FPGA-Accelerated Transactional Execution of Graph Workloads

Xiaoyu Ma[1], Dan Zhang[1], and Derek Chiou[1,2]

[1]The University of Texas at Austin

[2]Microsoft

2017-2-24

Graph Applications

- Graphs are a core data structure for many problems
  - E.g. social/computer network, EDA, machine learning, ...
- Massive irregular DLP in large graphs
  - Same operations are applied to many nodes/edges
  - Dominated by pointer-based operations
- Graphs are inefficient on CPUs and GPUs
  - CPUs
    - Area and power inefficiency due to the focus on ILP
  - GPUs
    - Resource under-utilization due to irregularity

Our Approach

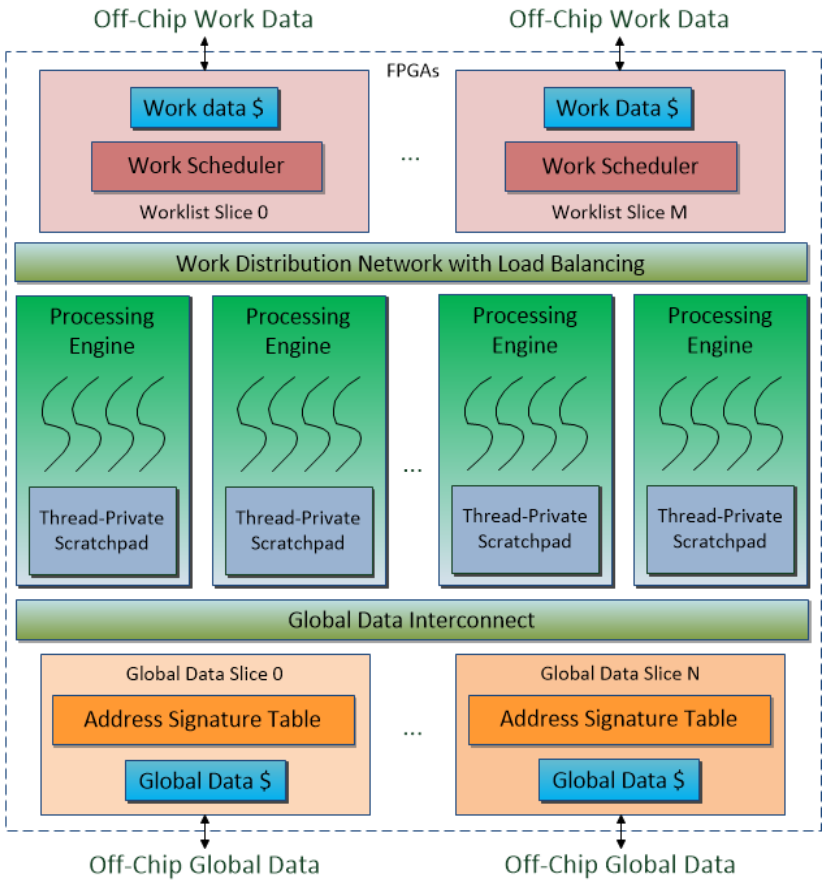
- A parallel architecture to exploit irregular DLP for graph acceleration
  - A large number of threads
    - Interleaved execution for latency hiding
  - Non-lockstep (asynchronous) execution for efficiency
    - No SIMD execution
  - Use transactions for synchronization
    - Each thread is a transaction
    - Hardware support
- FPGA-based Specialization
  - Reduce general-purpose compute overhead
  - Exploit fine-grained parallelism
  - Improve power efficiency

Challenges of Large-Scale Transactional Execution

1. Increased conflicts due to increased concurrency
   - Conflicts can hurt or negate the benefits of parallelization
2. Scalable conflict detection
   - Need to handle 100s/1000s concurrent threads
   - Conflict detection without bulk synchronization
     - KILO-TM (Fung et al. MICRO2011) on GPUs
     - Not applicable to asynchronous execution
3. Possible livelocks
   - Due to circular transaction aborting
4. On-chip buffer overflow
   - Due to too many/large transactions

Accelerator Architecture Overview

- Memory model
  - Global shared memory
  - Work data memory
  - Thread-private scratchpad
- Worklists
  - Work schedulers
    - Bucket priority scheduling
    - FIFO scheduling
  - Work distribution and load balancing
- Many processing engines
  - Lightweight; focus on TLP not ILP
  - Threads running in non-lockstep
  - HW multi-threading for latency tolerance
  - Many outstanding memory requests
- Synchronization
  - Optimistic parallel execution
  - Transactions supported by hardware Transactional Memory
- Implemented as FPGA synthesizable RTL using Bluespec

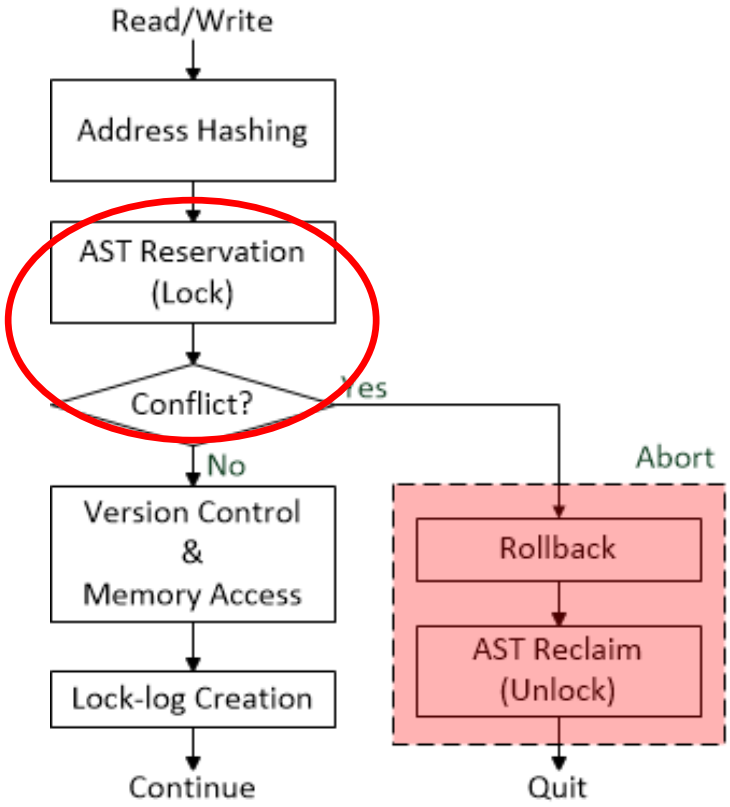


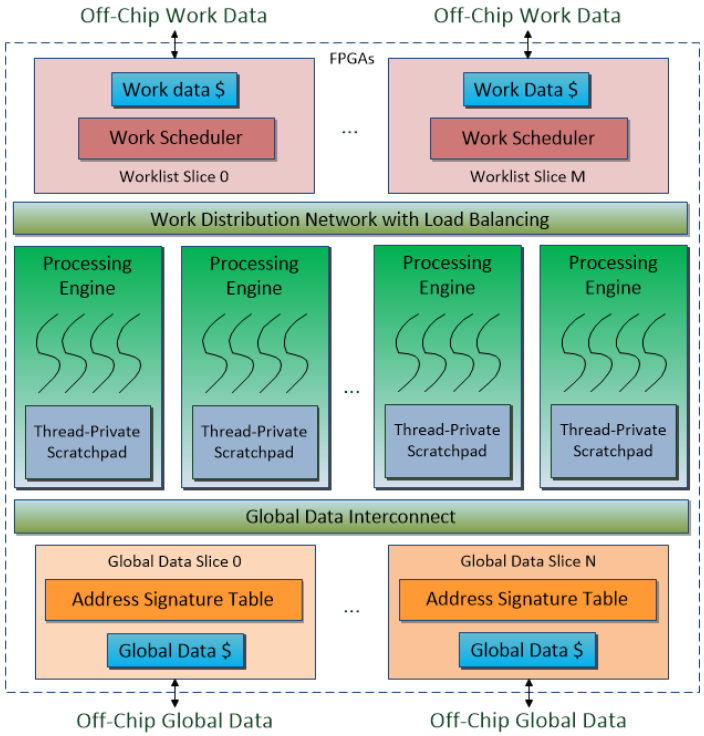
Key Techniques

- Conflict detection
  - A directory-based, eager approach with on-chip metadata
- Version management
  - Support both eager and lazy
  - An extra option that eliminates versioning overhead
- Dynamic concurrency control
  - Dynamically turn on/off threads based on conflict rate
    - Adapt the thread count to available parallelism
  - Also eliminate livelocks
    - In case of livelock, all transactions are aborts
    - Will keep reducing threads until single-thread execution
- A cache hierarchy for transactional states

Conflict Detection (1) Read/Write Execution Flow

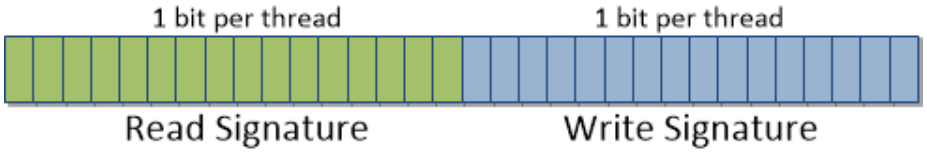
- Detect conflict before servicing each global memory request



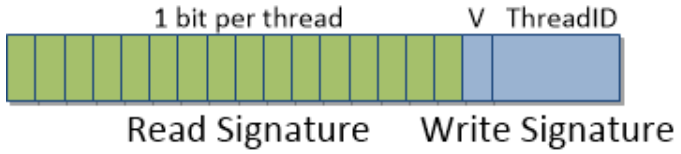


Conflict Detection (2)
Address Signature Table (AST)

- A distributed directory containing metadata for conflict detection
- How are read/write signatures recorded in each AST entry?
  - Prior work: per-thread bits for reads and writes
    - For example, 1K HW threads and 64K AST entries
    - AST size: (1K+1K)*64K/8 Bytes = 16M Bytes



    - Optimization: a thread-ID for all writes
      - AST size: (1K+10+1)*64K/8 Bytes = ~8M Bytes



  - Our approach: a thread-ID for both reads and writes
    - AST size: (10 + 1 + 1)*64K/8 Bytes = 96K Bytes
    - Enable small, on-chip AST for low-latency conflict detection
    - One problem: read-read (R-R) false conflict?
      - Having multiple thread-IDs can reduce but not eliminate R-R conflicts

Read/Write Signature

Eliminate Versioning Overhead

- Cautious transactions do not need version management
  - The decision of commit/abort can be made before any write occurs
- A transaction is cautious if
  - All reads occur before all writes in program order
  - Any write requires a preceding read to the same address
- Most graph applications are naturally cautious (Mendez-Lojo et al. PPoPP2010)
  - All non-cautious transactions are transformable to cautious transactions

Applications and Inputs

- Applications

Category	Application
Graph traversal	Vertex Exploration (VE)
Shortest path problems	Single Source Shortest Path (SSSP)
	Breadth First Search (BFS)
Connectivity Analysis	Connected Components (CC)
	Transitive Closure (TC)
Graph coloring	Bipartite Coloring (BC)

- Inputs

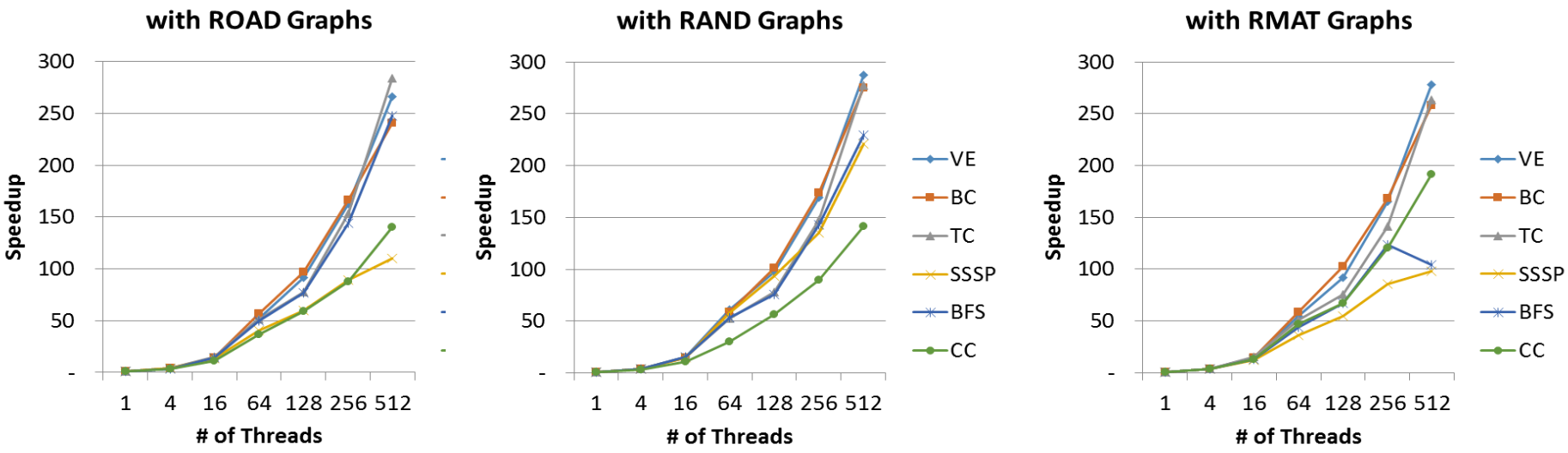
Graph	Characteristics	Size			
		S/L	Nodes	Edges	Bytes
Road	Uniform degree Large diameter	Small	1.9M	4.7M	136MB
		Large	24M	58M	1.6GB
Random	Uniform degree Random connectivity	Small	1M	4M	96MB
		Large	16M	64M	1.5GB
Scale-free (RMAT)	Power-law degree	Small	256K	2M	40MB
		Large	4M	32M	640MB

Evaluation

- Implemented using Bluespec System Verilog
  - Generated C++ for simulation (10X faster than RTL sim)
  - Generated Verilog for FPGA synthesis
    - 200MHz on Xilinx Virtex UltraScale 440
- Developed on FAbRIC (www.openfabric.org)
  - An open science FPGA cloud infrastructure hosted by Texas Advanced Compute Center (TACC)
- Evaluated up to 4 FPGAs only in simulation
  - Per-FPGA Configuration
    - 128 threads on 8 engines per FPGA
    - 2/4 worklist slices, each feeding 64/32 threads
    - 2 DDR4 SDRAM channels per FPGA
      - 25.6 GB/s peak memory bandwidth per FPGA

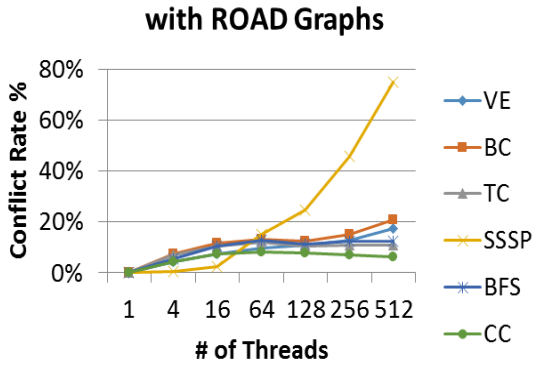
Simulation Results (1) Scalability

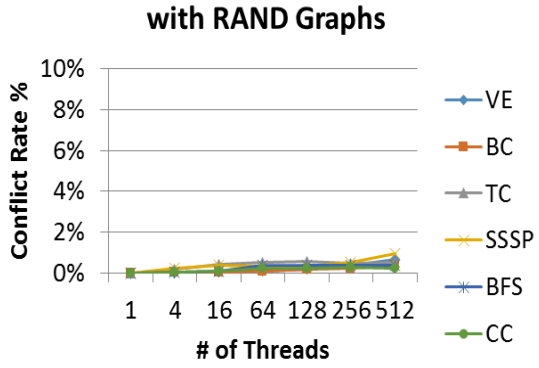
- Speedup over single-thread execution

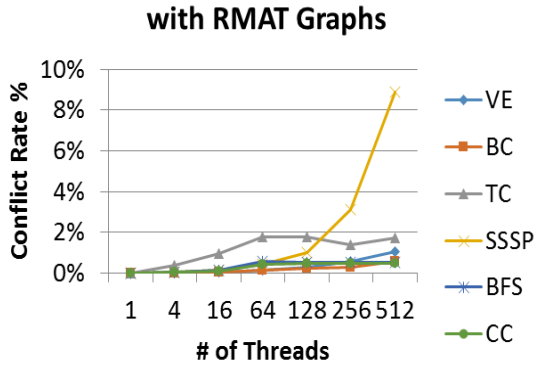


Simulation Results (2) Bottleneck Analysis

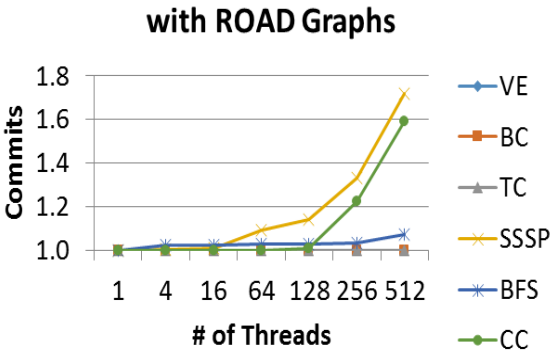
- Conflict rate (conflicts/commits)

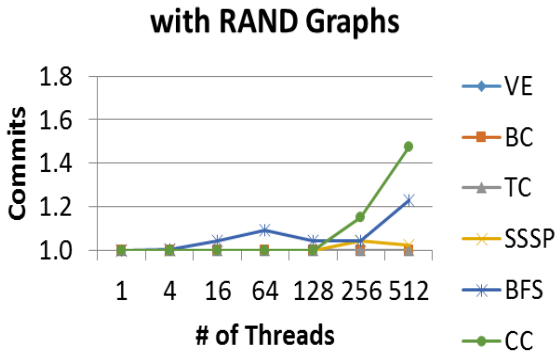


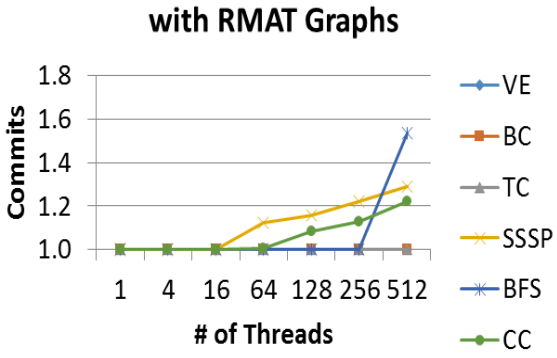




- Commits (divided by single-thread baseline)





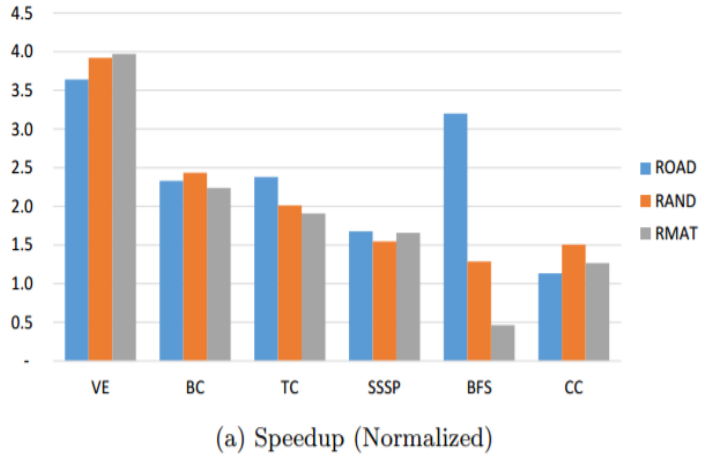


- Memory bandwidth
  - 25.6GB/s per-FPGA bandwidth is saturated at 128 threads

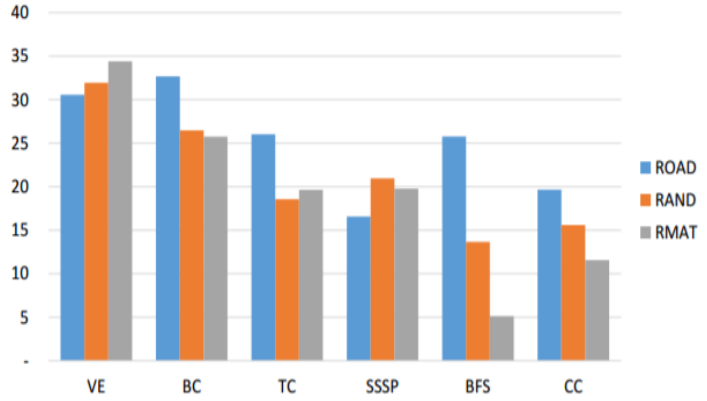
Simulation Results (3)
Dual-socket Comparison with Intel CPUs

- Baseline
  - Dual-socket Intel Haswell
    - 12 cores per socket
    - 2 DRAM channels per socket
  - Galois (Pingali et al. at UT-Austin)
    - Using fine-grained locks
- Our approach
  - Dual-socket FPGAs
  - 128 threads per socket
  - 2 DRAM channels per socket

	Clock	DRAM Bandwidth	Threads	Perf.	Perf. Per Watt
CPUs	2.6G	68.0GB/s	24	1X	1X
FPGAs	200M	51.2GB/s	256	2.14X	21.93X



(a) Speedup (Normalized)



(b) Performance-Per-Watt (Normalized)

Figure 8: FPGAs VS. CPUs

Conclusion

- We proposed
  - An approach of large-scale transactional execution
  - An architecture to achieve this approach
  - A set of techniques to reduce conflict overhead
- An FPGA-based implementation of our approach improves performance and energy efficiency compared to an Intel Haswell-based platform
- Future work
  - An extensive study of micro-architectural alternatives
  - Replace transactions with fine grain locks
  - More applications

Thanks!

- Questions?