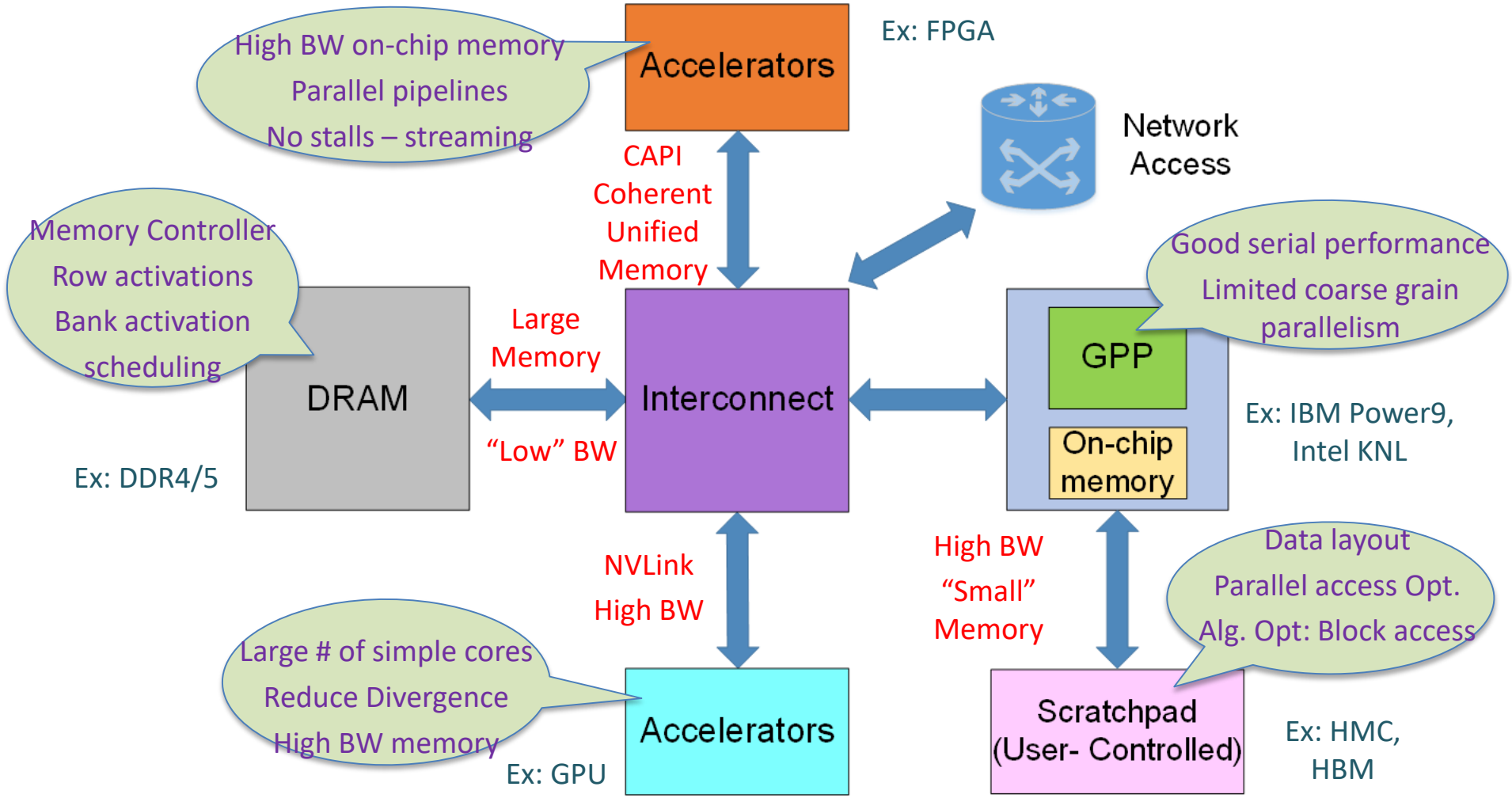
USC University of Southern California

Large Scale Graph Analytics on FPGAs

H2RC 2016
14 November, 2016

Viktor K Prasanna
University of Southern California
fpga.usc.edu

Node Architecture



Scratchpad Memory

- Next Gen Processor Architectures
  - NERSC Cori: Intel KNL
  - TaihuLight: Sunway
- Scratchpad
  - No cache hierarchy: User controlled
  - Cori uses KNL with 3D memory
  - TaihuLight uses Sunway with DDR3
- 3D memory
  - Latency ≈ DDR3
  - BW ≈ 4 - 8× DDR3
  - Block size = 4 - 8 × DDR3
- Ideal performance (?)
  - Scratchpad acts as large buffer in the hierarchy
  - Peak BW for any random block access

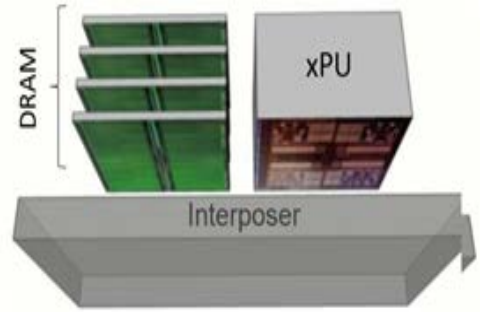


Fig: 3D Memory Architecture

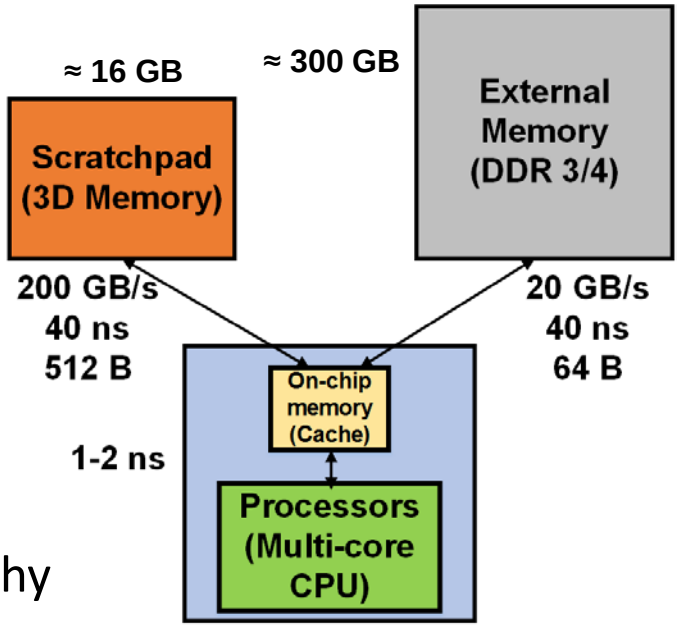


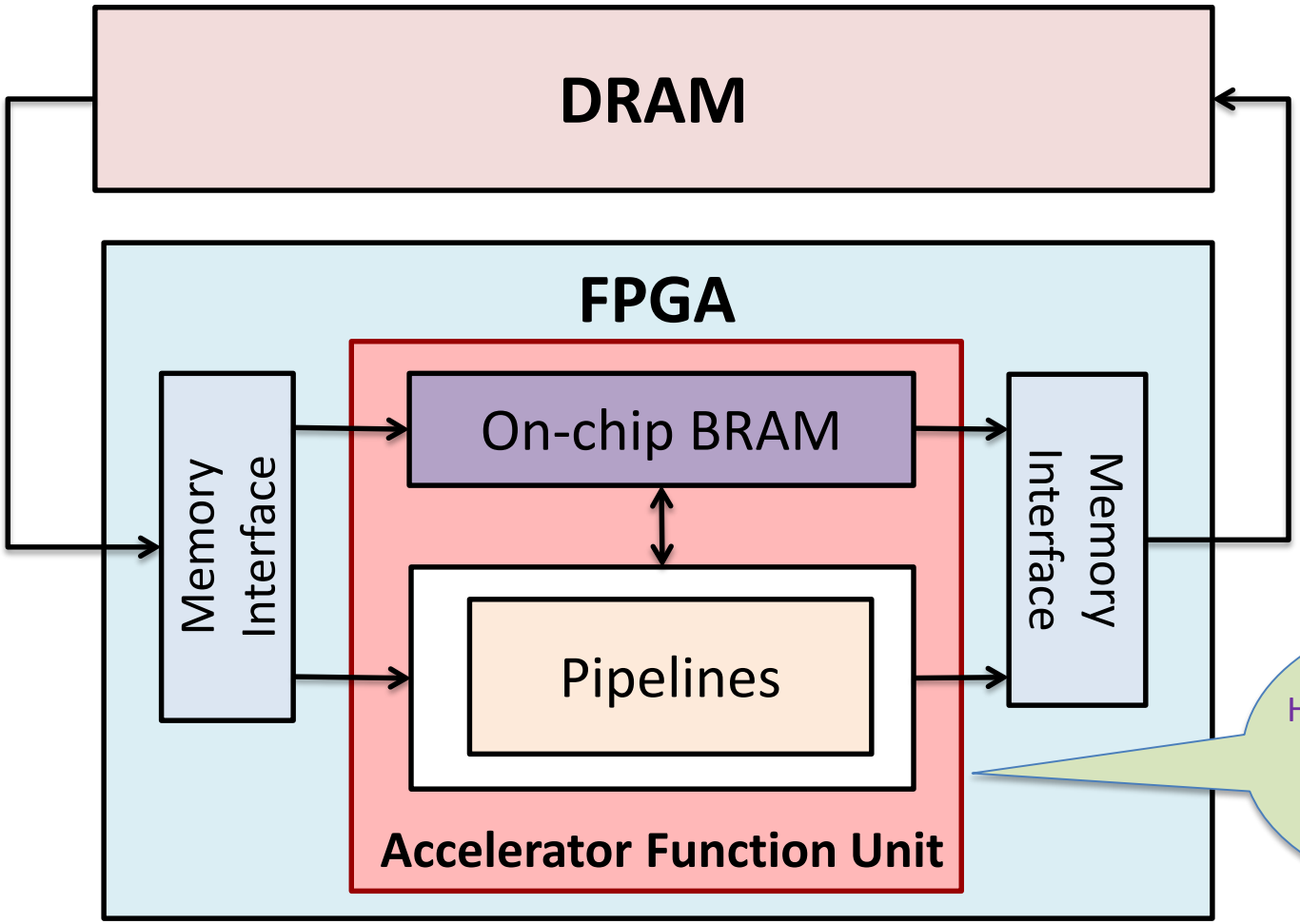
Fig: Scratchpad Architecture

Optimizing Edge-Centric Processing on FPGAs (1)

Edge-centric Paradigm

- Iterative algorithm
- Scatter-gather processing in each iteration
  - Scatter:
    - Edges stream in
    - Updates stream out
  - Gather:
    - Updates stream in
    - Apply updates on vertices
- Advantages
  - No random memory accesses to edges
  - High memory bandwidth utilization
- Many Graph Analytics: Page Rank, BFS, Connected Components, Maximum Independent Set, ……

Optimizing Edge-Centric Processing on FPGAs (2)



Optimizing Edge-Centric Processing on FPGAs (3)

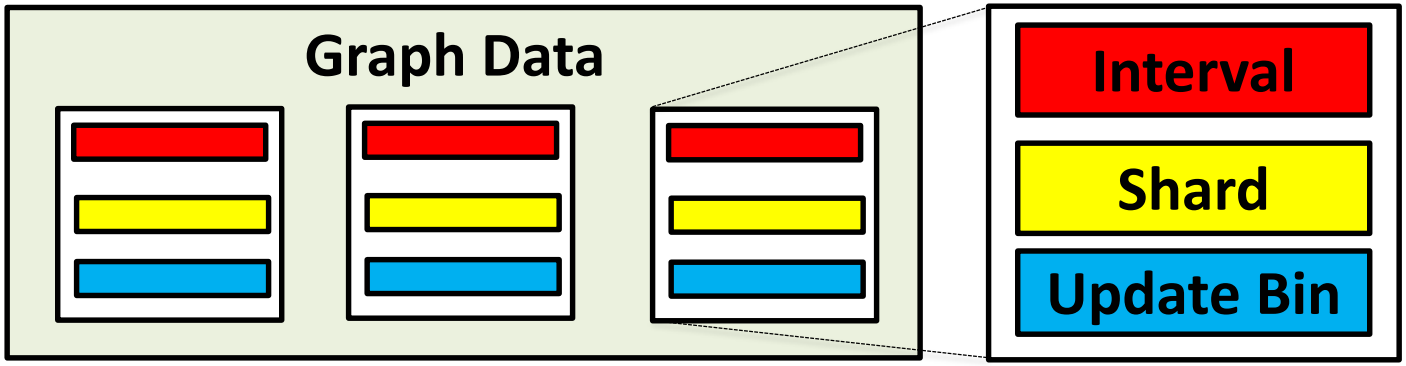
Key Issues

- Data layout to feed the pipelines on FPGA
- # of DRAM row activations (hide latencies)
- Memory buffer management
- Minimize (eliminate?) pipeline stalls
- Graph partitioning– overheads, on chip memory, I/O (communication)

Data Layout Optimization for Edge-Centric Graph Processing (1)

- Partition graph data into k partitions
  - Interval: an array to store vertices (and their labels) with contiguous indices
    - All the intervals of different partitions have the same size
  - Shard: an array to store the edges whose **source** vertices are in the interval
  - Update bin: an array to store the updates whose **destination** vertices are in the interval

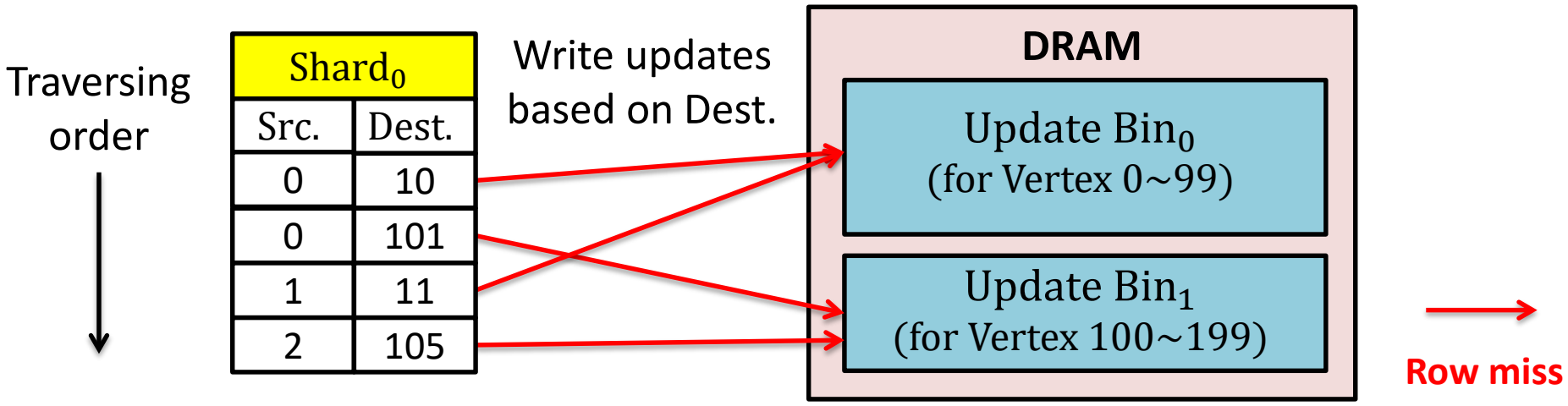
- Goal: Interval of each partition fits in on-chip BRAM



Data Layout Optimization for Edge-Centric Graph Processing (2)

Random Memory Accesses in Scatter Phase

Random memory accesses → DRAM row misses → Non-compulsory row activations →Long access latency → Pipeline stalls

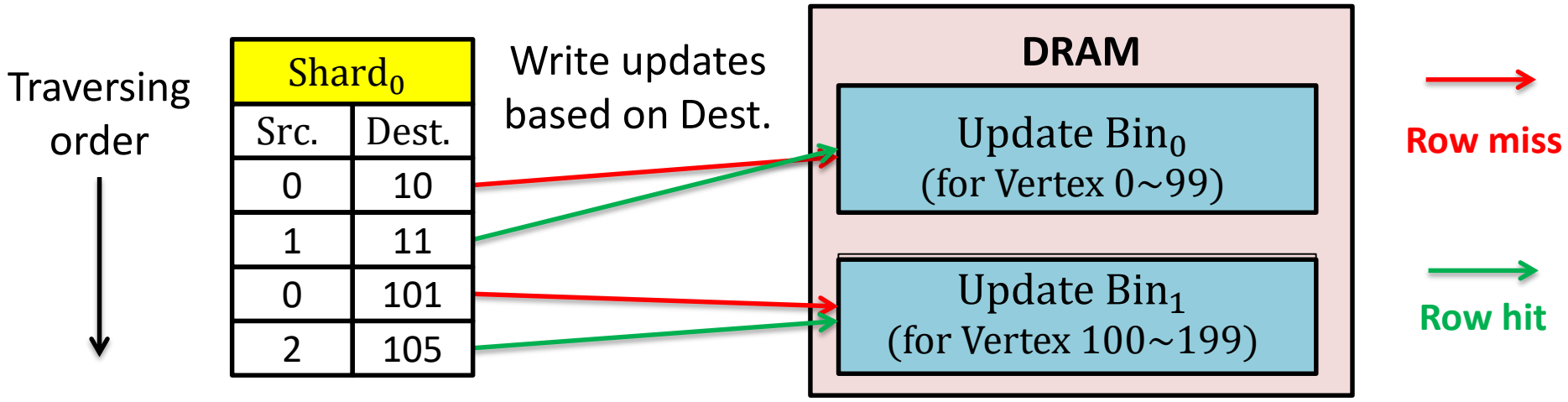


$\Omega(|E|)$ row misses → $\boldsymbol{\Omega(|E|)}$ Non-compulsory row activations

Data Layout Optimization for Edge-Centric Graph Processing (3)

- Data layout optimization: Sort each edge list in each shard based on **destination** vertices

Theorem: In the **Scatter** phase of each iteration, the total number of non-compulsory row misses due to write updates into DRAM for all the partitions is reduced from $\boldsymbol{O(|E|)}$ to $\boldsymbol{O(k^2)}$, where k is the number of partitions.



Data Layout Optimization for Edge-Centric Graph Processing (4)

Number of pipeline stalls due to DRAM accesses

(Ex.: PageRank algorithm, 1 iteration)

Dataset	$\|V\|$	$\|E\|$	No. of pipeline stall cycles		Total No. of clock cycles	
			Optimized	Baseline	Optimized	Baseline
Web-NotreDam	325,729	1,497,134	30,973	682,014	931,272	1,581,077
Web-Google	875,713	5,105,039	120,674	8,238,665	3,120,756	11,346,230
Web-Berkstan	685,230	7,600,595	154,473	2,218,232	4,254,568	6,319,223
Wiki-Talk	2,394,385	5,021,410	172,041	3,242,586	3,872,531	7,369,423

Over **20x** pipeline stall cycles reduction

Row miss in the same bank → 9-cycle stall

Row miss across banks → 5-cycle stall

Data Layout Optimization for Edge-Centric Graph Processing (5)

Number of non-compulsory row activations due to random accesses (PageRank algorithm, 1 iteration)

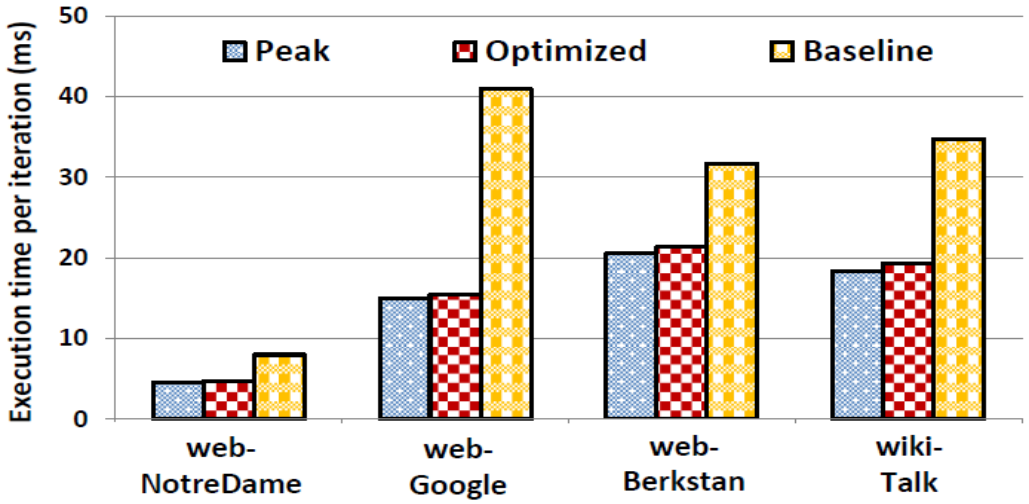
Dataset	No. of non-compulsory row activations	
	Optimized	Baseline
Web-NotreDam	3,197	82,337
Web-Google	6,324	894,220
Web-Berkstan	7,172	265,142
Wiki-Talk	10,642	394,457

Over **25x** row activation reduction

Data Layout Optimization for Edge-Centric Graph Processing (6)

Execution time comparison (FPGA at 200 MHz)

(Ex.: PageRank algorithm, 1 iteration)



- Peak performance: Best achievable performance by assuming pipeline never stalls
- At least **70%** improvement over baseline design

Accelerating Graph Analytics on Heterogeneous Platform (1)

Vertex-centric Paradigm

- Iterative algorithm
- Scatter-gather processing in each iteration
  - Scatter:
    - Active vertices send updates to neighbors through outgoing edges
  - Gather:
    - Apply updates on vertices
- Advantage
  - No redundant edge traversals
- Disadvantages
  - Random memory accesses to edges
  - Low memory bandwidth utilization

Accelerating Graph Analytics on Heterogeneous Platform (2)

Vertex-centric or Edge-centric ?

- Similarities
  - Iterative algorithm
  - Scatter-gather processing in each iteration

- Differences
  - Vertex-centric paradigm only traverses the edges of **active** vertices in an iteration → no redundant edge traversals, but random memory accesses to edges
  - Edge-centric paradigm traverses **all** the edges in an iteration → streaming accesses to edges, but redundant edge traversals

Accelerating Graph Analytics on Heterogeneous Platform (3)

CPU or FPGA?

Dynamically select between vertex-centric and edge-centric paradigms based on active vertex ratio

- **High** active vertex ratio → a small amount of redundant edge traversals is OK → edge-centric paradigm **FPGA? (pipelined)**
- **Low** active vertex ratio → a small amount of random memory accesses is OK → vertex-centric paradigm **CPU? (memory access)**

Accelerating Graph Analytics on Heterogeneous Platform (4)

Hybrid Data Structure

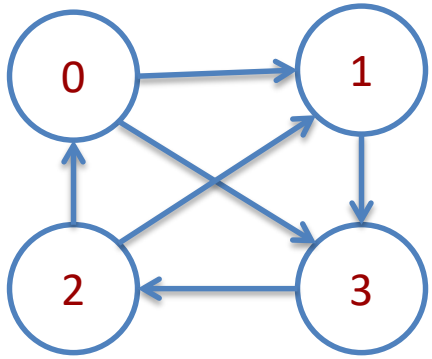
- Combination of CRS (compressed row storage) and COO (coordinate format) to support both vertex-centric and edge-centric paradigms

Vertex array

V_{id}	Pointer
V_0	E_0
V_1	E_2
V_2	E_3
V_3	E_5

Edge array

E_{id}	Src.	Dest.
E_0	V_0	V_1
E_1	V_0	V_3
E_2	V_1	V_3
E_3	V_2	V_0
E_4	V_2	V_1
E_5	V_3	V_2



Accelerating Graph Analytics on Heterogeneous Platform (5)

CPU or FPGA ?

- Partition graph to enable efficient CPU-FPGA co-processing
- Process partitions concurrently
  - High active vertex ratio → FPGA
  - Low active vertex ratio → CPU

- Proposition
  - active vertex ratio >= BW_{VC}/BW_{EC} → high active vertex ratio
  - active vertex ratio < BW_{VC}/BW_{EC} → low active vertex ratio
  - BW_{VC} (BW_{EC}): sustained memory bandwidth for edge traversals by using vertex-centric (edge-centric) paradigm
    - Statically computed

Accelerating Graph Analytics on Heterogeneous Platform (6)

Hybrid Algorithm

```
While any partition has active vertices
        For each partition do
            If its active vertex ratio is high then store it into edge-centric-queue
            Else  store it into vertex-centric-queue
            End if
        End for
        For each partition in vertex-centric-queue do
            Scatter on CPU
        End for
       For each partition in edge-centric-queue do
            Scatter on FPGA
        End for
        For each partition whose update bin is not empty do
            Gather on CPU
        End for
End while
```

Accelerating Graph Analytics on Heterogeneous Platform (10)

Scatter Phase

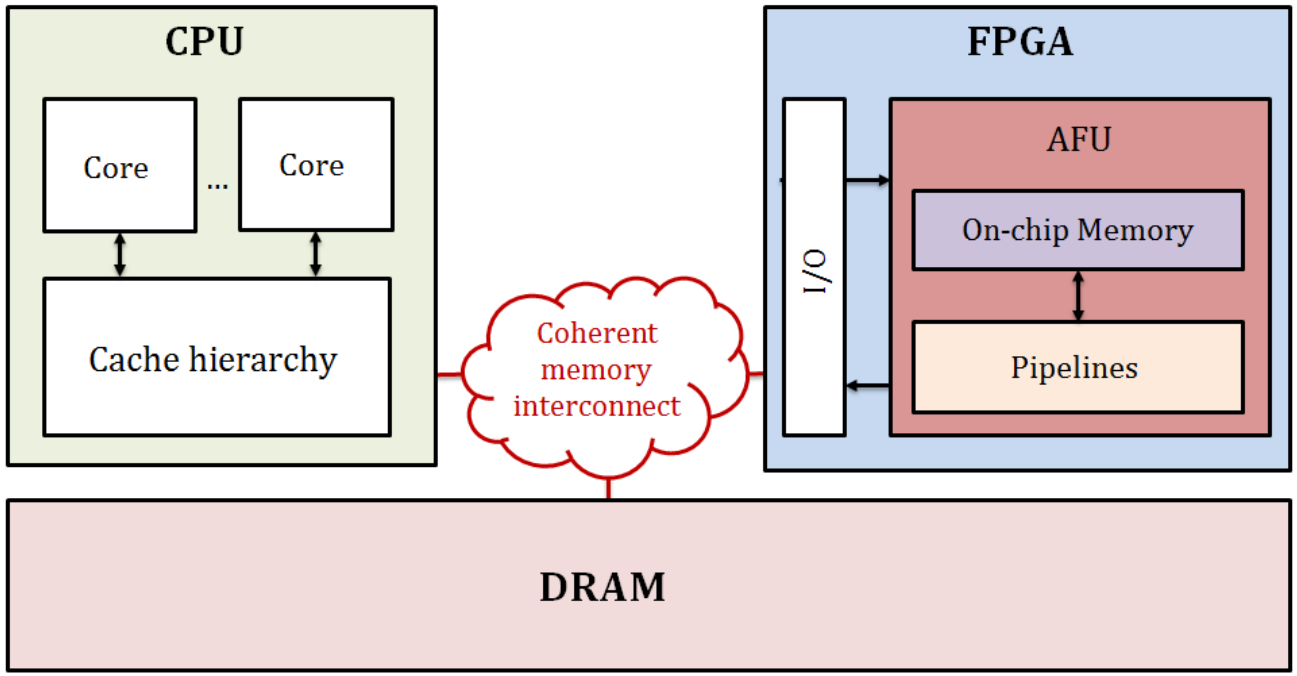
Theorem: The Scatter phase of distinct partitions can be independently executed in parallel with atomic operations using one shared variable per partition.

Gather Phase

Theorem: The Gather phase of distinct partitions can be independently executed in parallel.

Accelerating Graph Analytics on Heterogeneous Platform (11)

- Target platform
  - CPU-FPGA heterogeneous platform with coherent shared-memory (e.g., Intel Xeon processor + Stratix V FPGA)



Accelerating Graph Analytics on Heterogeneous Platform (12)

- Datasets

Name	$\|V\|$	$\|E\|$	Graph Type
Graph 1	3.7 M	16.5 M	Citation network graph
Graph 2	4.8 M	68.9 M	Social network graph
Graph 3	10 M	80 M	Synthetic graph

Accelerating Graph Analytics on Heterogeneous Platform (13)

- Comparison between vertex-centric and edge-centric paradigm on multi-core platform based on the largest dataset (Graph 3)

Algorithm	Vertex-centric			Edge-centric		
	BW_{VC}	BW utilization	Execution time	BW_{EC}	BW utilization	Execution time
BFS	2.08 GB/s	7.3%	0.308 s	20.37 GB/s	71.5%	0.377 s
SSSP	1.38 GB/s	4.8%	2.05 s	10.99 GB/s	38.6%	2.27 s

Accelerating Graph Analytics on Heterogeneous Platform (14)

- Comparison between CPU-only and CPU-FPGA Co-processing

Algorithm	CPU-only			CPU-FPGA Co-processing			
	BW_{EC}	BW utilization	Execution time	BW_{EC}	BW utilization	Execution time	Speedup
BFS	20.37 GB/s	71.5%	0.377 s	25.34 GB/s	88.9%	0.313 s	1.2x
SSSP	10.99 GB/s	38.6%	2.27 s	13.57 GB/s	47.6%	1.84 s	1.2x

Accelerating Graph Analytics on Heterogeneous Platform (15)

- Comparison with baseline designs using the largest dataset

Algorithm	Execution time			
	Vertex-centric only	Edge-centric only	Our approach	Speedup (Best)
BFS	0.373 s	0.378 s	0.233 s	≥ 1.6x (1.9x)
SSSP	2.45 s	2.24 s	1.87 s	≥ 1.2x (1.3x)

Accelerating Graph Analytics on Heterogeneous Platform (16)

- Comparison with state-of-the-art

	Approach	Platform	Millions of edges traversed per second	Ad-hoc ?
BFS	RAW '14	12-core Intel processor + Virtex 5	500	Yes
	FPL '15	Zynq SoC Z7020	172	Yes
	FPGA '16	12-core Intel processor + Virtex 5	12	No
	This work	10-core Intel processor + Stratix V	360	No

	Approach	Platform	Speedup	Ad-hoc ?
SSSP	TOCS '16	Virtex 7	1x	Yes
	This work	10-core Intel processor + Stratix V	8x	No

Thanks

fpga.usc.edu

prasanna@usc.edu