# High Performance Computing Course Presentation

Hamid Reza Zohouri

01/02/2016

# Paper

- **A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services**
  - Also known as Microsoft's Catapult project
- Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, …
- ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), 14-18 June 2014, Minneapolis, MN, USA

# Abstract

- Designed and built a composable, reconfigurable fabric to accelerate portions of large-scale software services
- 6x8 2-D torus of Stratix V FPGAs embedded into a half-rack of 48 machines
  - One FPGA per server
  - Inter-FPGA network using 10 Gb SAS
- Deployed on 1632 servers for accelerating the Bing search engine
- The paper reports:
  - Requirements and architecture of the system
  - Engineering challenges and solutions for robustness
  - Performance, power, and resilience for ranking candidate documents
- Performance results:
  - 95% higher throughput for a fixed latency
  - 29% lower latency for a fixed throughput

# Introduction

- Improvement in server performance has slowed down largely due to power issues
- Specialization in *datacenters* is problematic since:
  - Homogeneity is desirable to reduce management issues
  - Non-programmable hardware features make evolution of datacenter services impractical
- FPGAs offer the potential for flexible acceleration of many workloads
- Fitting the accelerated function into the available reconfigurable area is challenging
  - Run-time reconfiguration is too slow
  - Multiple FPGAs offer scalable area but might not be cost/power efficient
  - Single FPGA per server restricts workload and might not be cost-effective

# Introduction (Cont.)

- Catapult: a reconfigurable fabric designed to balance these competing concerns
- A significant amount of Bing's page ranking is offloaded to groups of 8 FPGAs
- Ranking process:
  - Software-based processing
  - Converting to FPGA-suitable format
  - Routing to head FPGA
  - Going through 8-FPGA pipeline
  - Routing score back to requesting server
- Performance target: 2x throughput in the number of documents ranked per second per server, including software portions

# Introduction (Cont.)

- One big maintenance challenge is resilience
  - The fabric must stay available in presence of errors, failing hardware, reboots, and updates to the ranking algorithm
  - FPGAs can potentially corrupt their neighbors or crash the hosting servers during bitstream reconfiguration
- Solution: failure handling protocol
  - Reconfigure groups of FPGAs or remap services robustly
  - Recover from failures by remapping FPGAs
  - Report a vector of errors to the management software

# Hardware

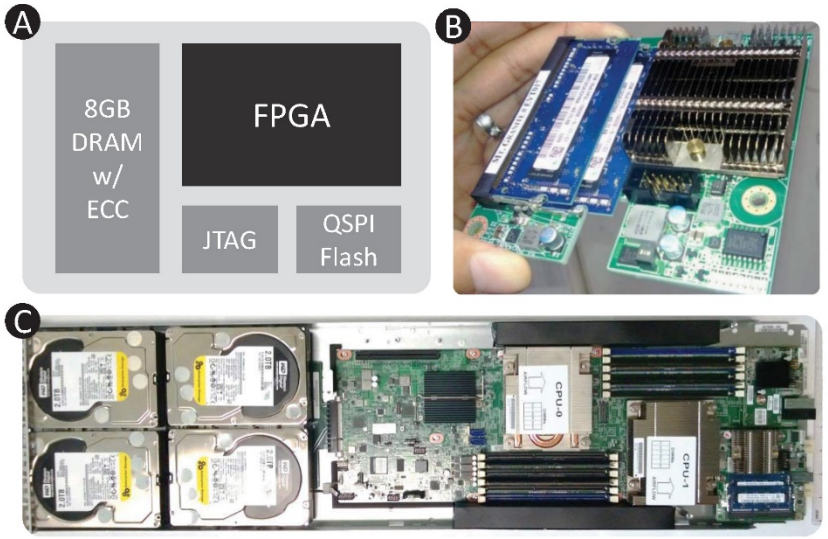
- Design requirements
  - Large amount of reconfigurable logic, due to datacenter workload complexity
  - Fitting within the datacenter architecture and cost constraints
- Option one: multiple FPGAs on one daughter-card for a subset of servers
  - Prototype of six Xilinx Virtex 6 SX315T FPGAs with mesh network through the FPGA's general-purpose I/Os
  - Straightforward but four problems:
    - Inelasticity: if a service needs more than 6 FPGAs, it cannot be mapped
    - If less FPGAs are needed, there will be unused capacity
    - Would not fit in ultra-dense servers unless with either heterogeneous servers in each rack or complete redesign of the whole datacenter
    - Single point of failure for the subset of servers

# Hardware (Cont.)

- Option two: small daughter-card in each server and direct connection between boards through a separate network
  - With a high-throughput and low-latency network, services can be mapped on multiple FPGAs across the network
  - This option allows efficient utilization of logic and does not exceed power, thermal, and space limits
  - This plan doesn’t surpasses the 30% limit on cost of ownership (TCO) including a limit of 10% for total server power

# Board Design

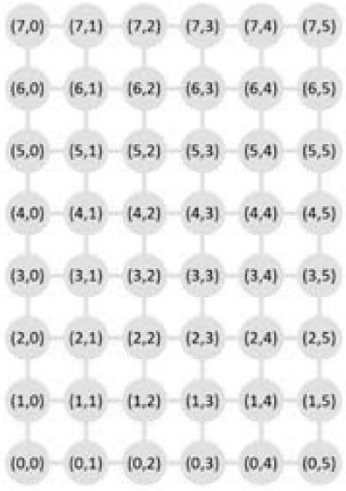
- PCI-E attachment
  - Minimize disruption to the motherboard
- FPGA: industrial-grade Stratix V D5
  - At the exhaust of the chassis
- Memory: 8 GB DDR3-1333 or 4 GB DDR3-1600 + ECC
- EMI shielding to the board to protect other server components
- PCI-E + Network through Mezzanine connector
- Power through PCI-E

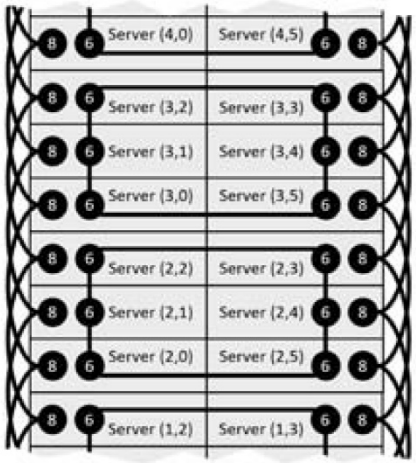



A: Board Overview
B: Board Picture
C: Server Picture

# Network Design

- Requirements:
  - Low latency
  - High bandwidth
  - Low component costs
  - Marginal servicing expenses
- 48 half-width 1U servers per rack
- 2 servers per 1U tray
- 2D 6x8 Torus network
  - Balanced routability and cabling complexity
- Mezzanine traces go through the motherboard to the back of the chassis
  - Exposed as two SFF-8088 SAS ports on passive backplane
  - 10 Gb/s signaling rates, 20 Gb/s bidirectional bandwidth, sub-µs latency
  - No additional NICs or switches



# Datacenter Deployment

- 17 racks for a total of 1632 servers
- Each server has:
  - Dual-socket motherboard
  - 2x 12-core Sandy Bridge CPUs
  - 64 GB RAM
  - 2x SSD & 4x HDD
  - 10 Gbps network card
- 7 defective card (0.4%) and one instance of defective cabling (0.03%) at deployment
  - No more defects after several months of operation

# Software Interface

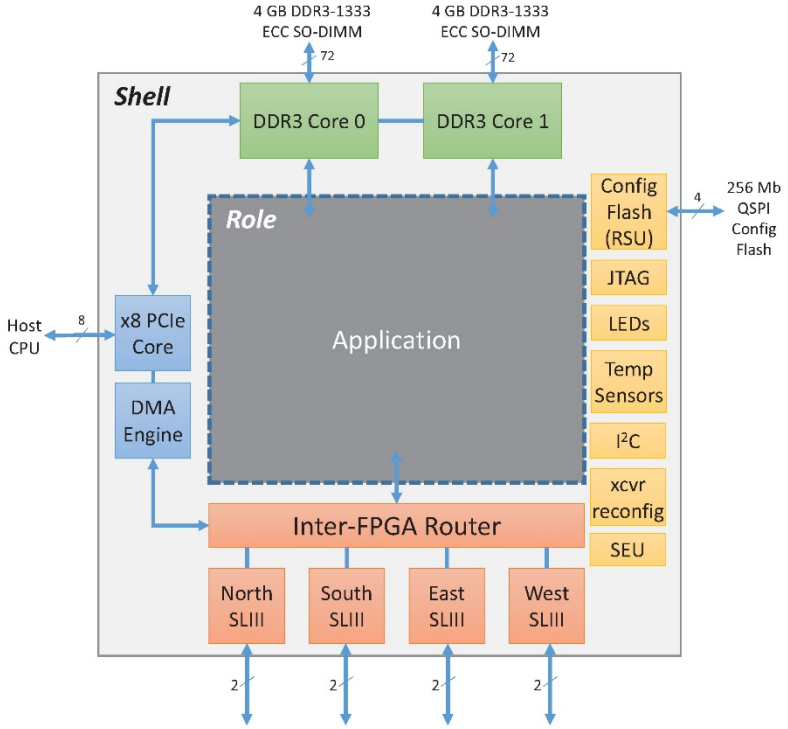
- Design requirements for CPU/FPGA communication interface:
  - Low latency, less than 10µs for transfers of 16 KB or less
  - Safe for multi-threading
- Low latency is achieved by avoiding system calls
- Thread-safety is achieved by:
  - One input and one output buffer in non-paged host memory
  - Supply FPGA with base pointer
  - Split buffers into 64 slots, 64 KB each
  - Allocate one or multiple slots to each thread
  - Status bits used to mark full slots
  - FPGA fairly selects and DMAs full slots and clears their full bit
  - FPGA writes back output to output buffer and awakens consumer thread through an interrupt

# Shell Architecture

- To avoid burdening the programmer with controlling data transfers, host-to-FPGA and inter-FPGA communication, logic is partitioned into:
  - Shell
    - Interface for accessing PCI-E, DRAM, Routing, etc.
    - 23% of FPGA space
  - Role
    - User logic
    - Rest of the FPGA

# Shell Architecture (Cont.)

- Two DRAM controllers
- Four high-speed serial links running SerialLite III
  - Inter-FPGA communication
  - Supports FIFO semantics, Xon/Xoff flow control, and ECC
- Router logic
- Reconfiguration logic
  - Read/write the configuration Flash
- PCI-E core + DMA
- SEU logic
  - Periodic scrubbing



# Shell Architecture (Cont.)

- Router is a crossbar between four network ports, PCI-E and Role
  - Static software-configured routing table
  - Virtual cut-through with no retransmission or source buffering
- Double error detection and single error correction on SL3 and DRAM
  - Flit-based
  - 20% lower SL3 bandwidth
  - Undetected 3-bit or longer errors will be detected by CRC at the end of transmission
  - Uncorrectable errors will result in packet drop and host timeout
    - Handled by higher-level protocol

# Software Infrastructure

- Three categories of changes to system software to:
  - Ensuring correct operation
  - Failure detection and recovery
  - Debugging
- Two new services
  - Mapping Manager: configuration of FPGAs with correct application image
  - Health Monitor: handling suspected failure

# Correct Operation

- Three sources of potential instability due to FPGA reconfiguration
  - Non-maskable interrupt due to the FPGA under reconfiguration appearing as a failed PCI-E device
  - Failing or reconfiguring FPGAs sending random traffic through SL3 that might appear valid to its neighbors
  - Asynchronous reconfiguration resulting in FPGAs receiving *old* data from their neighbors after reconfiguration
- Solutions
  - Drivers disables non-maskable interrupts before reconfiguration
  - FPGA to be reconfigured sends TX halt to its neighbors before reconfiguration
  - FPGA comes up with RX halt enabled after reconfiguration until Mapping Manager tells it to release it after all the group has been reconfigured

# Failure Detection and Recovery

- Whenever a system hangs and it is noticed:
  - Health Monitor is invoked
  - Hung machine is queried
    - If unresponsive: soft reboot or hard reboot or flagged for manual service
    - If responsive: machine reports a vector of error flags for Network, DRAM, PLL, PCI-E, etc.
  - Failed machine list is updated
  - Mapping Monitor is invoked
  - Based on failure type:
    - Service is relocated or
    - FPGA is reconfigured and corrupted states are cleared
  - Reported observation
    - Failures have been exceedingly rare
    - No hangs due to data corruption
    - Failures due to transient phenomena, primarily machine reboots due to maintenance or hangs

# Debugging Support

- The use of traditional FPGA debugging tools is limited by:
  - Finite buffering capacity
  - Impracticality of automatic recovery
  - Impracticality of putting USB JTAG units into each machine
- Solution: lightweight always-on Flight Data Recorder
  - Interesting information stored on FPGA on-chip memory
    - Power-on sequence
    - Intermittent errors
    - Circular buffer recording the most recent head and tail flits of all packets
      - Document trace ID
      - Transaction size
      - Travel direction
      - Other miscellaneous states
  - Streamed via PCI-E when needed

# Application Case Study

- A significant portion of Bing's *ranking* engine implemented on Catapult fabric
- FPGA part handwritten in Verilog
  - Partitioned across 7 FPGAs + 1 for redundancy
- Results identical to software-only, even bugs, except:
  - Uncontrollable incompatibilities, such as floating-point rounding artifacts caused by out-of-order operations
- Search query handling stages:
  - Front-end cache; if missed, sent to a top-level aggregator or TLA
  - TLA sends query to a large number of machines performing *selection*
    - Finding documents that match the query
  - Selected documents are sent to machine running the *ranking* service
    - Scores documents based on the query, FPGA-accelerated
  - Documents and ranks are sent back to TLA for sorting and display

# Application Case Study (Cont.)

- Ranking stages
  - Ranking server retrieves document and its metadata from local SSD
  - Document is split into several sections
  - A “hit vector" describing the locations of query words in each section is computed
  - A tuple is created for each word that matches a query term
  - Each tuple describes:
    - The relative offset from the previous tuple
    - The matching query term
    - A number of other properties
  - Feature computation; e.g. number of times a query word occurs in the document
  - Free-form expressions are computed by arithmetically combining the features
  - Features are sent to a machine-learned model to generate a score
    - Score determines document’s position in the returned result

# Application Case Study (Cont.)

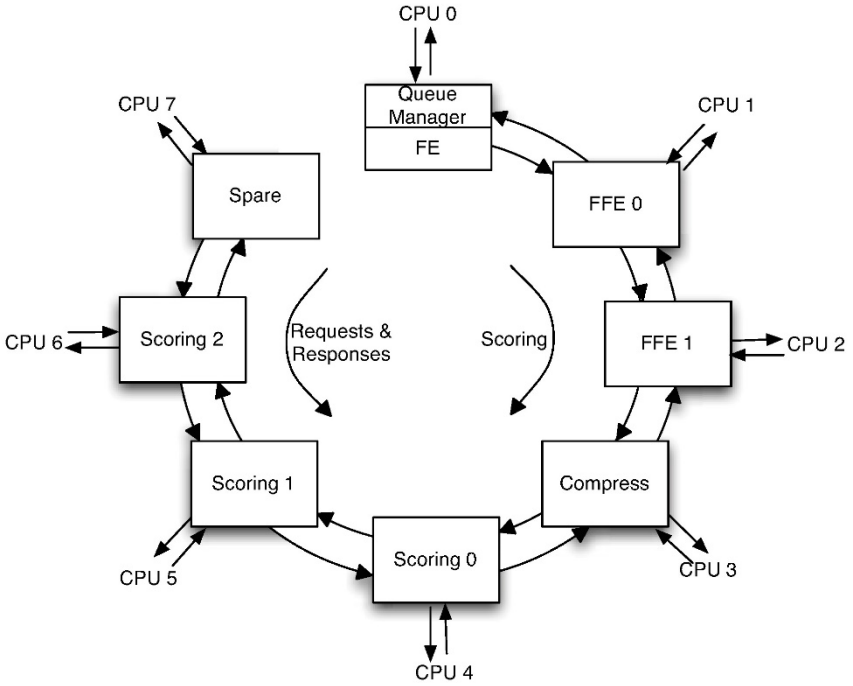
- FPGA-accelerated parts:
  - Most of the feature computations
  - All of the free-form expressions
  - All of the machine-learned model
- Software parts:
  - SSD lookup
  - Hit vector computation
  - A small number of software-computed features

# Software Interface

- Document is sent to fabric in compressed form, includes:
  - A header with basic request parameters
  - Software-computed features
  - Hit vector
- Due to slot-based DMA and latency of Feature Extraction being proportional to tuple count, compressed documents are truncated to 64 KB
  - Only unusual deviation from software-only ranker
  - In a real-word 210,000-document sample, only 300 documents were truncated
    - Average size is 6.5 KB, 99th percentile is at 53 KB
- The FPGA pipeline outputs a 4-byte float score which travels back through the inter-FPGA network to the head of the group
- A PCI-E DMA transfer moves the score, query ID, and performance counters back to the host

# Macropipeline

- Processing split into multiple macropipeline stages
- One macropipeline stage per FPGA
- Maximum 8µs per stage
- 200 MHz frequency
  - Maximum 1600 clocks per stage
- Stages shown in image



# Queue Manager and Model Reload

- Features, free forms, and scorers are not fixed
- Each set of these is called a *model*
- Models are selected based on language, query type, etc.
- Ranking request specifies model
- Multiple queues, one per model, in head-FPGA DRAM
- Queue Manager chooses which queue to process
- When queue empties or times out, another queue will be chosen
- Queue Manager sends Queue Reload command down the pipeline
- New instructions and data is loaded on FPGAs

# Queue Manager and Model Reload (Cont.)

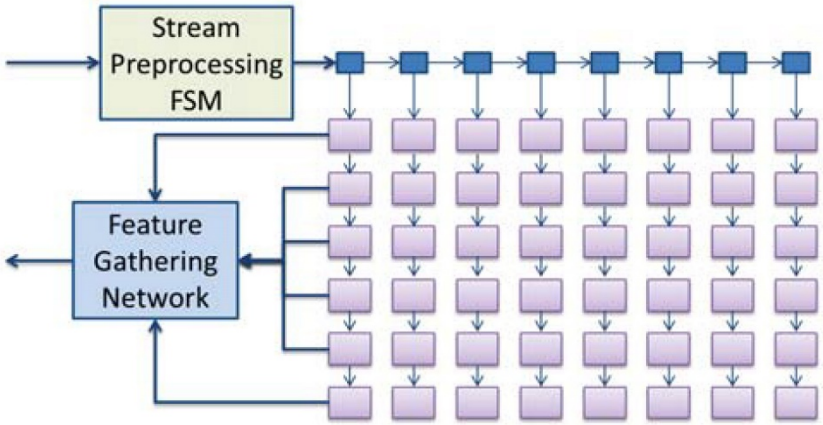
- Model Reload is an expensive operation (compared to document scoring)
- All of the embedded M20K Block RAMs on the FPGAs are reloaded with new data from DRAM
- Can take up to 250µs
- Queue Manager should minimize Model Reload
- Actual reload time depends on stage and model

# Feature Extraction

- First stage of the scoring acceleration pipeline
- Calculates numeric scores for a variety of features based on the query and document
- FPGA's advantage over software: feature extraction engines can run in parallel
  - A form of Multiple Instruction Single Data (MISD)
- 43 unique feature extraction state machines
- Up to 4,484 calculated features
- Some features that have similar computations
  - A single state machine for multiple features

# Feature Extraction (Cont.)

- Input is streamed into Stream Processing FSM
- Split into data and control tokens
- Processed in parallel by 43 unique state machines
- Generated features collected by Feature Gathering Network
- Forwarded to FFE stage



# Free Form Expressions

- Mathematical combinations of the extracted features
- Thousands of FFEs, ranging from very simple addition to large and complex floating point operations
  - Customized datapath for each is impractical
- Solution one: multiple instances of off-the-shelf soft-core processors
  - Too slow for large number of threads and complex floating-point operations
- Solution two: custom multicore processor
  - 60 cores on a single FPGA
  - Three important characteristics
  - Capable of executing all expressions before the deadline

# Free Form Expressions (Cont.)

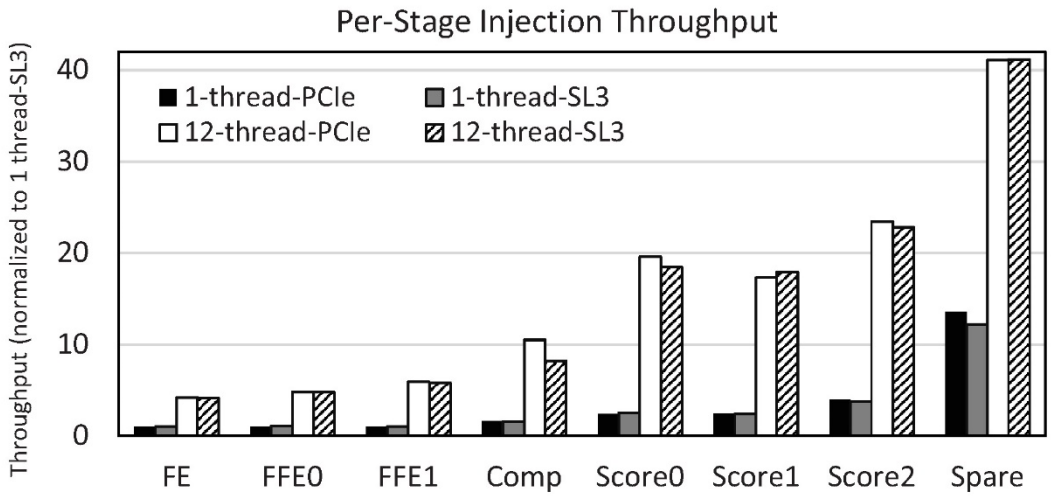
- Characteristic one:
  - 4 simultaneous threads per core that arbitrate for functional units clock by clock
  - When one thread is stalled, other threads can progress
  - Functional units are fully-pipelined
- Characteristic two:
  - Statistical thread prioritization instead of fair scheduling
  - Expressions with longest expected latency mapped to Thread 0 slot on all cores
  - Then Thread 1 slot and others are filled
  - When all slots are full, new threads are mapped to the end of Thread 0 to 3

# Free Form Expressions (Cont.)

- Characteristic three:
  - Longest-latency expressions are split across multiple FPGAs
- Multiple cores share the same complex arithmetic block to save FPGA area
  - Fair arbitration for the block using round-robin
  - Block consists of units for *ln*, *fpdiv*, *exp*, and *float-to-int*
  - *Pow*, *integer divide*, and *mod* are implemented using other instructions, rather than dedicated units
  - Complex block also contains the feature storage tile (FST) which is double-buffered

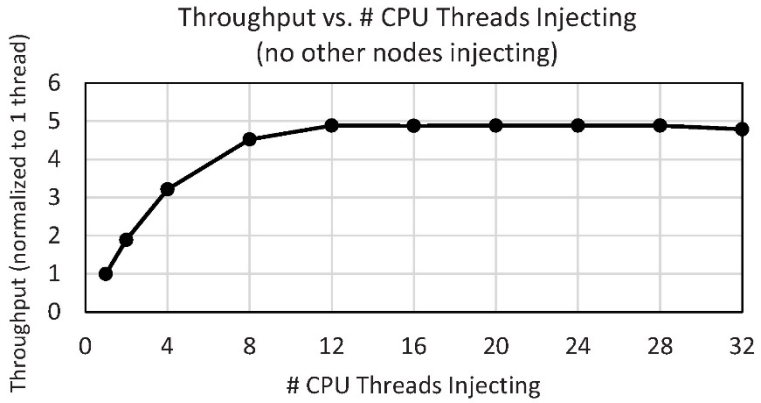
# Evaluation

- Single-stage test
- Per-stage injection throughput
- Real-world traces
- Results show significant variation
- FE is critical path

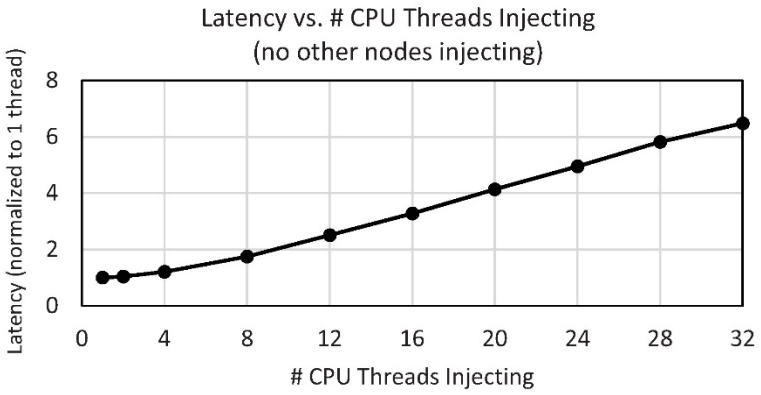


# Evaluation (Cont.)

- Full pipeline test
- Single injector (FE)
- Varying number of threads
- Throughput saturates at 12 threads
- More than 12 threads, throughput is limited by FE performance

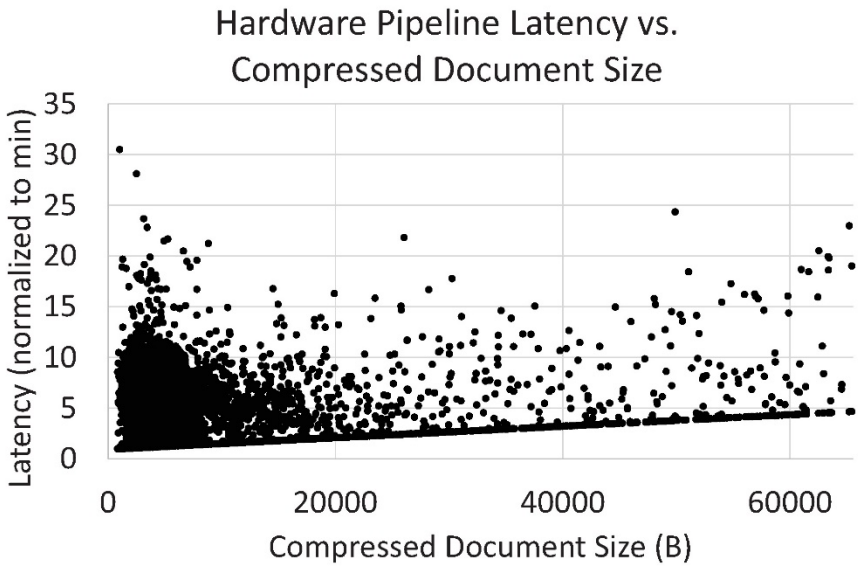



- Latency is from software's point of view
- From request until reply
- Increases with number of threads



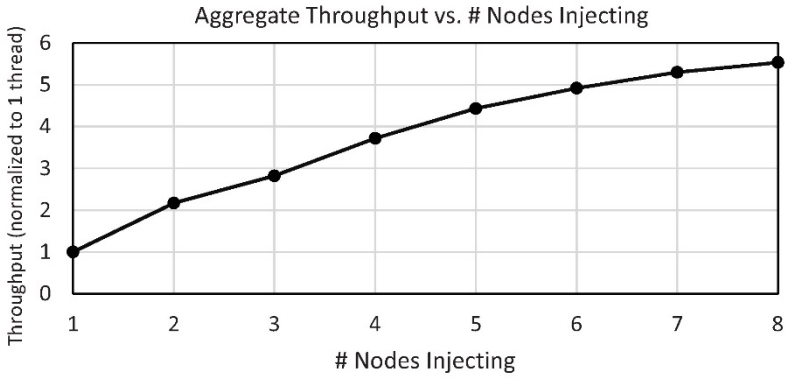
# Evaluation (Cont.)

- Document size vs. latency
- A minimum latency proportional to document size
  - Buffering and streaming
- A variable component that depends on input

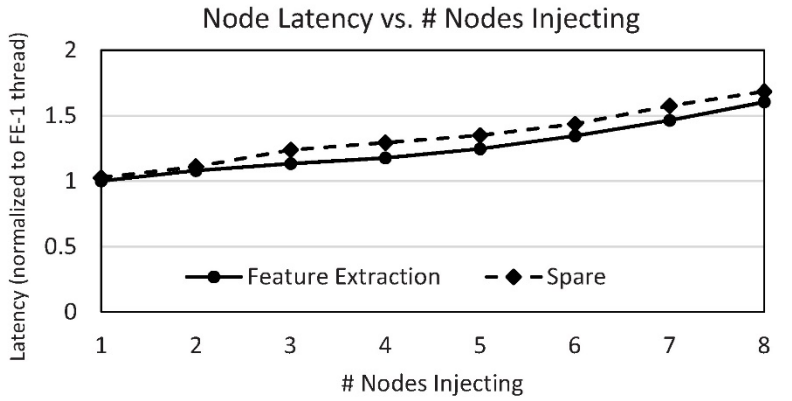


# Evaluation (Cont.)

- Full pipeline test
- Multiple injectors
- Maximum when all nodes are injecting
  - Same throughput as FE

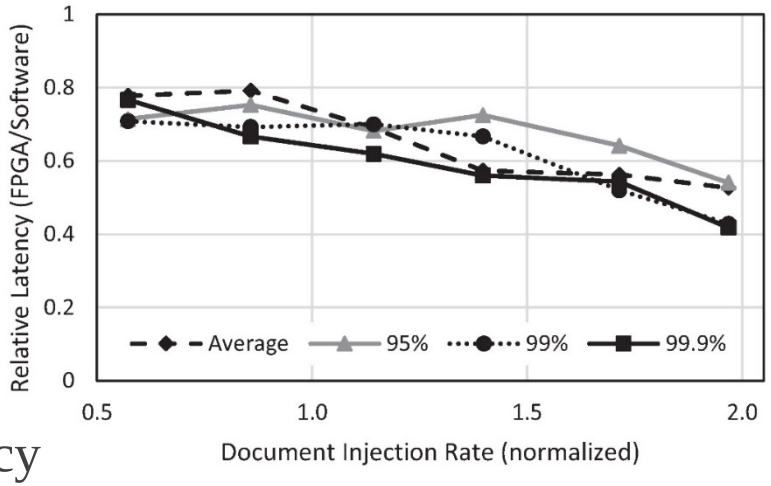



- Latency from head (FE) and tail's (Spare) point of view
- Tail has slightly higher latency due to shared channel with responses



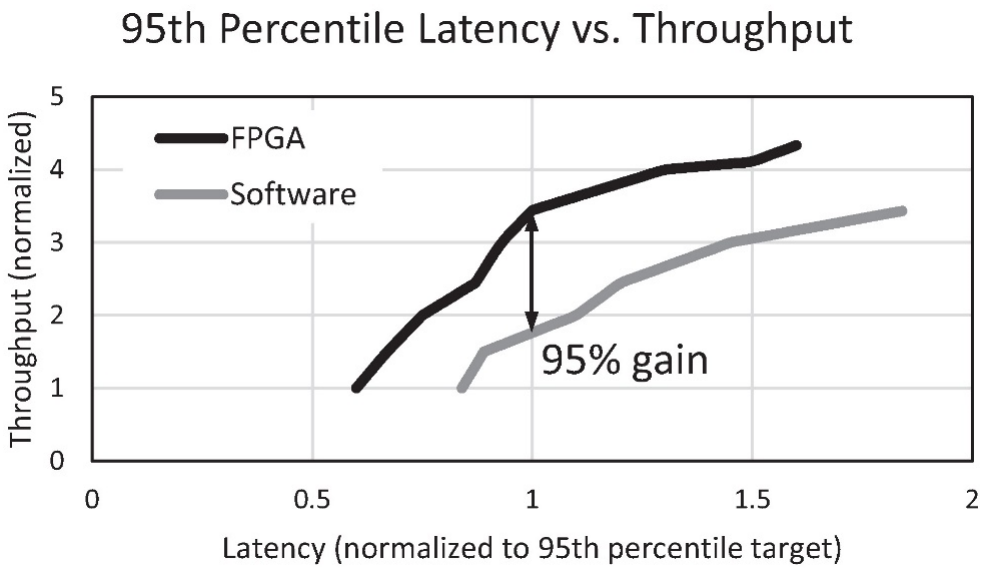
# Evaluation (Cont.)

- Software-only vs. FPGA-accelerated comparison
- 1,632 servers
  - 672 run the ranking service
- FPGA reduces latency by 29% at 95th percentile distribution
- Significantly reduces worst-case latency
- More improvement at higher injection rates
- Software latency highly variable at high load
  - Due to memory contention
- FPGA latency highly stable



# Evaluation (Cont.)

- 1 is maximum latency tolerated by Bing at 95th percentile
- FPGA improves throughput by 95% at this latency



# Evaluation (Cont.)

- FPGA statistics
- Power usage is 22.7 W using a “power virus” bitstream
  - Maximum area and activity
- Pipeline seems to be highly unbalanced
- FE is critical path by wasn’t split into multiple FPGAs
  - Yet scoring was split even though it has highest throughput
- Operating frequencies are lower than goal (200 MHz)
- DSPs are hardly used even though arithmetic operations were a bottleneck

| | FE | FFE0 | FFE1 | Comp | Scr0 | Scr1 | Scr2 | Spare |
|---|---|---|---|---|---|---|---|---|
| **Logic (%)** | 74 | 86 | 86 | 20 | 47 | 47 | 48 | 10 |
| **RAM (%)** | 49 | 50 | 50 | 64 | 88 | 88 | 90 | 15 |
| **DSP (%)** | 12 | 29 | 29 | 0 | 0 | 0 | 1 | 0 |
| **Clock (MHz)** | 150 | 125 | 125 | 180 | 166 | 166 | 166 | 175 |

# Conclusion

- Project goal was to determine:
  - Problems that should be solved for large-scale deployment of FPGAs
  - Whether significant performance improvements are achievable for large-scale production workloads
- GPUs were not deployed since:
  - High power consumption for conventional datacenter workloads
  - Not necessarily suitable for latency-sensitive application
- Showed that:
  - A significant portion of a complex datacenter service can be efficiently mapped to FPGAs, with the help of an inter-FPGA network
  - Resilience can be ensured when reconfiguring FPGAs, using a high-level safety protocol
  - FPGA can increase throughput by 95% at comparable latency to software
  - Less than 10% increase in power consumption and less than 30% in cost

# Conclusion (Cont.)

- Reconfigurable fabrics are a viable path forward with the imminent death of Moore's Law
- Reconfigurability is a critical means to keep pace with the rapidly-changing datacenter services
- A major challenge in the long term is FPGA programmability
  - Requires extensive hand-coding in RTL and manual tuning
  - For now, domain-specific languages (like OpenCL), HLS and libraries of reusable components might be sufficient
  - Longer term, more integrated development tools will be necessary
- With death of Moor's Law, compilation to a combination of hardware and software will be commonplace