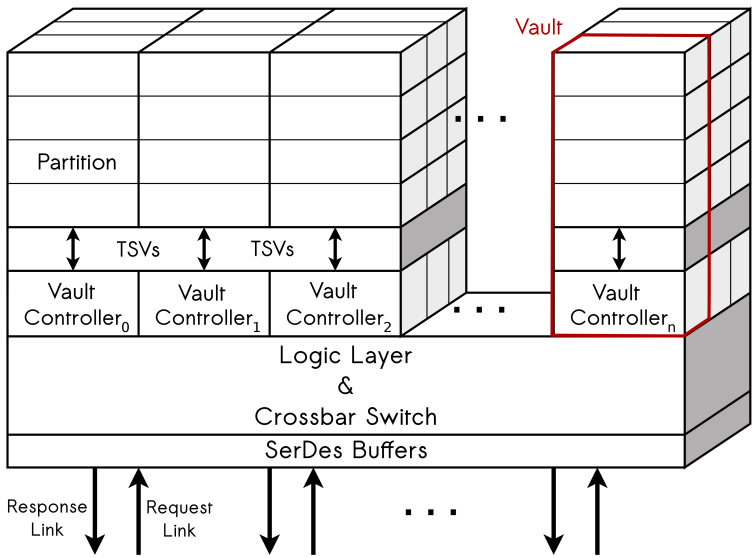
- move **edge storage** off-chip
- edge storage is larger, accesses are read-only and have some locality (more the higher the arity)
- vertex storage is smaller, accesses are completely random, R/W, and hazards induce more stalls if pipeline gets longer

Module to be replaced: adjacency list retrieval

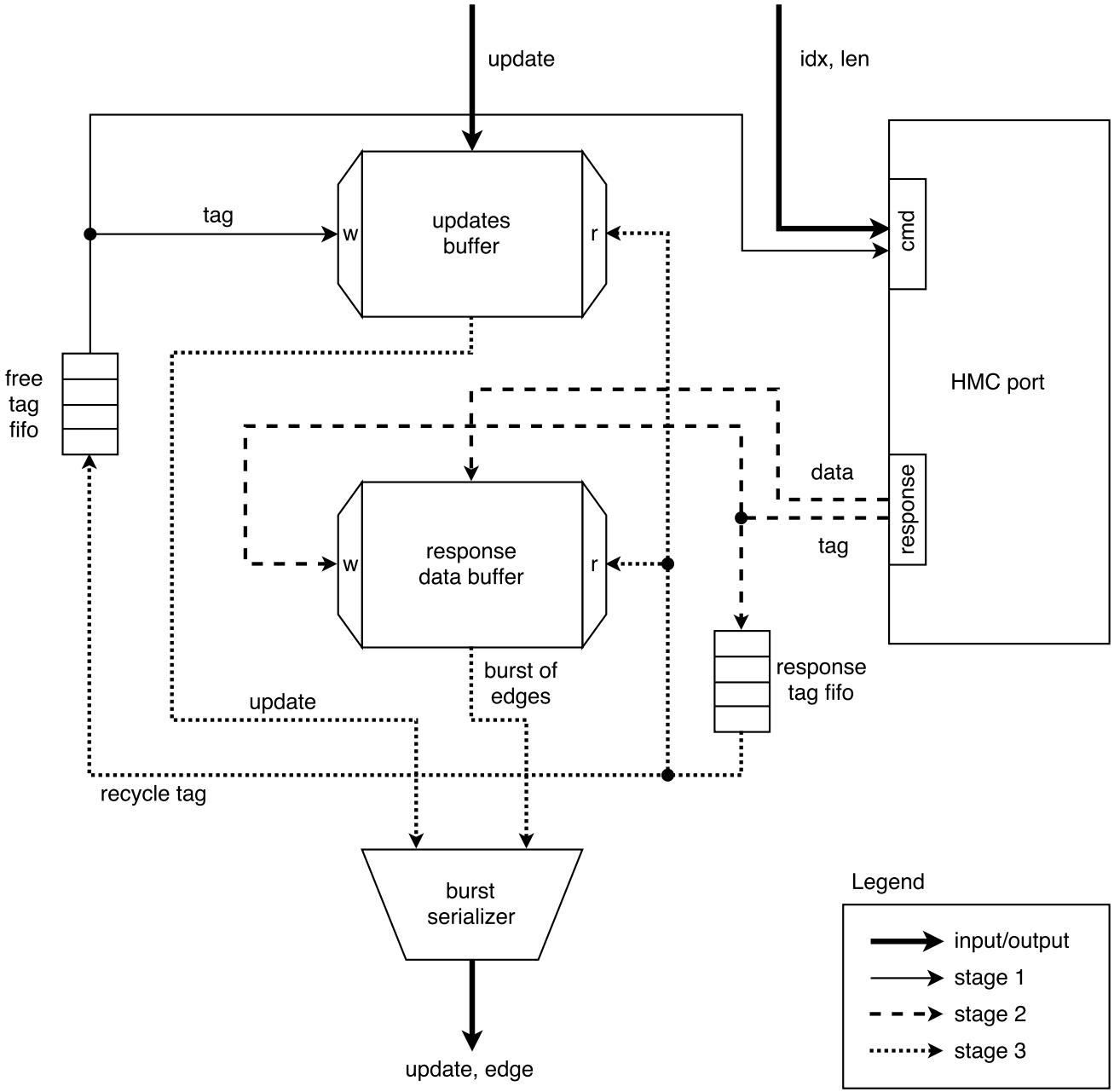


HMC Memory Properties

- Hybrid memory cube: 3D stacked DRAM with memory controller on bottom logic layer
- Packet-based protocol: independent command and response channel, out-of-order response
- User assigns tag to request, same tag identifies response
- No flow control on response - only emit requests you have capacity to handle



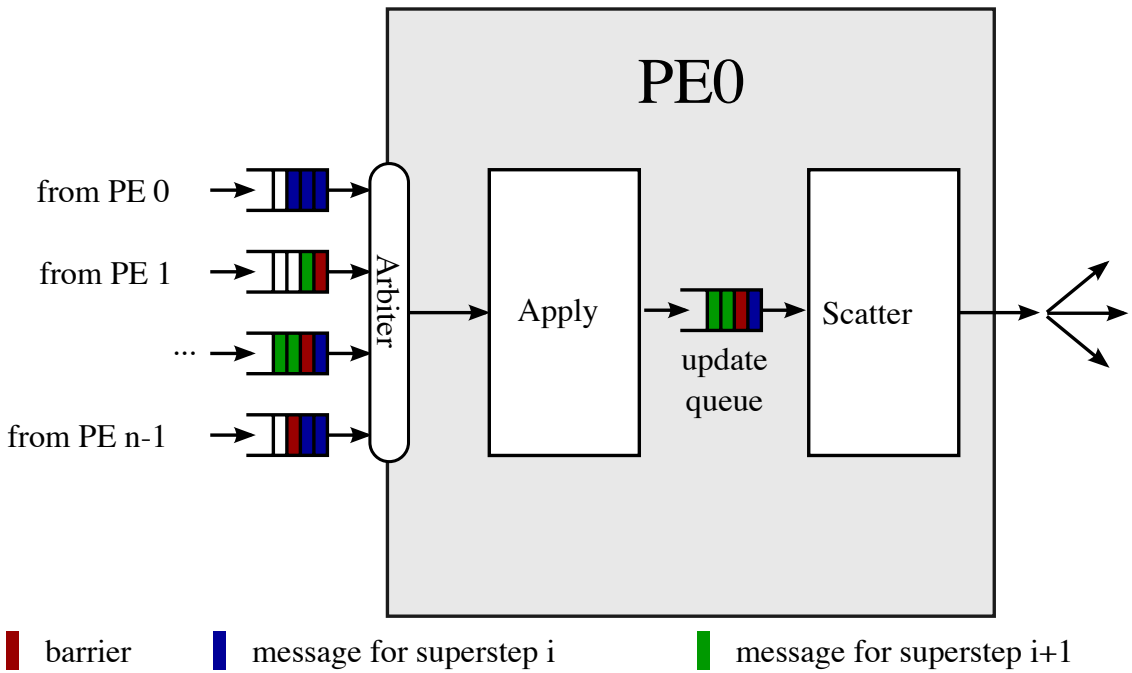
New Module Overview



Increasing the flexibility of the network protocol

Original Network Design

Floating Barrier:



Problems:

- n input FIFOs per PE, i.e. n^2 total
- messages need to be delivered in-order

Adaptations to Synchronization Protocol and Network

Permit reordering of messages:

- Order of messages within superstep not relevant, just need to ensure superstep separation
- New field in message header indicating which superstep message belongs to
- Sender counts messages sent to each PE, includes count in barrier message
- At receiving end, PE counts messages received and compares with count, waits for stragglers if necessary

How many bits does the new field need? 2 (up to 3 message sets at once)

Adaptations to Synchronization Protocol and Network

Required network guarantees:

- messages must be delivered (no packet drop)
- at least 3 channels to separate messages from different supersteps
- separate flow control per channel (message from later superstep may not block message from previous superstep)

Detecting termination in a distributed manner

Termination condition: no message is sent by any vertex during superstep

- Originally: PE sending 2 barriers in a row indicates local inactivity, AND signals from all PEs for global signal
- New: add bitfield to barrier messages indicating sending PE's inactivity. Since PE already counts outgoing messages, it can set bitfield if total is zero
- Receiving side checks bitfield of barriers from all PEs. If all are set, terminate.

Balancing/Partitioning the Graph

Partitioning

Goal: same workload on all PEs

- motivated by switch to graph500 graph generator for better comparisons
- previously just filled PEs with vertices sequentially as they are read in
- 2 issues: completely empty PEs at end, all high-arity vertices in PE 0
- solution: assign vertices in round-robin to PEs instead
- additionally, size memories to fit data to maximize BRAM usage

Results

Evaluation setup

Benchmarks: Breadth First Search and PageRank

Input graphs:

- using graph500 kronecker generator (scale-free graphs)
- two variables: *scale* and *edgefactor*
- Size: graph generated with approx 2^{scale} vertices
- Edgefactor (average arity): fixed at default of 16

Evaluation platform:

- Micron AC-510 FPGA board
- Contains Xilinx Kintex Ultrascale KU060 FPGA and 4GB Micron HMC 1.1 chip
- 9 memory ports $\rightarrow$ 9 PEs
- increase graph size until resource limit reached

Results

- Original: before modifications
- BRAM: with network and partitioning updates, no HMC
- HMC: with all improvements including HMC

	Design	Scale	Runtime	MTEPS
BFS	Original	13	31 ms	229
	BRAM	14	13.5 ms	1105
	HMC	17	131 ms	1001
PR	Original	13	1155 ms	187
	BRAM	14	561 ms	801
	HMC	16	1073 ms	1789

For truly random access, HMC documentation projects performance bound of 1.6 GTEPS

Comparison with related works

Performance data extracted from papers in the literature, normalized per FPGA board:

Design	Input Size	MTEPS
GraphStep(*)	64K Edges	168-450
GraphGen	341K Edges	459
GraphSoC	126K Edges	approx. 100
ForeGraph(*)	490M Edges	464
GraVF	3.7M Edges	1001

(*) These works are simulation-only.

Conclusion

- HMC can provide sufficient bandwidth to keep performance similar to BRAM version
- Larger graph size provides longer adjacency lists for high-arity nodes, gives boost to HMC performance
- Improved synchronization and termination mechanism now fully distributed, network layout independent

Future improvements:

- Connecting multiple FPGAs together
- Moving vertex data off-chip

Q & A

How messages from 3 supersteps come to coexist

1. A and B run superstep $n-1$, which sends messages for superstep n
2. Both A and B finish superstep $n-1$ and send barriers indicating last message for superstep n
3. Messages for A are held up in the network, messages for B get delivered and cause B to run superstep n, which sends messages for $n+1$
4. B receives barriers indicating last message for n from both A and B, and terminates superstep n
5. B starts processing messages for the superstep $n+1$ as it knows (due to the barrier for the superstep n) that there are no more messages for B in superstep n
6. B, during superstep $n+1$, sends messages for superstep $n+2$ to A