A Data-Centric Parallel Architecture for High Performance Graph Processing

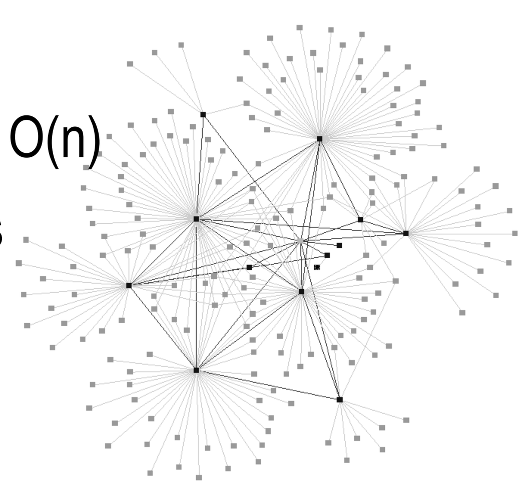
Pervasive Parallelism Laboratory, Stanford University

Tayo Oguntebi, Nicole Rodia, Sungpack Hong, Kunle Olukotun

Graph Algorithms

Problem: Graph Processing

- Memory intensive
  - Large memory footprint (data size)
  - Low spatial and temporal locality
  - Little computation to hide memory latency
- Data-level parallelism
- Expensive, e.g. O(mn) or O(n)
  - n = number of vertices
  - m = number of edges



Architectural Goals

- Lower energy: significantly less data movement
- Minimize communication between processor nodes
- Natural and efficient scaling
- Architectural optimizations for graph algorithms
- Performance: traversed edges per second (TEPS)

Hardware Prototype (planned)

- MaxNode: Maxeler High Density Compute Node
  - Four Xilinx FPGAs in a ring
  - Dedicated memory per FPGA
  - Host CPU with dedicated memory
- Chosen for ample DRAM and interconnect bandwidths

Contact Information

E-mail

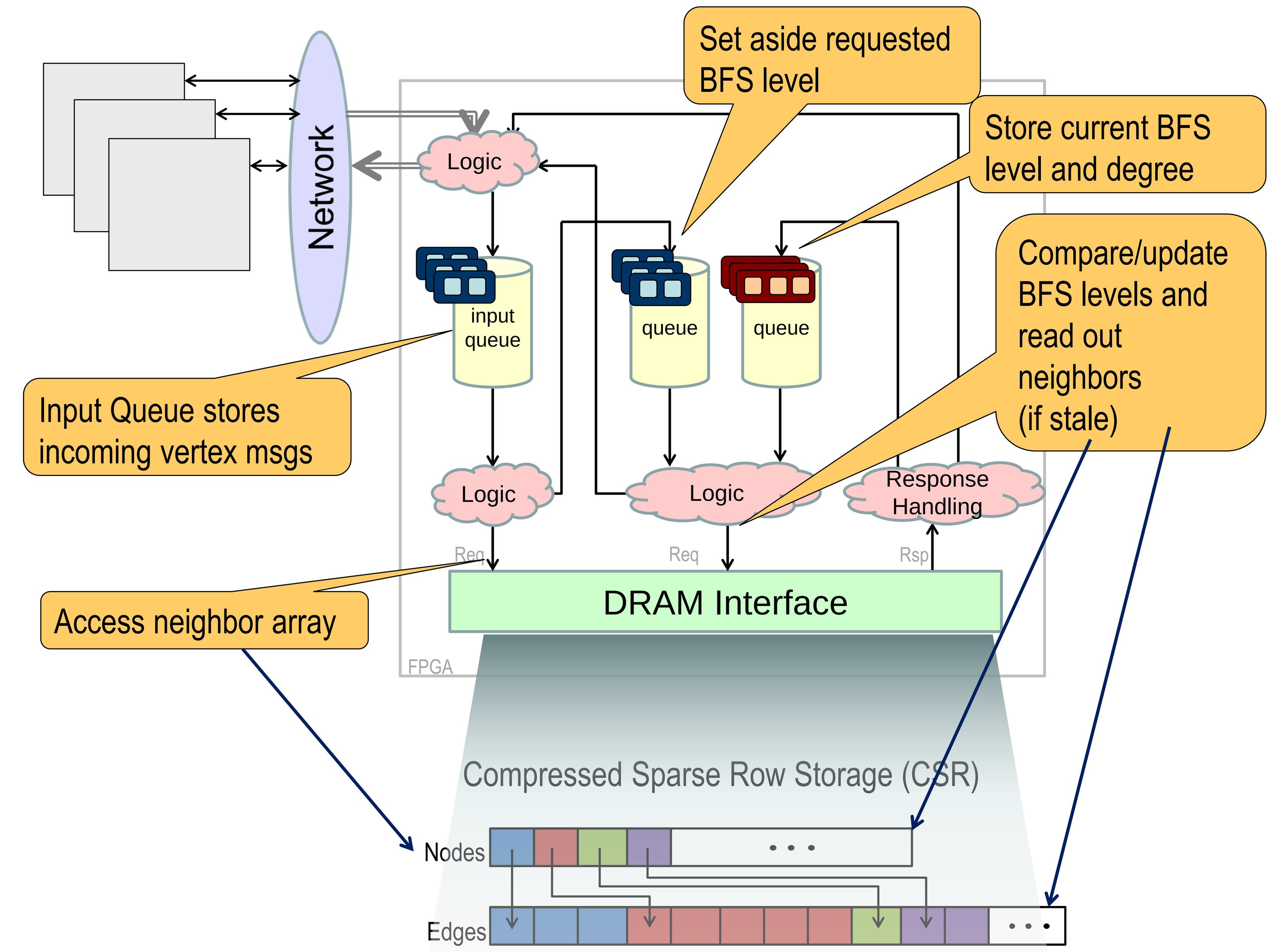
- {tayo, nrodia, hongsup}@stanford.edu

A Simple Asynchronous BFS Design

Approach

- Breadth-first search (BFS) traversal as a representative algorithm
- Assume a partitioned and distributed graph
- Message passing-like distributed memory model
- Vertex updates are encoded in streams of messages that are passed to various nodes
- When well balanced, maximizes efficiency of the memory interface and interconnect bandwidths and will allow for natural scaling, given sufficient interconnect bandwidth

- Minimal message formats: [vertex ID | BFS level] [edge ptr | BFS level | degree]
- Organize the system as a set of queues and messages



Motivation

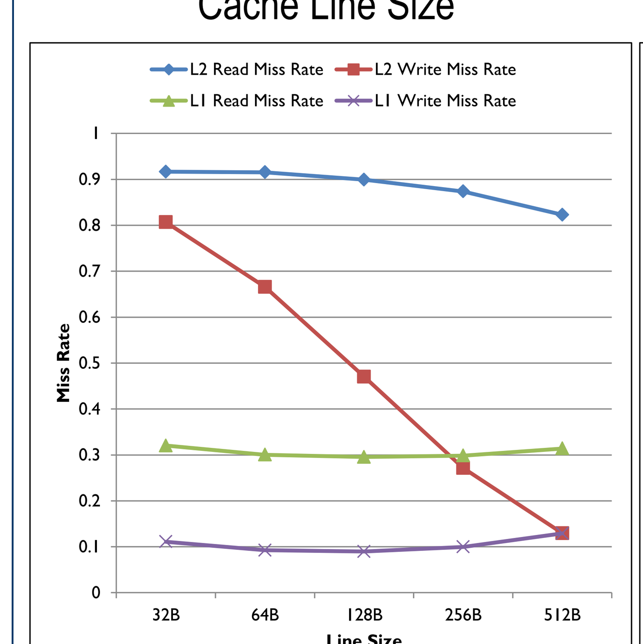
Existing Architectures

- Shared memory multi-core (e.g. x86/SPARC servers)
  - \+ Inexpensive commodity hardware
  - − Multi-level caches – low locality performs poorly
  - − Non-uniform memory access time for multi-socket
  - − Limited number of system threads
- Massively multithreaded (e.g. Cray XMT)
  - \+ Many cores and threads hide memory latency
  - \+ Shared address space with uniform memory access time
  - − Peak compute performance underutilized
  - − Expensive
- GPU
  - \+ High data parallel throughput
  - \+ Large memory bandwidth
  - − Limited memory capacity for large data sets
  - − Scattered memory accesses are serialized and inefficient
- Graph analysis poorly suited to current architectures
  - Low computation-communication ratio
  - Lack of spatial and temporal locality

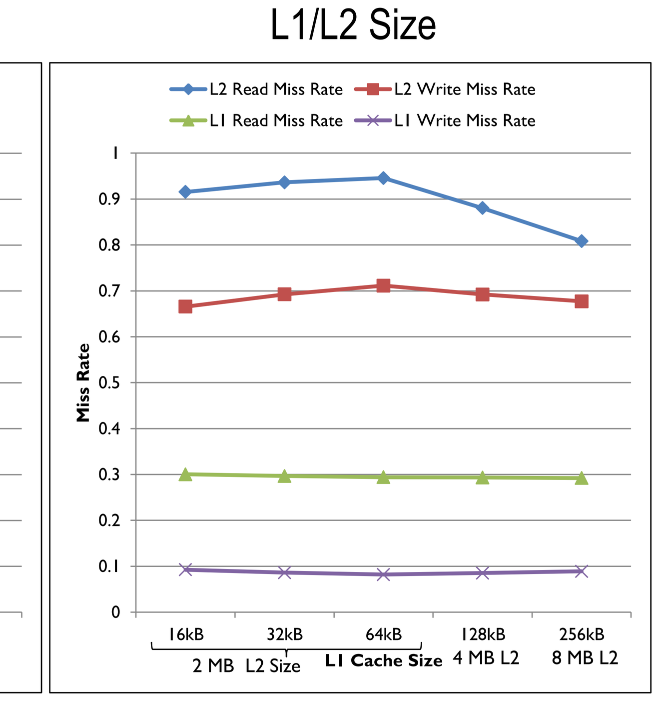
Experimental Results

- Shared memory, cache-coherent, multi-core simulation
- Application: Betweenness Centrality (BFS kernel)
- High read miss rate per reference for large data set
  - > 90% L2; > 30% L1
- ↓ read miss rate with ↑ L1 and L2 cache size
- ↓ L2 read/write miss rate with ↑ cache line size

Miss Rates Per Reference vs. Cache Line Size



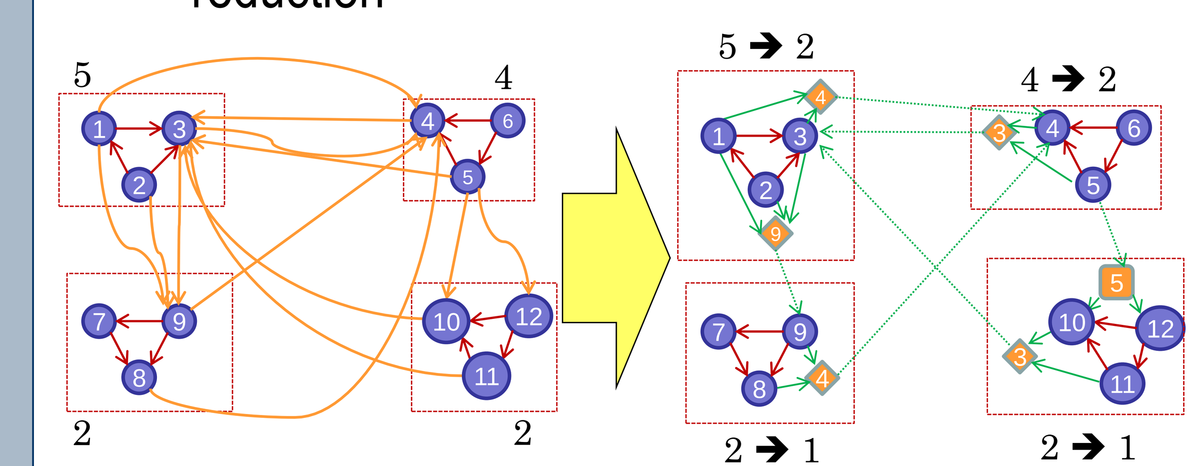
Miss Rates Per Reference vs. L1/L2 Size



Architectural Optimizations

Candidates for Hardware Acceleration

- Barriers
  - Many of the important algorithms are globally level synchronous in nature
  - Hardware support for fast barriers would be beneficial
- Reduction operations
  - Reduce bandwidth and energy requirements by proactively executing commutative reduction operations
  - Especially beneficial for cross-partition reduction



Exploratory Data: Receiving Agents

Graph instance: ‘copter2’ [N = 352238, M = 55476]

- Low-degree graph (avg degree ~ 6)
- Partitioned using METIS, default options

Partitions	4	8
Partition Crossings	27550	36790
Crossings Eliminated	9841	17450
Percent Eliminated	36%	47%

- Data collected using only Receiving Agents → potential for further savings

References

[1] S. Hong, S. K. Kim, T. Oguntebi, and K. Olukotun, “Acc. CUDA graph algorithms at max. warp,” in PPOPP, ’11, pp. 267–276.

[2] S. Hong, T. Oguntebi, and K. Olukotun, “Efficient parallel graph exploration on multi-core CPU and GPU,” in PACT, ’11.

[3] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” in ICPP ‘95, pp. 113-122.