Boosting the Performance of FPGA-based Graph Processor using Hybrid Memory Cube: A Case for Breadth First Search

Jialiang Zhang, Soroosh Khoram and Jing Li

Outline

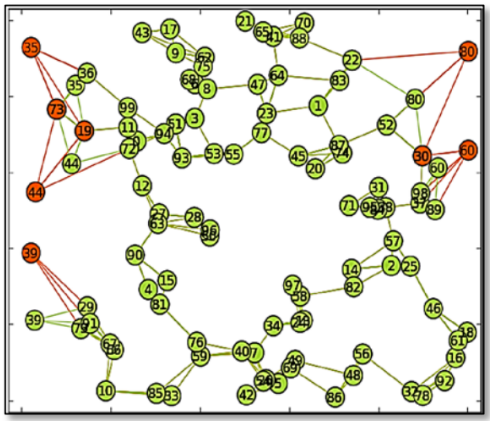
- Background
  - Big graph analytics
  - Hybrid Memory Cube (HMC)
- BFS implementation on HMC-FPGA platform
  - Level synchronized BFS
  - Optimization
- Performance model
- Evaluation
- Conclusion

Outline

- Background
  - Big graph analytics
  - Hybrid Memory Cube (HMC)
- BFS implementation on HMC-FPGA platform
  - Level synchronized BFS
  - Optimization
- Performance model
- Evaluation
- Conclusion

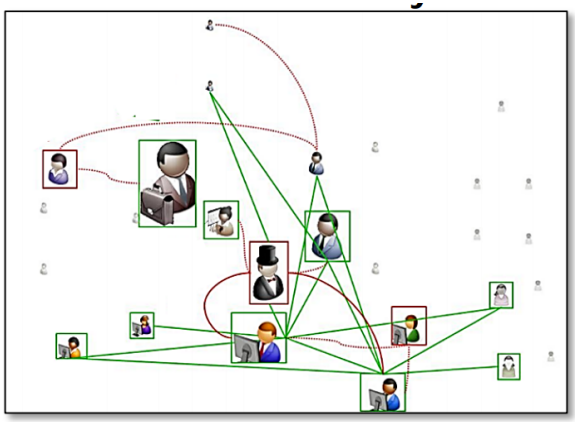
Big Graph Applications

Cyber security



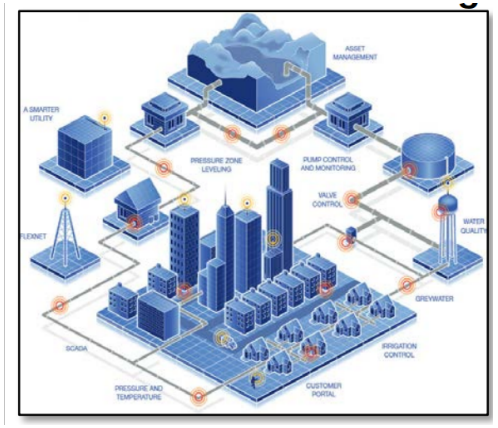
Which cyber events are probes on the network?

Social Media Analysis



Who influences me to buy a product?

Infrastructure Monitoring



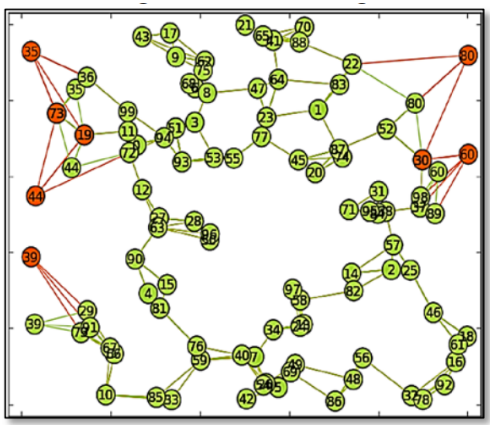
Can I spot failures before they become critical?

- Graph analytics is beginning to be applied to a broad set of problems

From http://www.darpa.mil/attachments/HIVE_Proposers_Day_PM_Briefing.pdf

Big Graph is Sparse

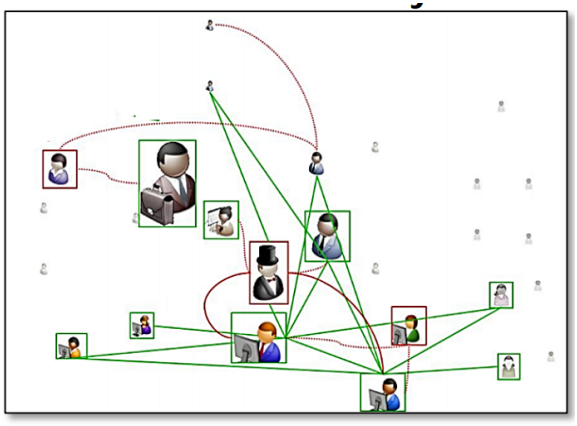
Cyber security



Which cyber events are probes on the network?

Only a small number of events are probes

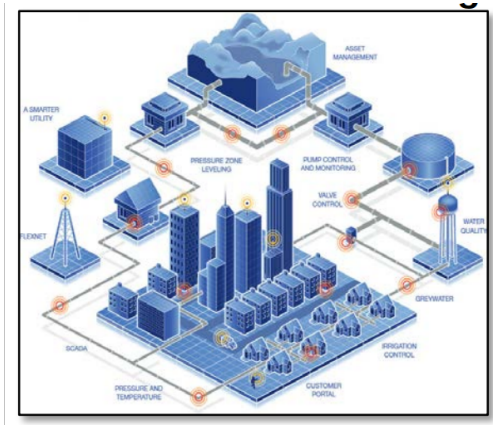
Social Media Analysis



Who influences me to buy a product?

Since only a few people have direct influence on me

Infrastructure Monitoring



Can I spot failures before they become critical?

Only a small number of critical dependencies

- Graph is sparse

From http://www.darpa.mil/attachments/HIVE_Proposers_Day_PM_Briefing.pdf

Challenges in Sparse Graph Traversal

- Big sparse graph is stored in Vertex-Centric Model [1]
- Vertex-Centric model leads to
  - Random memory access pattern
  - Poor locality
  - High synchronization cost
- DDR SDRAM is not a good fit
  - Streaming friendly interface
  - Lack of parallelism

[1] Mccune, et al. Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing

Outline

- Background
  - Big graph analytics
  - Hybrid Memory Cube (HMC)
- BFS implementation on HMC-FPGA platform
  - Level synchronized BFS
  - Optimization
- Performance model
- Evaluation
- Conclusion

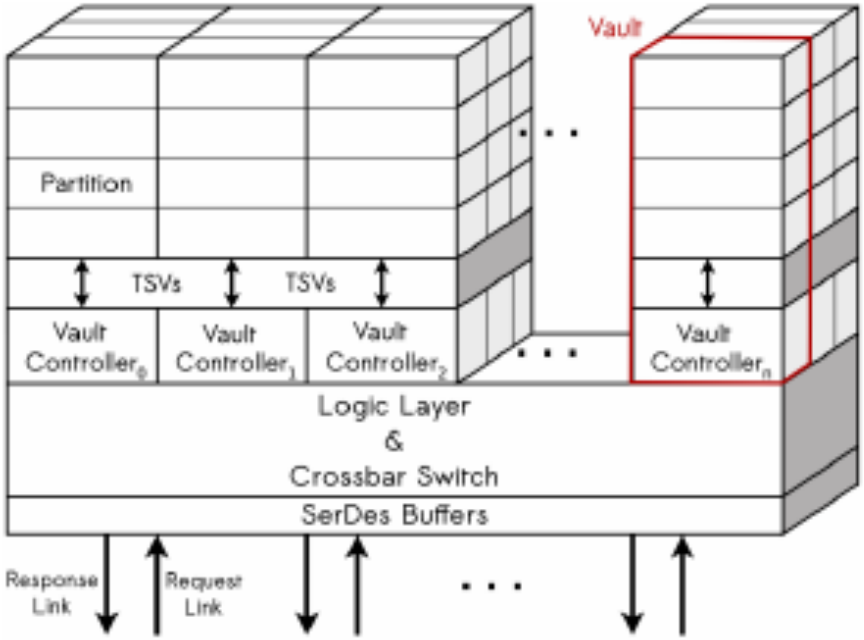
Hybrid Memory Cube (HMC)

- HMC is an emerging memory technology

- More parallelism:
  - 3D Stacking: 8 layers
  - More bank: 512 bank in single chip
  - Returned data is out-of-order

- Interface is friendly to Random Access
  - Packet based serial interface
  - Smaller granularity (16B, 32B, 64B, 128B)

- Near Data Computation
  - Logic on the bottom die



Outline

- Background
  - Big graph analytics
  - Hybrid Memory Cube (HMC)
- BFS implementation on HMC-FPGA platform
  - Level synchronized BFS
  - Optimization
- Performance model
- Evaluation
- Conclusion

Breadth First Search

- Breadth First Search (BFS)
  - A systematic way to traverse the graph
  - A building block for many other algorithms
  - Plenty of data-parallelism in large graph instances
  - Preferred as parallel benchmark (GRAPH500)

- We use BFS as a case study to examine the performance of graph application using HMC

Level Synchronized BFS

- Start from a root, and visit all the connected nodes in a graph
- Nodes closer to the root are visited first
- Nodes of the same hop-distance (level) from the root can be visited in parallel

Algorithm 1 Level-synchronized BFS

```
 1: procedure BFS
 2:     level[v_s] = 1
 3:     parent[v_s] = NULL
 4:     current frontier ← v_s
 5:     current_level = 1
 6:     while current frontier not empty do
 7:         for v ∈ current frontier do
 8:             current frontier = current frontier − v
 9:             E_v = {n ∈ V|(v, n) ∈ E}
10:             for n ∈ E_v do
11:                 if level[n] is 0 then
12:                     level[n] = current_level + 1
13:                     parent[n] = v
14:                     next frontier ← n
15:         current_level = current_level + 1
16:         Swap current frontier with next frontier
```

- 3 vertex set: Visited, Current, Next

Level Synchronized BFS

- Start from a root, and visit all the connected nodes in a graph
- Nodes closer to the root are visited first
- Nodes of the same hop-distance (level) from the root can be visited in parallel

Algorithm 1 Level-synchronized BFS

```
 1: procedure BFS
 2:     level[v_s] = 1
 3:     parent[v_s] = NULL
 4:     current frontier ← v_s
 5:     current_level = 1
 6:     while current frontier not empty do
 7:         for v ∈ current frontier do
 8:             current frontier = current frontier − v
 9:             E_v = {n ∈ V|(v, n) ∈ E}
10:             for n ∈ E_v do
11:                 if level[n] is 0 then
12:                     level[n] = current_level + 1
13:                     parent[n] = v
14:                     next frontier ← n
15:         current_level = current_level + 1
16:         Swap current frontier with next frontier
```

- Load node of current level in parallel

Level Synchronized BFS

- Start from a root, and visit all the connected nodes in a graph
- Nodes closer to the root are visited first
- Nodes of the same hop-distance (level) from the root can be visited in parallel

Algorithm 1 Level-synchronized BFS

```
1:  procedure BFS
2:      level[v_s] = 1
3:      parent[v_s] = NULL
4:      current frontier ← v_s
5:      current_level = 1
6:      while current frontier not empty do
7:          for v ∈ current frontier do
8:              current frontier = current frontier − v
9:              E_v = {n ∈ V|(v, n) ∈ E}
10:             for n ∈ E_v do
11:                 if level[n] is 0 then
12:                     level[n] = current_level + 1
13:                     parent[n] = v
14:                     next frontier ← n
15:         current_level = current_level + 1
16:         Swap current frontier with next frontier
```

- Load Neighbors of current level nodes in parallel

Level Synchronized BFS

- Start from a root, and visit all the connected nodes in a graph
- Nodes closer to the root are visited first
- Nodes of the same hop-distance (level) from the root can be visited in parallel

Algorithm 1 Level-synchronized BFS

```
1:  procedure BFS
2:      level[v_s] = 1
3:      parent[v_s] = NULL
4:      current frontier ← v_s
5:      current_level = 1
6:      while current frontier not empty do
7:          for v ∈ current frontier do
8:              current frontier = current frontier − v
9:              E_v = {n ∈ V|(v, n) ∈ E}
10:             for n ∈ E_v do
11:                 if level[n] is 0 then
12:                     level[n] = current_level + 1
13:                     parent[n] = v
14:                     next frontier ← n
15:         current_level = current_level + 1
16:         Swap current frontier with next frontier
```

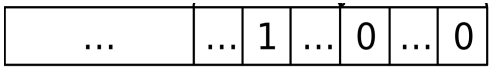
- Synchronize at the end of each level

Outline

- Background
  - Big graph analytics
  - Hybrid Memory Cube (HMC)
- BFS implementation on HMC-FPGA platform
  - Level synchronized BFS
  - Optimization
- Performance model
- Evaluation
- Conclusion

Mark Vertices using Bitmap

- Using bitmap to mark the vertices during visit
  - Reduce the work size of the visited set
  - Requires atomic operation as two parallel kernel may want to update different address in the same memory address
  - Atomic operation is costly



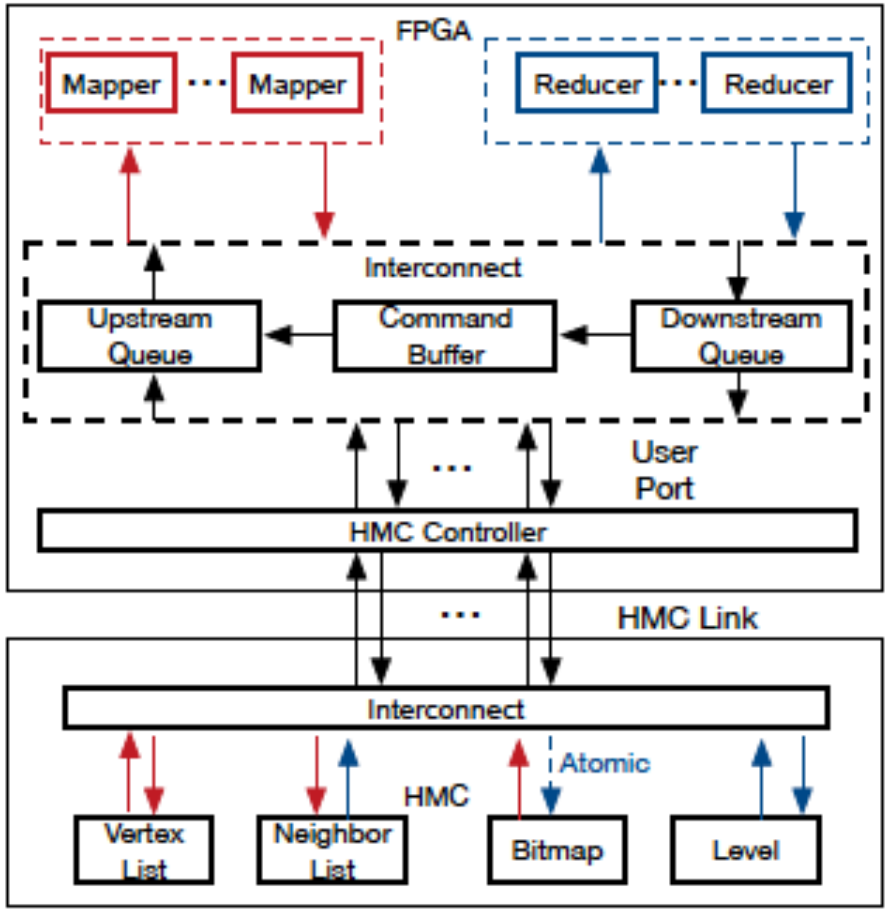
"0" indicates the vertices should be visted in next level

- HMC could help with its built-in atomic bit update command
  - Use a bitmask to change the granularity to 1 bit

Map-Reduce-Like BFS Framework

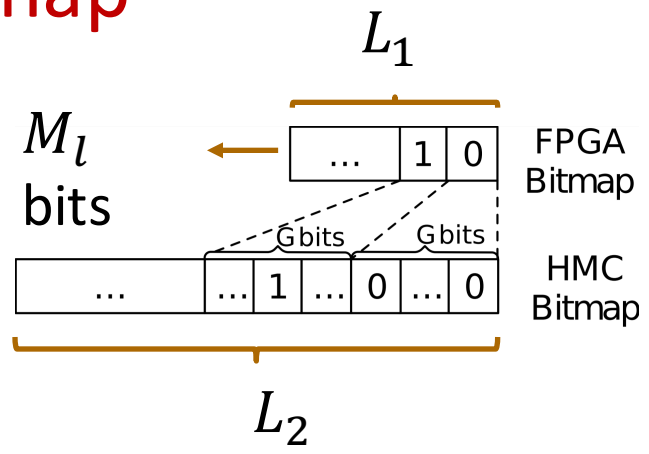
- Map-Reduce-like framework divide BFS kernel into independent stages (no data dependency):
  - Mapper: Get neighbors of current level
  - Reducer: Mark the vertices needs to be visited next

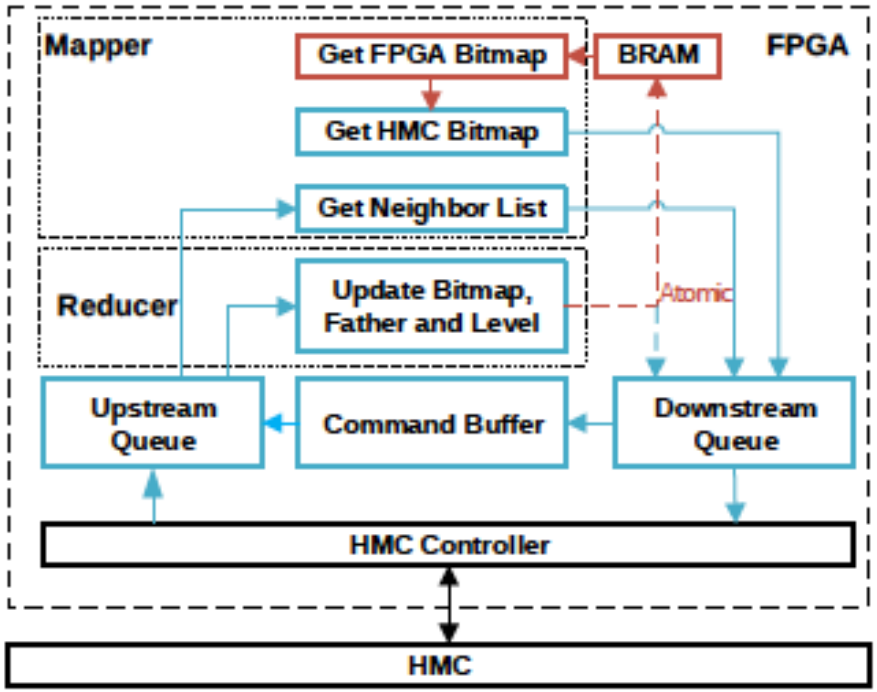
- Mappers and reducers communicates via HMC
  - The returned data is out of order
  - Using a command buffer to separate the traffic from different kernels



Two-level Bitmap

- Scanning bitmap at each level is costly
  - Lots of "0" in the bitmap due to the nature of sparse graph
  - Bitmap is too large for the BRAM
- Propose to use two-level bitmap
  - Store a small bitmap on-chip
  - Remove the unnecessary HMC access due to bitmap scanning





Outline

- Background
  - Big graph analytics
  - Hybrid Memory Cube (HMC)
- BFS implementation on HMC-FPGA platform
  - Level synchronized BFS
  - Optimization
- Performance model
- Evaluation
- Conclusion

Performance Analysis of the BFS Implementation

- We present an analytical model for HMC access latency.
- Apply the model to our BFS implementation and understand the performance of the BFS bitmap scan step.
- Choose the optimal parameter based the analysis

Key Observations from HMC Architecture

- Packets are serialized through the IO. The packet duration is proportional to the size of the packet including the data being transferred, the header, and the tail.

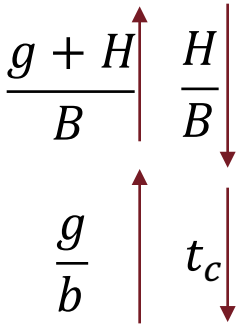
- The HMC latency of a packet comprises
  - A constant delay of processing the packet header
  - Data transfer delay which is proportional to data size.

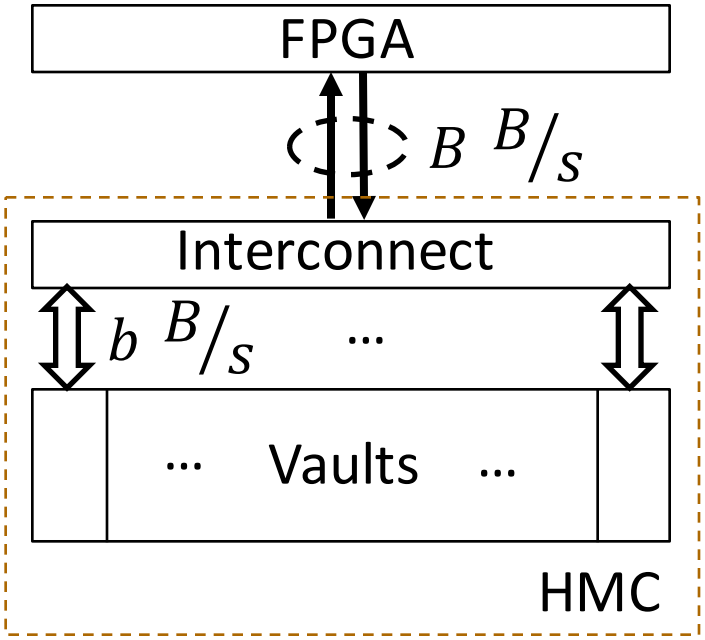
- The internal delay changes if there is a vault conflict . Parallel access to different vaults result in less latency compared to accesses with vault conflicts.

Analytical Performance Model

- Access latency depends on:
  - g: packet size
  - H: Header size
  - B: Link bandwidth
  - b: Internal bandwidth
  - t_c: Header processing latency

HMC parameters





- For a n byte HMC reads :
- $Access\ Latency = n\frac{g}{b} + n\frac{g+2H}{B} + t_c$

g should be large for most reads and small for writes.

Performance of Bitmap Scan of BFS

- Bitmap scan latency is

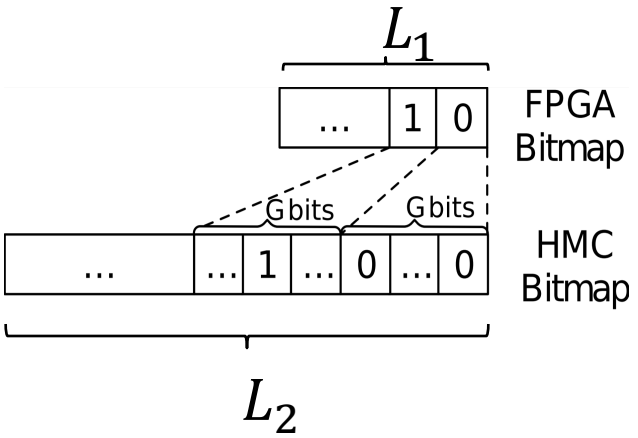
$$T_{scan_l} = M_l(k\frac{g}{b} + k\frac{g+2H}{b} + t_c)$$

  - M_l: Number of "1" in the on-chip bitmap
  - k: Number of HMC requests
  - G: Mapping granularity (L_2/L_1)

- For graph with V vertices and L levels, to get β speedup, we need to satisfy:

$$G < L_2(1 - (1 - \frac{8g'T}{L_2\beta T'})^{\frac{L}{V}})$$

  - Higher speedup (β) requires smaller G.
  - Increasing G leads to less on-chip memory usage without hurting performance, as long as the condition holds.



Insights from the Model and Analysis

- The packet size g should be large for reads and small for writes.

- The mapping granularity condition

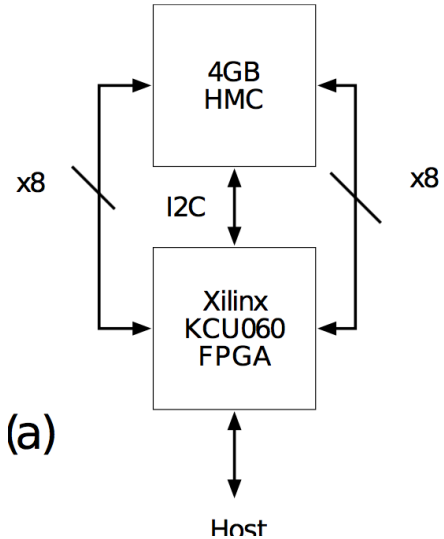
$$G < L_2(1 - (1 - \frac{8g'T}{L_2\beta T'})^{\frac{L}{V}})$$

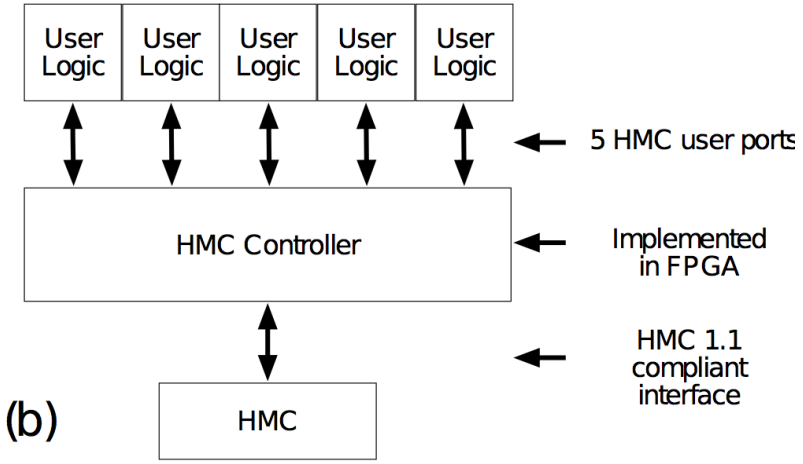
Outline

- Background
  - Big graph analytics
  - Hybrid Memory Cube (HMC)
- BFS implementation on HMC-FPGA platform
  - Level synchronized BFS
  - Optimization
- Performance model
- Evaluation
- Conclusion

Experimental Setup

- Platform: Pico Computing AC510
- FPGA: Xilinx Ultra scale KCU060
- HMC bandwidth: 30GB/s on both direction
- HMC capacity: 4GB

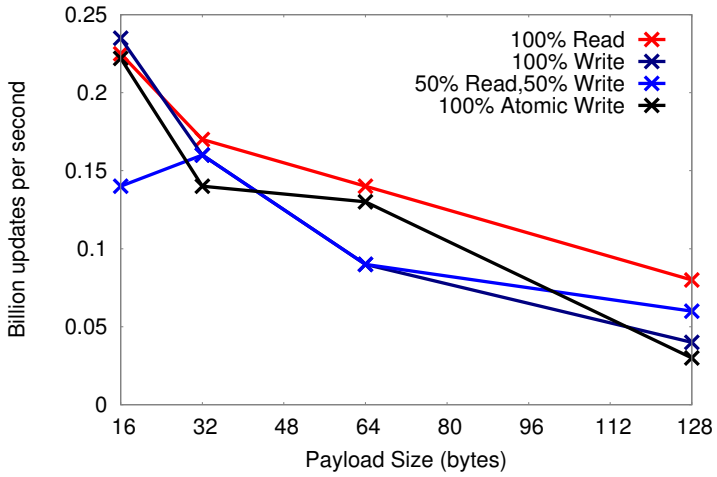




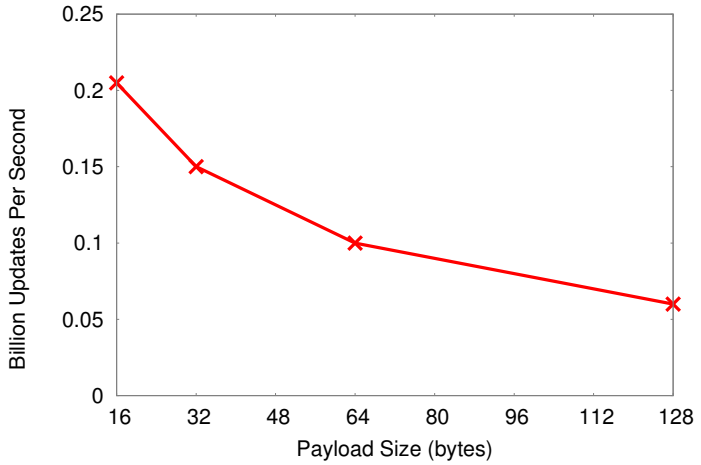
Experimental Dataset

- Random Graph
  - Generated follows GRAPH500 benchmark
- Data size:
  - Scale (log_2(Number of Vertices))
    - 23: 8 millions
    - 24: 16 millions
    - 25: 32 millions
  - Edge Factor(Average number of neighbors):
    - 2, 4, 8, 16

HMC Access Performance



Random Access Benchmark

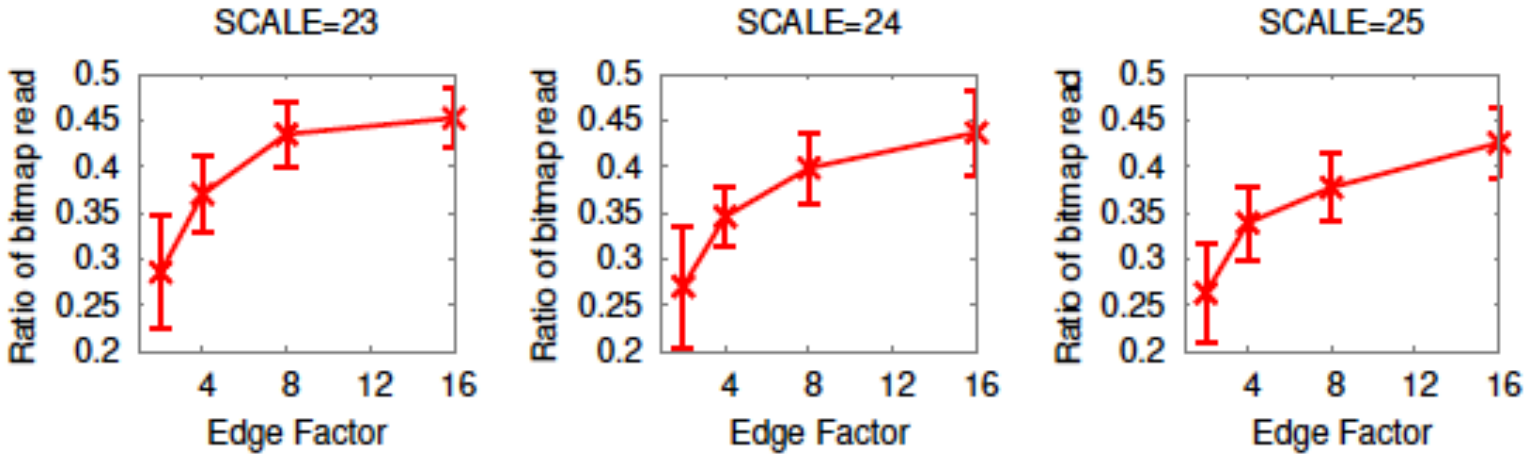


BFS

We achieve a balanced access distribution among vaults and banks

Two level Bit-map Gain (HMC Access)

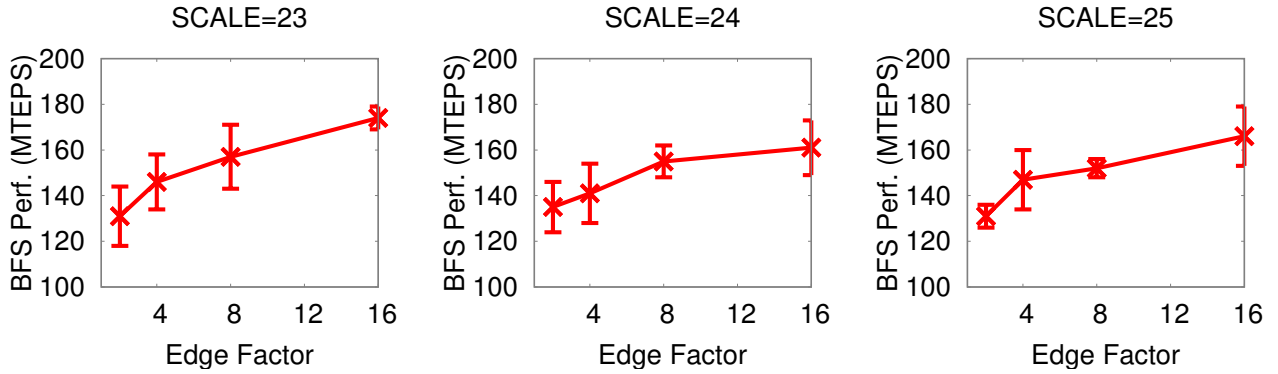
- Two-level bitmap can effectively reduce HMC request for bitmap scanning
  - Larger gain on large graph
  - Larger gain on more sparse graph



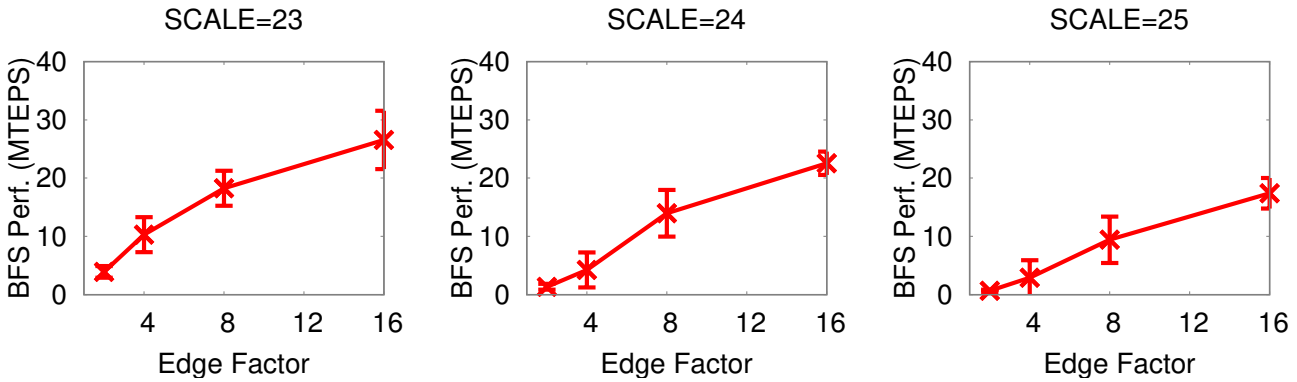
Two level Bit-map Gain (BFS)

- Two-level bitmap
  - Can effectively speed up BFS
  - Scalable to the graph scale
  - Less sensitive to the graph sparsity

Two Level



Single Level



Performance Comparison

System	Ours	FPGP[1]	GRAPHGEN[2]	Torous Graph [3]
Dataset	Random	Twitter	Twitter	Random
Scale	26	26	26	22
Edge Factor	16	35	16	16
Runtime(ms)	3.851 ms	121 ms	148 ms	76ms
Performance (MTEPS)	166.2	12.0	9.9	19.2

[1] FPGP: Graph processing framework on fpga a case study of breadth-first search

[2] Torusbfs: A novel message-passing parallel breadth-first search architecture on FPGAs

[3] Graphgen: An FPGA framework for vertex-centric graph computation

Conclusion

- HMC is a good fit for sparse graph traversal
  - Good random access performance
  - Support near-memory atomic operation
- BFS implementation using FPGA+HMC
  - Map-Reduce-like framework
  - Two-level Bitmap
- Analytical model
  - Access granularity
  - Bitmap granularity
- Experimental results verified the effectiveness of proposed techniques

Thanks!

We especially thank Micron for the donation of the development tool and hardware.