CyGraph: A Reconfigurable Architecture for Parallel Breadth-first Search

Osama G. Attia, Tyler Johnson, Kevin Townsend, Philip Jones, Joseph Zambreno

Reconfigurable Computing Laboratory

Department of Electrical and Computer Engineering

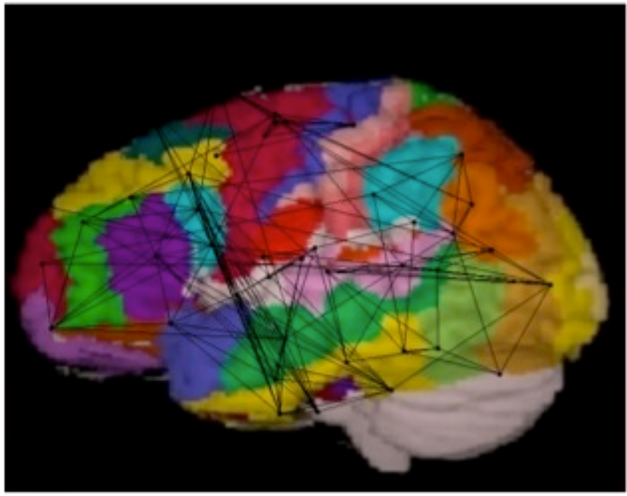
Iowa State University

21st Reconfigurable Architectures Workshop
May 20, 2014, Phoenix, USA

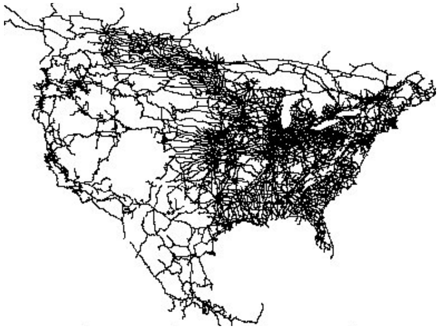
Graphs are Everywhere

- Social Networks
  - Facebook (1.3 Billion user)
  - Twitter (6.5 Million user)
    - 2.1 Billion search queries/day

- Genomics
  - 1 gram of soil = 1 Gb data that is represented as graphs



- Human brain (100 Billion neurons)
- Road Networks



IPDPS 2014 PHOENIX

IEEE computer society

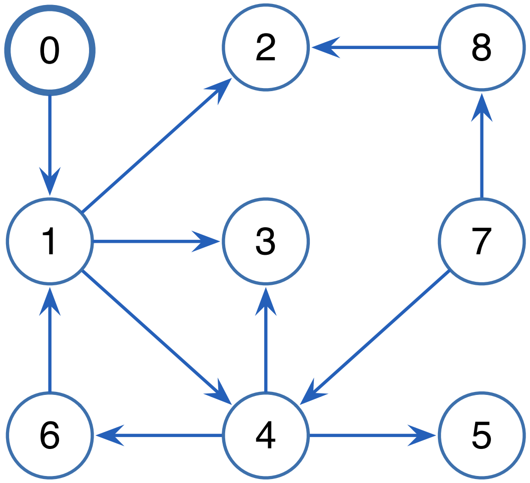
Breadth First Search (BFS)

- Definition:
  - Systematically traverse the connected nodes in a graph starting from a given root node
  - Nodes are visited in the order of hop distance from root.
  - Many applications add extra computation during each BFS iteration and/or post-process the result

- BFS serves as a fundamental building block for many graph processing algorithms

Graph Representation

- Most graph processing algorithms use the well-known compressed sparse row (CSR) format
- The format contains two vectors:
  - Column-indices array (C)
  - Row-offsets array (R)



Column-indices array (C)

1	2, 3, 4	3, 5, 6	1	8	4	2

0 1 2 3 4 5 6 7 8 9 10

Row-offset array (R)

0	1	4	4	4	7	7	8	10
0	1	2	3	4	5	6	7	8

Level-Synchronous BFS

```
$Q_n$.push(root)
current_level $\leftarrow$ 1
while not $Q_n$.empty() do
    $Q_c \leftarrow Q_n$
    while not $Q_c$.empty() do
        $v \leftarrow Q_c$.pop()
        level $\leftarrow$ Levels[$v$]
        if level = 0 then
            Levels[$v$] $\leftarrow$ current_level
            $i \leftarrow R[v]$
            $j \leftarrow R[v+1]$
            for $i < j$; $i \leftarrow i+1$ do
                $u \leftarrow$ C[$i$]
                level $\leftarrow$ Levels[$u$]
                if level = 0 then
                    $Q_n$.push($u$)
    current_level $\leftarrow$ current_level + 1
```

- Execution time dominated by memory latency
- Large memory foot print
- Poor locality
- Random access

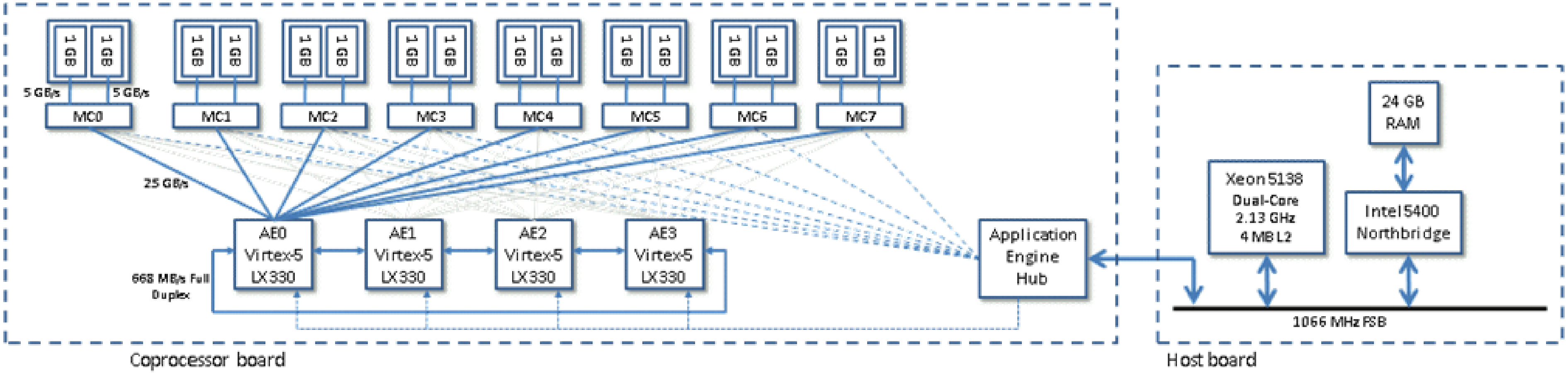
- Memory read request in blue
- Memory Write requests in red

Methodology

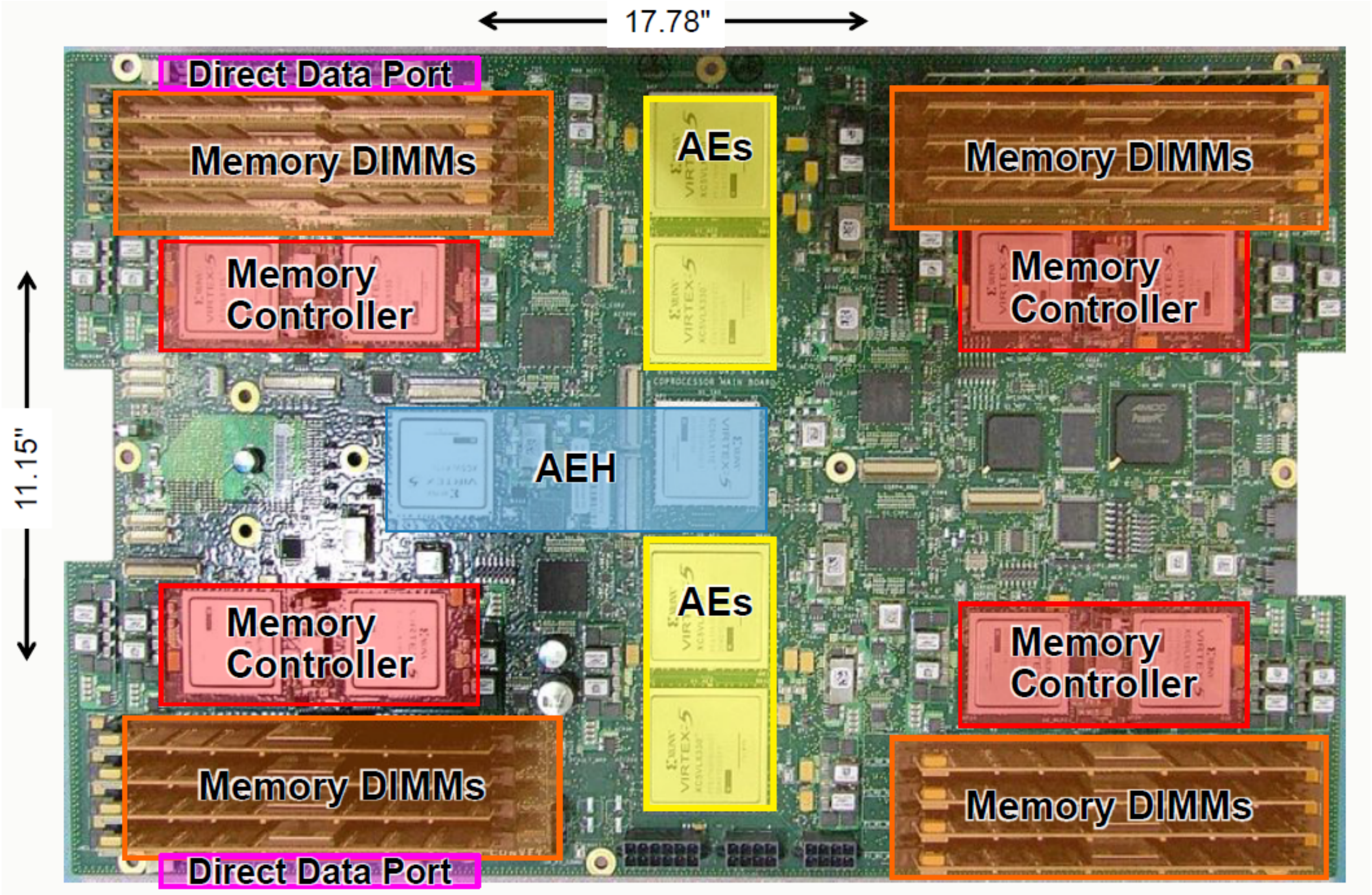
- Custom CSR
- Optimized BFS algorithm
- Parallel hardware implementation
- Multiplexed memory requests
- Kernel-to-kernel communication

Convey HC-2 Computer

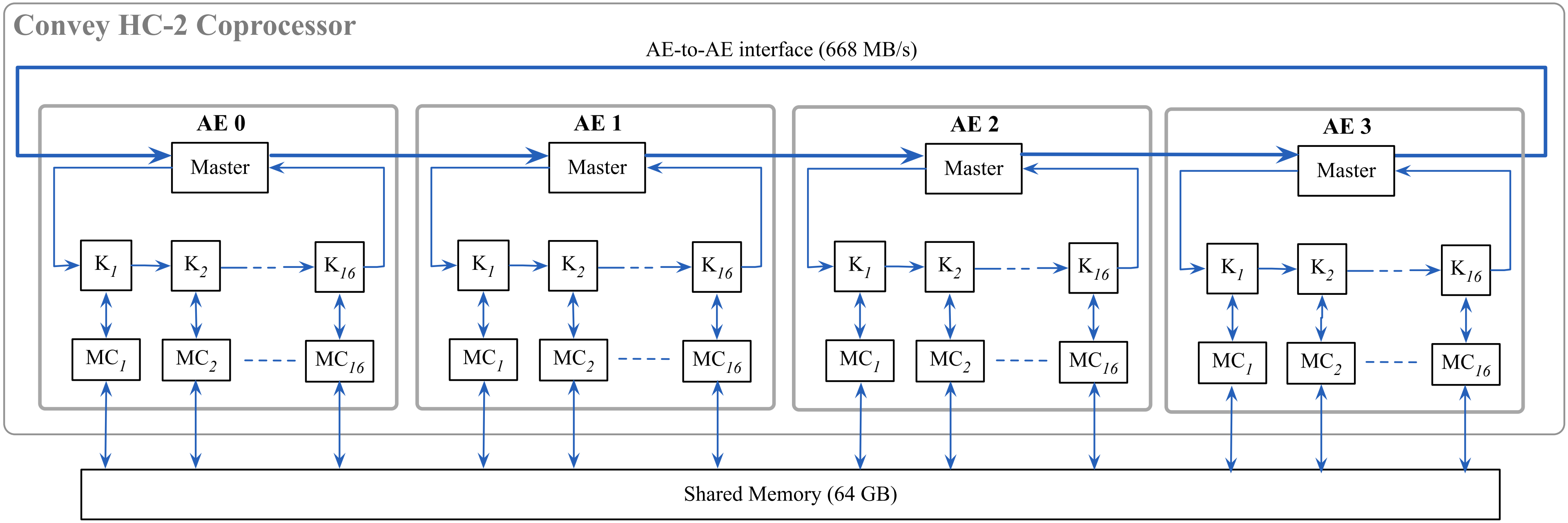
- Four programmable FPGAs, called Application Engines (AEs)
- Eight FPGAs that are used as memory controllers
- AEs has access to memory at peak bandwidth 80GB/s
- Each AE is connected to all memory controllers
- Two FPGAs bridges the host motherboard and the coprocessor board
- Operates at 150 Mhz clock
- AEs are interconnected with full duplex 660 MBps AE-to-AE interface



Convey HC-2 Coprocessor



CyGraph Top Level



- Each AE has access to 16 memory ports through full-crossbar
- 4 AEs = 4 × 16 Kernels = 64 kernels total
- CyGraph uses AE-to-AE interface to scale implementation among different AEs (through Kernel-to-Kernel communication)

Kernel-to-kernel Communication

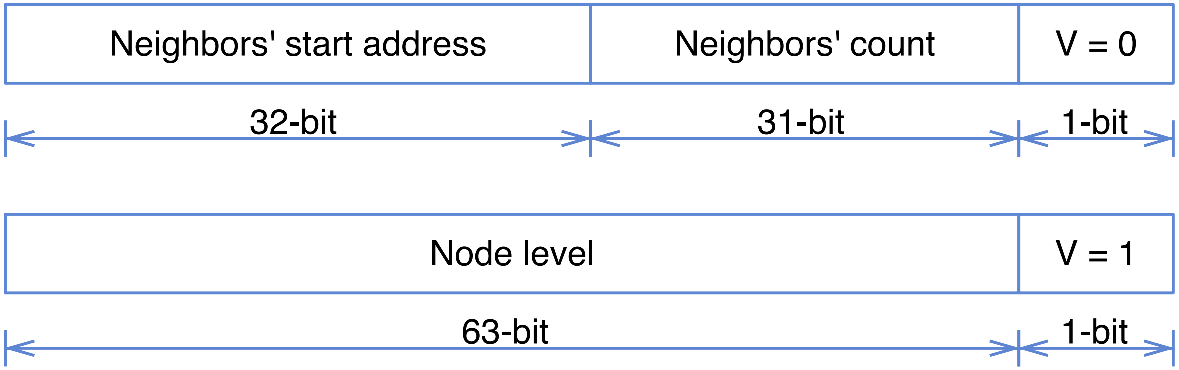
- Behaves as a token ring network
- Allows multiple kernels to write to the next queue
- Makes CyGraph scalable through multiple FPGAs
  - Master Kernel initiates the interface by sending an empty token to the first kernel in the ring
  - Token starts circulating, each kernel gets the token for one cycle
  - Kernel with token, reserves how many items it wants to write to memory and passes the token to next kernel
  - Token = Received token + space to be reserved
  - When a level is finished, token will equal how many nodes were written to memory

Time	T_i	T_{i+1}	T_{i+2}	T_{i+3}	T_{i+4}	T_{i+5}	T_{i+6}
K_0	Send 0	*	*	Send 5			
K_1	*	Send 1			Send 5		
K_2	*	*	Send 3	*		Send 6	

Custom CSR

- With massive large-scale graph, it is hard to implement visited bitmaps on FPGA’s BlockRAM
- We use off-chip memories instead

- In order to optimize the performance, we modified the R array in CSR format as follows:
  - Start address of node (as original implementation)
  - Count of neighbors if visited flag is ’0’ or node’s level if visited flag is ’1’
  - Visited flag



Neighbors' start address | Neighbors' count | V = 0

32-bit | 31-bit | 1-bit

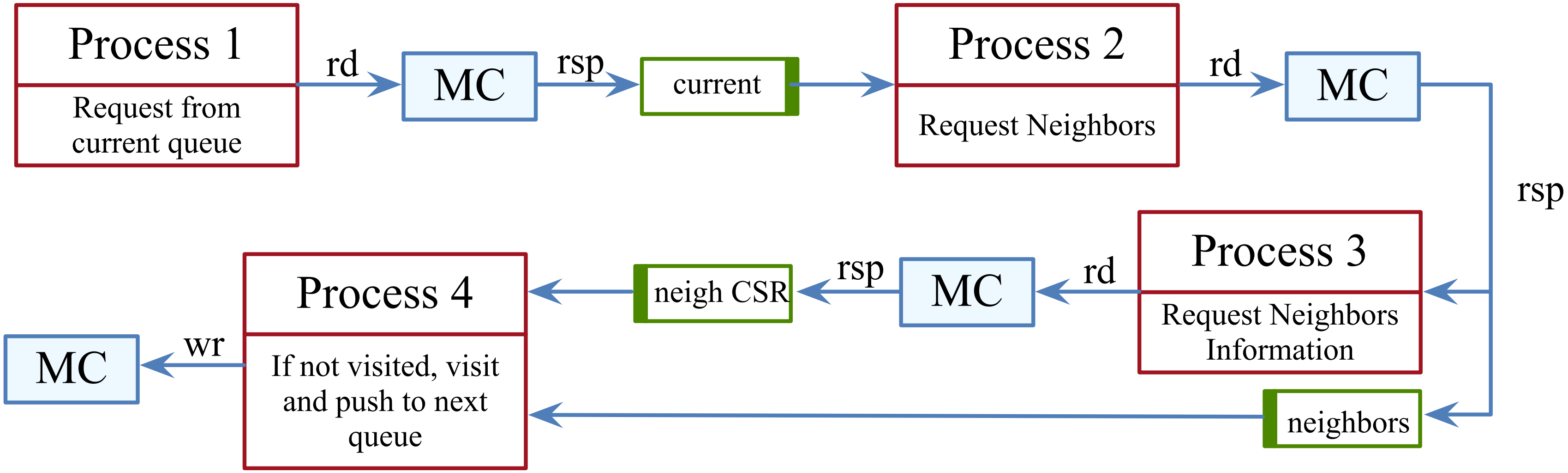
Node level | V = 1

63-bit | 1-bit

CyGraph Kernel Algorithm

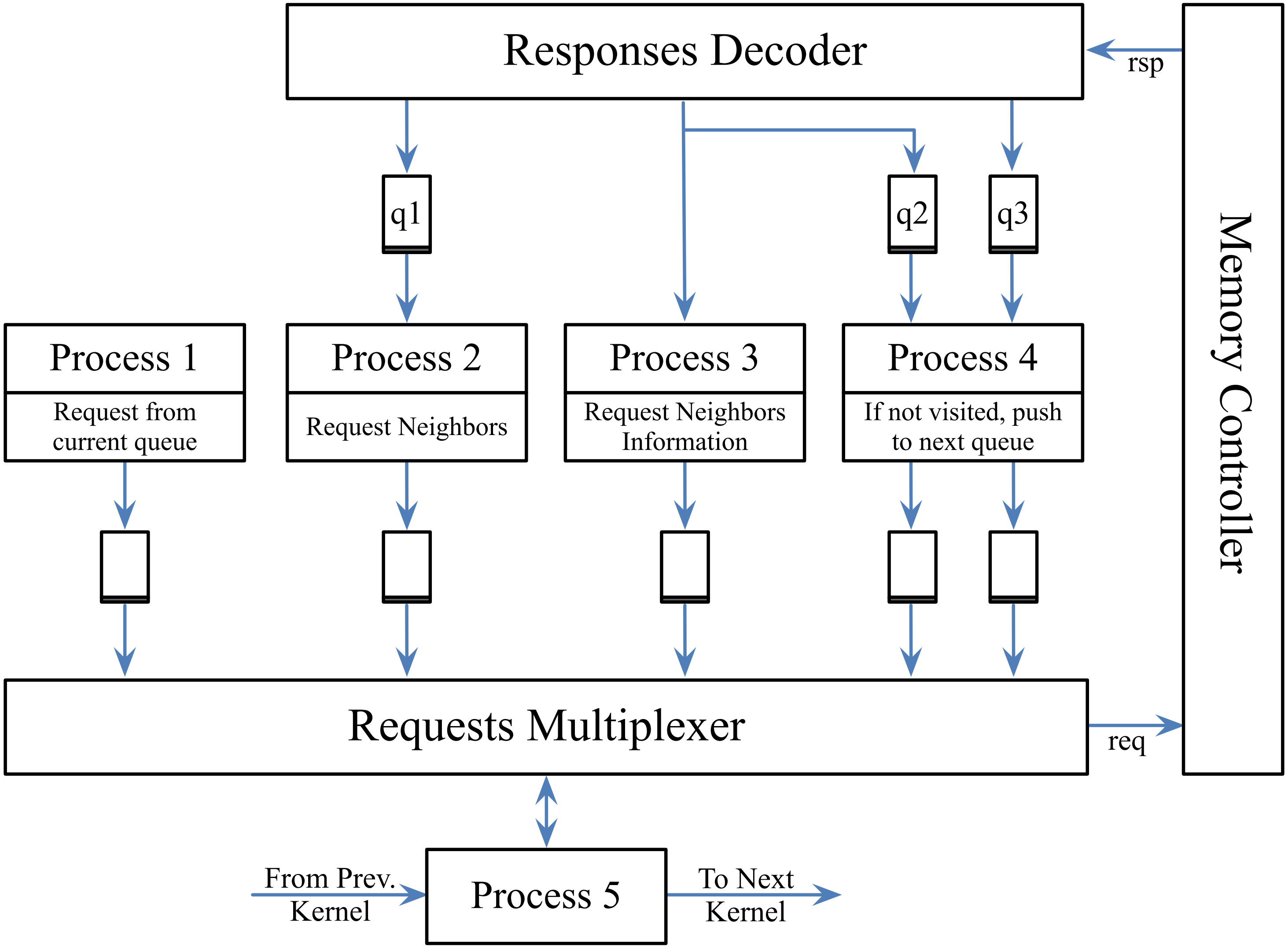
```
$\mathrm{csr} \leftarrow R[\mathrm{root}]$
$Q_n.\mathrm{push}(\mathrm{csr})$
$R[\mathrm{root}] \leftarrow$ b'11
$\mathrm{current\_level} = 1$
while $not\ Q_n.empty()$ do
    $Q_c \leftarrow Q_n$
    while $not\ Q_c.empty()$ do
        $v\_csr \leftarrow Q_c.\mathrm{pop}()$
        $i \leftarrow v\_\mathrm{csr}[63..32]$
        $j \leftarrow v\_\mathrm{csr}[31..1]$
        for $i < j;\ i \leftarrow i + 1$ do
            $u \leftarrow \mathrm{C}[i]$
            $u\_\mathrm{csr} \leftarrow \mathrm{R}[u]$
            if $u\_csr[0] = 0$ then
                $Q_n.\mathrm{push}(u\_\mathrm{csr})$
                $\mathrm{R}[i] \leftarrow (\mathrm{current\_level} + 1)$ & b'1
    $\mathrm{current\_level} \leftarrow \mathrm{current\_level} + 1$
```

Pipelined Design

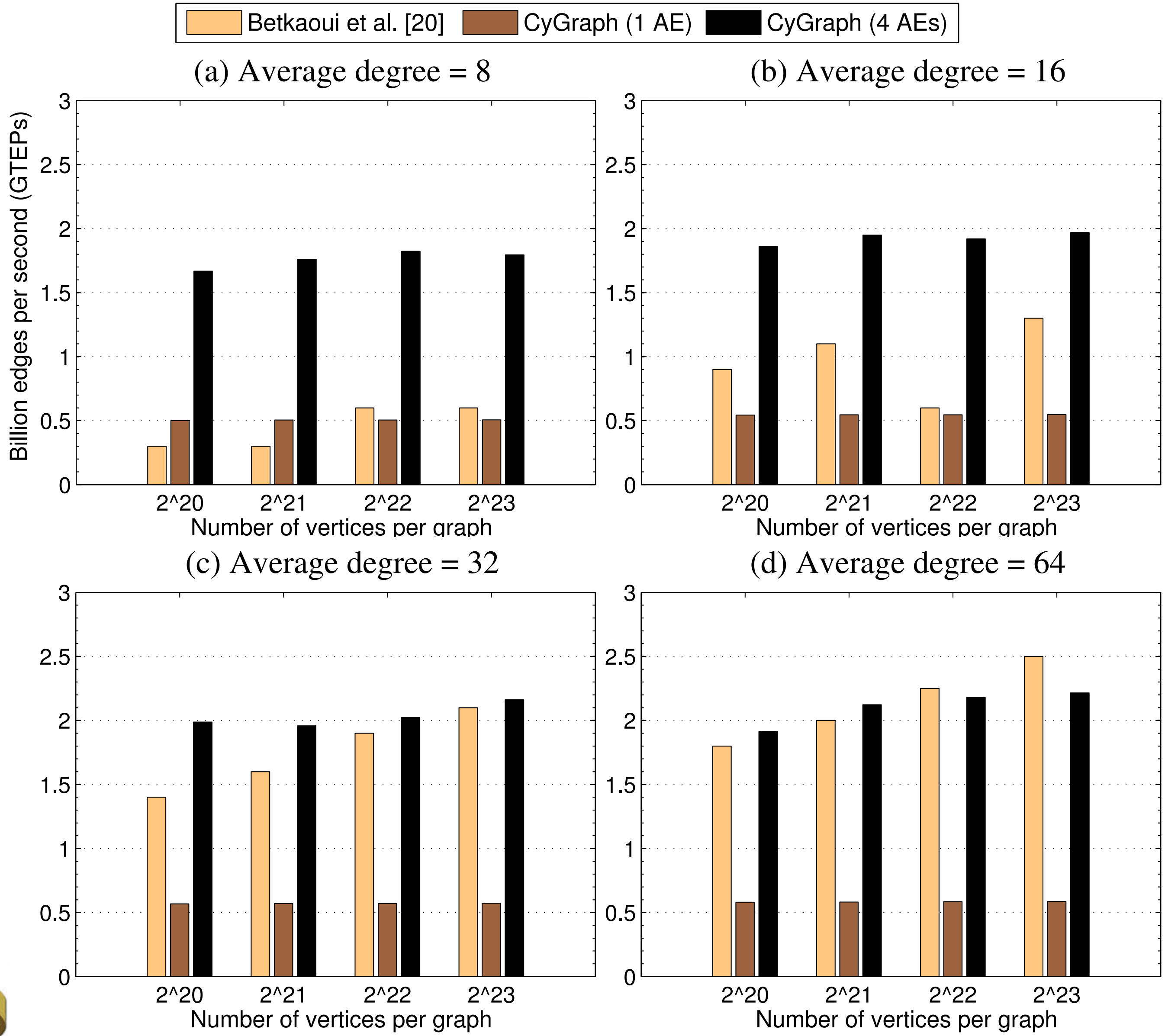


- The kernel could be implemented as a pipeline of four stages/processes
- CyGraph Kernel will serve as processing element (PE) in the Parallel CyGraph implementation

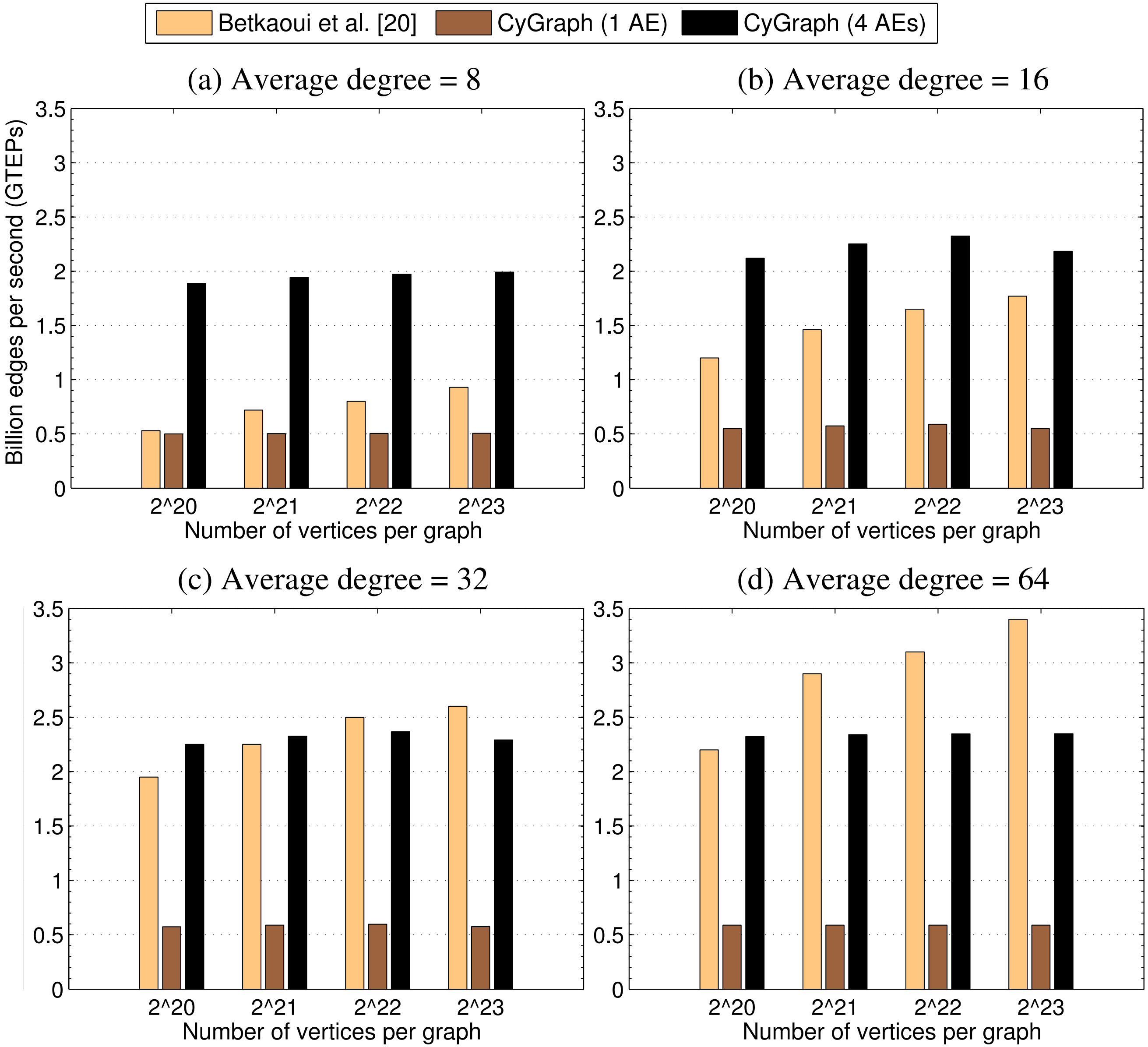
CyGraph Kernel



Experimental Results (R-MAT Graphs)

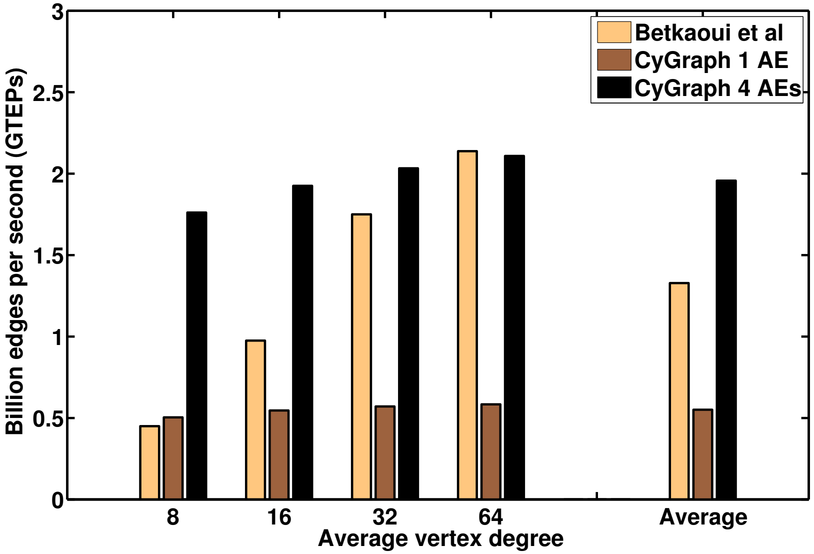


Experimental Results (Random Graphs)

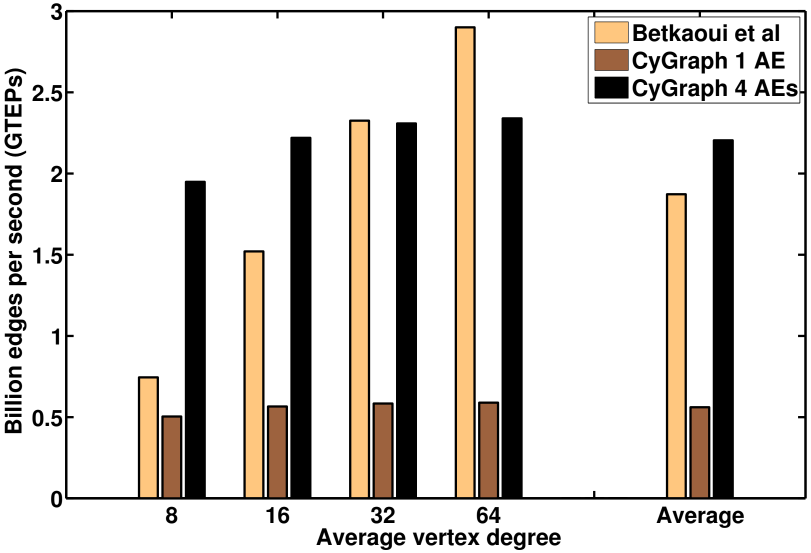


Analysis

R-MAT Graphs



Random Graphs



Resource Utilization

	Slice LUTs	BRAM	Slice LUT-FF
CyGraph 1 AE	53%	55%	74%
CyGraph 4 AEs	55%	55%	74%
Betkaui et al. [20]	80%	64%	n/a

- CyGraph uses less resources
- Remaining resources could be used for adding extra computations through traversal

- Conclusion
  - A reconfigurable hardware for accelerating BFS algorithm
    - Scalable solution for multiple-FPGAs
    - Outperformed pervious state-of-the-art implementation

- Future Work
  - Design approach could be leveraged for other graph processing algorithms
  - Customizing and building upon the CyGraph kernel for real-world applications

Questions

www.rcl.ece.iastate.edu

Reconfigurable Computing Laboratory