A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services

Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, Doug Burger

Abstract

To advance datacenter capabilities beyond what commodity server designs can provide, we have designed and built a composable, reconfigurable fabric to accelerate large-scale software services. Each instantiation of the fabric consists of a 6x8 2-D torus of high-end field programmable gate arrays embedded into a half-rack of 48 servers. This reconfigurable fabric was deployed in a bed of 1,632 servers and FPGAs in a production datacenter and successfully used to accelerate the ranking portion of the Bing web search engine by nearly a factor of two.

1. Introduction

The rate at which server performance improves has slowed considerably. This slowdown, due largely to power limitations, has severe implications for datacenter operators, who have traditionally relied on continuous performance and efficiency improvements in servers to make improved services economically viable. While specializing servers for specific scale workloads can provide efficiency gains, it is problematic for two reasons. First, homogeneity in the datacenter is highly desirable to reduce management issues and to provide a consistent platform for applications. Second, datacenter services evolve extremely rapidly, making non-programmable hardware features impractical. Thus, datacenter providers are faced with a conundrum: they need continued improvements in performance and efficiency, but cannot obtain those improvements from general-purpose systems.

Reconfigurable chips, such as Field Programmable Gate Arrays (FPGAs), offer the potential for flexible acceleration of many workloads. However, as of this writing, FPGAs have not been widely deployed as compute accelerators in either datacenter infrastructure or in client devices. One challenge traditionally associated with FPGAs is the need to fit the accelerated function into the available reconfigurable area on one chip. FPGAs can be virtualized using run-time reconfiguration to support more functions than could fit into a single device. However, current reconfiguration times for standard FPGAs are too slow to make this approach practical. Multiple FPGAs provide scalable area, but cost more, consume more power, and are wasteful when unneeded. On the other hand, using a single small FPGA per server restricts the workloads that may be accelerated, and may make the associated gains too small to justify the cost.

This paper describes a reconfigurable fabric (that we call Catapult for brevity) designed to balance these competing concerns. The Catapult fabric is embedded into each half-rack of 48 servers in the form of a small board with a medium-sized FPGA and local DRAM attached to each server. FPGAs are directly wired together in a 6x8 two-dimensional torus, allowing services to allocate groups of FPGAs to provide the necessary area to implement the desired functionality.

We evaluate the Catapult fabric by offloading a significant fraction of the Microsoft Bing ranker onto groups of eight FPGAs to support each instance of this service. When a server wishes to score (rank) a document, it performs the software portion of the scoring, converts the document into a format suitable for FPGA evaluation, and then injects the document to its local FPGA. The document is routed on the inter-FPGA torus network to the FPGA at the head of the ranking pipeline. After running the document through the eight-FPGA pipeline, the computed score is routed back to the requesting server. Although we designed the fabric for general-purpose service acceleration, we used web search to drive its requirements, due to both the economic importance of search and its size and complexity. We set a performance target that would be a significant boost over software—2x throughput in the number of documents ranked per second per server, including portions of ranking which are not offloaded to the FPGA.

One of the challenges of maintaining such a fabric in the datacenter is resilience. The fabric must stay substantially available in the presence of errors, failing hardware, reboots, and updates to the ranking algorithm. FPGAs can potentially corrupt their neighbors or crash their host servers during bitstream reconfiguration. We incorporated a failure handling protocol that can reconfigure groups of FPGAs or remap services robustly, recover from failures by remapping FPGAs, and report a vector of errors to the management software to diagnose problems.

We tested the reconfigurable fabric, search workload, and failure handling service on a bed of 1,632 servers equipped with FPGAs. The experiments show that large gains in search throughput and latency are achievable using the large-scale reconfigurable fabric. Compared to a pure software implementation, the Catapult fabric achieves a 95% improvement in throughput at each ranking server with an equivalent latency distribution—or at the same throughput, reduces tail latency by 29%. The system is able to run stably for long periods, with a failure handling service quickly reconfiguring the fabric upon errors or machine failures. The rest of this paper describes the Catapult architecture and our measurements in more detail.

2. Catapult Hardware

The acceleration of datacenter services imposes several stringent requirements on the design of a large-scale reconfigurable fabric. To succeed in the datacenter environment, an FPGA-based reconfigurable fabric must meet the following requirements: (1) preserve server homogeneity to avoid complex management of heterogeneous servers, (2) scale to very large workloads that may not fit into a single FPGA, (3) avoid consuming too much power, (4) avoid single points of failure, (5) provide positive return on investment (ROI), and (6) operate within the space and power confines of existing servers without hurting network performance or reliability.

Integration. There are several ways to integrate FPGAs into the datacenter—they can be clustered into FPGA-only racks, or installed into a few specialized servers per rack. Both methods violate datacenter homogeneity and are vulnerable to single points of failure, where the failure of a single FPGA-enabled rack/server can take down many conventional servers. In addition, communication to these specialized servers has the potential to create bottlenecks in the existing network. Instead, this work proposes an organization where one FPGA and local DRAM is embedded into each server, retaining server homogeneity while keeping communication local between the CPU and the FPGA using PCIe.

Scalability. On its own, integrating only one FPGA per server limits applications to those that can be mapped to a single FPGA. To overcome this limitation, a specialized network was built to facilitate FPGA-to-FPGA communication without impacting the existing network. Figure 1 illustrates how the 48 FPGAs are organized at the rack level. Each server has a single locally attached FPGA, and an inter-FPGA network allows services to be logically mapped across multiple FPGAs. A two-dimensional, 6x8 torus topology was selected, which balanced routability, resilience, and cabling complexity. Each inter-FPGA network link supports 20 Gb/s of bidirectional bandwidth at sub-microsecond latency, and consists only of passive copper cables, with no additional networking costs such as NICs or switches.

FPGA Board Design. Figure 2 shows the FPGA board and the server it installs into [2]. It incorporates an Altera Stratix V D5 FPGA [1] that has considerable reconfigurable logic, on-chip memory blocks, and DSP units. The 8 GB of DRAM consists of two dual-rank DDR3-1600 ECC SO-DIMMs. The PCIe and inter-FPGA network traces are routed to a mezzanine connector on the bottom of the board which plugs directly into the motherboard. To avoid changes to the server itself, the board was custom-designed to fit within a small 10cm x 9cm x 16mm slot occupying the rear of a 1U-1/2 wide server, which only offered sufficient power and cooling for a 25W PCIe peripheral. In addition, the FPGA is placed downstream of both CPUs' exhausts, making it challenging to ensure the FPGA does not exceed thermal and power limits.

Resiliency. At datacenter scales, providing resiliency is essential given that hardware failures occur frequently, while availability requirements are high. For instance, the fabric must stay available in the presence of errors, failing hardware, reboots, and updates to the algorithm. FPGAs can potentially corrupt their neighbors or crash the hosting servers if care is not taken during reconfiguration. Our reconfigurable fabric further requires a protocol to reconfigure groups of FPGAs and remap services to recover from failures, and to report errors to the management software.

Total Cost of Ownership. To balance the expected per-server performance gains versus the necessary increase in total cost of ownership (TCO), including both increased capital costs and operating expenses, aggressive power and cost goals were set to achieve a positive ROI. Given the sensitivity of cost numbers on elements such as production servers, exact dollar figures cannot be given; however, adding the FPGA card and network cost less than 30% in TCO, including a limit of 10% for total server power.

Datacenter Deployment. To test this architecture on a number of datacenter services at scale, we manufactured and deployed the fabric in a production datacenter. The deployment consisted of 34 populated pods of machines in 17 racks, for a total of 1,632 machines. Each server uses an Intel Xeon 2-socket EP motherboard, 12-core Sandy Bridge CPUs, 64 GB of DRAM, and two SSDs in addition to four HDDs. The machines have a 10 Gb network card connected to a 48-port top-of-rack switch, which in turn connects to a set of level-two switches. The daughtercards and cable assemblies were both tested at manufacture and again at system integration. At deployment, we discovered that 7 cards (0.4%) had a hardware failure, and that one of the 3,264 links (0.03%) in the cable assemblies was defective. Since then, over several months of operation, we have seen no additional hardware failures.

3. Application Case Study

To drive the requirements of our hardware platform, we ported a significant fraction of Bing's ranking engine onto the Catapult fabric. We programmed the FPGA portion of the ranking engine by

hand in Verilog, and partitioned it across seven FPGAs plus one spare for redundancy. Thus, the engine maps to rings of eight FPGAs on one dimension of the torus.

Our implementation produces results that are identical to software (even reproducing known bugs), with the exception of uncontrollable incompatibilities, such as floating-point rounding artifacts caused by out-of-order operations. Although there were opportunities for further FPGA-specific optimizations, we decided against implementing them in favor of maintaining consistency with software.

Bing search has a number of stages, many outside the scope of our accelerated ranking service. As search queries arrive at the datacenter, they are checked to see if they hit in a front-end cache service. If a request misses in the cache, it is routed to a top-level aggregator (TLA) that coordinates the processing of the query and aggregates the final result. The TLA sends the same query (through mid-level aggregators) to a large number of machines performing a selection service that finds documents (web pages) that match the query, and narrows them down to a relatively small number per machine. Each selected document and its query is sent to a separate machine running the ranking service (the portion that we accelerate with FPGAs) that produces a score for that document-query. The scores and document IDs are returned to the TLA that sorts them, generates appropriate captions, and returns the results.

The ranking service is performed as follows. When a document-query pair arrives at a ranking service server, the server retrieves the document and its metadata, which together is called a metastream, from the local solid-state drive. The document is processed into several sections, creating several metastreams. A “hit vector", which describes the locations of query words in each metastream, is computed. It consists of a tuple for each word in the metastream that matches a query term. Each tuple describes the relative offset from the previous tuple (or start of stream), the matching query term, and a number of other properties.

Many “features”, such as the number of times each query word occurs in the document, are then computed. Synthetic features, called free-form expressions (FFEs) are computed by arithmetically combining computed features. All the features are sent to a machine-learned model that generates a score. That score determines the document’s position in the overall ranked list of documents returned to the user.

We implemented most of the feature computations, all of the free-form expressions, and all of the machine-learned model on FPGAs. What remains in software is the SSD lookup, the hit vector computation, and a small number of software-computed features.

3.1 Software Interface

While the ranking service processes document-queries, transmitting a compressed form of the document saves considerable bandwidth. Each encoded document-query request sent to the fabric contains three sections: (i) a header with basic request parameters, (ii) the set of software-computed features, and (iii) the hit vector of query match locations for each document’s metastreams.

The header contains a number of necessary additional fields, including the location and length of the hit vector as well as the software-computed features, document length, and number of query terms. The software-computed features section contains one or more pairs of {feature id, feature value}

tuples for features which are either not yet implemented on the FPGA, or do not make sense to implement in hardware (such as document features which are independent of the query and are stored within the document.)

To save bandwidth, software computed features and hit vector tuples are encoded in three different sizes using two, four, or six bytes depending on the query term. These streams and tuples are processed by the feature extraction stage to produce the dynamic features. These, combined with the precomputed software features, are forwarded to subsequent pipeline stages.

Due to our DMA interface and given that the latency of Feature Extraction is proportional to tuple count, we truncate compressed documents to 64KB. This represents the only deviation of the accelerated ranker from the pure software implementation, but the effect on search relevance is extremely small. Figure 3 shows a CDF of all document sizes in a 210K doc sample collected from real-world traces. As shown, nearly all of the compressed documents are under 64KB (only 300 require truncation). On average, documents are 6.5KB, with the 99th percentile at 53KB.

For each request, the pipeline produces a single score (a 4B float) representing how relevant the document is to the query. The score travels back up the pipeline through the dedicated network to the FPGA that injected the request. A PCIe DMA transfer moves the score, query ID, and performance counters back to the host.

3.2 Macropipeline

The processing pipeline is divided into macropipeline stages, with the goal of each macropipeline stage not exceeding 8µs, and a target frequency of 200MHz per stage. This means that each stage has 1,600 FPGA clock cycles or less to complete processing. Figure 4 shows how we allocate functions to FPGAs in the eight-node group: one FPGA for feature extraction, two for free-form expressions, one for a compression stage that increases the efficiency of the scoring engines, and three to hold the machine-learned scoring models. The eighth FPGA is a spare which allows the Service Manager to rotate the ring upon a machine failure and keep the ranking pipeline alive.

3.3 Queue Manager and Model Reload

So far the pipeline descriptions assumed a single set of features, free form expressions and machine-learned scorer. In practice, however, there are many different sets of features, free forms, and scorers. We call these different sets models. Different models are selected based on each query, and can vary for language (e.g. Spanish, English, Chinese), query type, or for trying out experimental models.

When a ranking request comes in, it specifies which model should be used to score the query. The query and document are forwarded to the head of the processing pipeline and placed in a queue in DRAM which contains all queries using that model. The Queue Manager (QM) takes documents from each queue and sends them down the processing pipeline. When the queue is empty or when a timeout is reached, QM will switch to the next queue. When a new queue (i.e. queries that use a different model) is selected, QM sends a Model Reload command down the pipeline, which will cause each stage to load the instructions and data needed to evaluate the query with the specified model.

Model Reload is a relatively expensive operation. In the worst case, it requires all of the embedded M20K RAMs to be reloaded with new contents from DRAM. On each board's D5 FPGA, there are 2,014 M20K RAM blocks, each with 20Kb capacity. Using the high-capacity DRAM configuration at

DDR3-1333 speeds, Model Reload can take up to 250µs. This is an order of magnitude slower than processing a single document, so the queue manager's role in minimizing model reloads among queries is crucial to achieving high performance. However, while model reload is slow relative to document processing, it is fast relative to FPGA configuration or partial reconfiguration, which ranges from milliseconds to seconds for the D5 FPGA. Actual reload times vary both by stage and by model. In practice model reload takes much less than 250µs because not all embedded memories in the design need to be reloaded, and not all models utilize all of the processing blocks on the FPGAs.

3.4 Feature Extraction

The first stage of the scoring acceleration pipeline, Feature Extraction (FE), calculates numeric scores for a variety of "features" based on the query and document combination. There are potentially thousands of unique features calculated for each document, as each feature calculation produces a result for every stream in the request. Furthermore, some features produce a result per query term as well. Our FPGA accelerator offers a significant advantage over software because each of the feature extraction engines can run in parallel, working on the same input stream. This is effectively a form of Multiple Instruction Single Data (MISD) computation.

We currently implement 43 unique feature extraction state machines, with up to 4,484 features calculated and used by downstream FFE stages in the pipeline. Each state machine reads the stream of tuples one at a time and performs a local calculation. For some features that have similar computations, a single state machine is responsible for calculating values for multiple features. As an example, the *NumberOfOccurences* feature simply counts up how many times each term in the query appears in each stream in the document. At the end of a stream, the state machine outputs all non-zero feature values—for *NumberOfOccurences*, this could be up to the number of terms in the query.

To support a large collection of state machines working in parallel on the same input data at a high clock rate, we organize the blocks into a tree-like hierarchy and replicate the input stream several times. Figure 5 shows the logical organization of the FE hierarchy. Input data (the hit-vector) is fed into a Stream Processing state machine which produces a series of control and data messages that the various feature state machines process. Each state machine processes the stream a rate of 1-2 clock cycles per token. When a state machine finishes its computation, it emits one or more feature index and values that are fed into the Feature Gathering Network that coalesces the results from the 43 state machines into a single output stream for the downstream FFE stages. Inputs to FE are double-buffered to increase throughput.

3.5 Free Form Expressions

Free Form Expressions (FFEs) are mathematical combinations of the features extracted during the Feature Extraction stage. FFEs give developers a way to create hybrid features that are not conveniently specified as feature extraction state machines. There are typically thousands of FFEs, ranging from very simple (such as adding two features) to large and complex (thousands of operations including conditional execution and complex floating point operators such as *ln*, *pow*, and *fpdiv*). FFEs vary greatly across different models, so it is impractical to synthesize customized datapaths for each expression.

One potential solution is to tile many off-the-shelf soft processor cores (e.g., Nios II), but these single-threaded cores are not efficient at processing thousands of threads with long latency floating

point operations in the desired amount of time per macropipeline stage (8µs). Instead, we developed a custom multicore processor with massive multithreading and long-latency operations in mind. The result is the FFE processor shown in Figure 6. As we will describe in more detail, the FFE microarchitecture is highly area-efficient, allowing us to instantiate 60 cores on a single D5 FPGA.

There are three key characteristics of the custom FFE processor that makes it capable of executing all of the expressions within the required deadline. First, each core supports 4 simultaneous threads that arbitrate for functional units on a cycle-by-cycle basis. While one thread is stalled on a long operation such as *fpdiv* or *ln*, other threads continue to make progress. All functional units are fully-pipelined, so any unit can accept a new operation on each cycle.

Second, rather than fair thread scheduling, threads are statically prioritized using a priority encoder. The assembler maps the expressions with the longest expected latency to Thread Slot 0 on all cores, then fills in Slot 1 on all cores, and so forth. Once all cores have one thread in each thread slot, the remaining threads are appended to the end of previously-mapped threads, starting again at Thread Slot 0.

Third, the longest latency expressions are split across multiple FPGAs. An upstream FFE unit can perform part of the computation and produce an intermediate result called a metafeature. These metafeatures are sent to the downstream FFEs like any other feature, effectively replacing that part of the expressions with a simple feature read.

Because complex floating point instructions consume a large amount of FPGA area, multiple cores (typically 6) are clustered together to share a single complex block. Arbitration for the block is fair with round-robin priority. The complex block consists of units for *ln*, *fpdiv*, *exp*, and *float-to-int*. *Pow*, *intdiv*, and *mod* are all translated into multiple instructions by the compiler to eliminate the need for expensive, dedicated units. In addition, the complex block contains the feature storage tile (FST). The FST is double-buffered, allowing one document to be loaded while another is processed.

3.6 Document Scoring

The last stage of the pipeline is a machine learned model evaluator which takes the features and free form expressions as inputs and produces single floating-point score. This score is sent back to the Search software, and all of the resulting scores for the query are sorted and returned to the user in sorted order as the sorted search results.

4. Evaluation

We evaluate the Catapult fabric by deploying and measuring the Bing ranking engine described in Section 3 on a bed of 1,632 servers with FPGAs, of which 672 run the ranking service. We compare the average and tail latency distributions of Bing's production-level ranker running with and without FPGAs on that bed. Figure 7 illustrates how the FPGA-accelerated ranker substantially reduces the end-to-end scoring latency relative to software for a range of representative injection rates per server used in production. For example, given a target injection rate of 1.0 per server, the FPGA reduces the worst-case latency by 29% in the 95th percentile distribution. The improvement in FPGA scoring latency increases further at higher injection rates, because the variability of software latency increases at higher loads (due to contention in the CPU's memory hierarchy) while the FPGA's performance remains stable.

Figure 8 shows the measured improvement in scoring throughput while bounding the latency at the 95th percentile distribution. For the points labeled on the x-axis at 1.0 (which represent the maximum latency tolerated by Bing at the 95th percentile), the FPGA achieves a 95% gain in scoring throughput relative to software.

Given that FPGAs can be used to improve both latency and throughput, Bing could reap the benefits in two ways: (1) for equivalent ranking capacity, fewer servers can be purchased (in the target above, by nearly a factor of two), or (2) new capabilities and features can be added to the software and/or hardware stack without exceeding the maximum allowed latency.

5. Related Work

Many other groups have worked on incorporating FPGAs into CPU systems to accelerate workloads in large-scale systems.

One challenge to developing a hybrid computing system is the integration of server-class CPUs with FPGAs. One approach is to plug the FPGA directly onto the native system bus, for example, in systems using AMD's HyperTransport [23, 10], or Intel's Front Side Bus [18] and QuickPath Interconnect (QPI) [15]. While integrating the FPGA directly onto the processor bus would reduce DMA latency, this latency is not the bottleneck in our application and would not significantly improve overall performance. In addition, attaching the FPGA to QPI would require replacing one CPU with an FPGA, severely impacting the overall utility of the server for applications which cannot use the FPGA.

IBM's Coherence Attach Processor Interface (CAPI) [26] and Convey's Hybrid-core Memory Interconnect (HCMI) [8] both enable advanced memory sharing with coherence between the FPGA and CPU. Since our ranking application only requires simple memory sharing, these mechanisms are not yet necessary but may be valuable for future applications.

Instead of incorporating FPGAs into the server, several groups have created network-attached FPGA appliances that operate over Ethernet or Infiniband. The Convey HC-2 [8], Maxeler MPC series [21], BeeCube BEE4 [5] and SRC MAPstation [25] are all examples of commercial FPGA acceleration appliances. While the appliance model appears to be an easy way to integrate FPGAs into the datacenter, it breaks homogeneity and reduces overall datacenter flexibility. In addition, many-to-one network communication can result in dropped packets, making the bounds on latencies much harder to guarantee. Finally, the appliance creates a single point of failure that can disable many servers, thus reducing overall reliability. For these reasons, we distribute FPGAs across all servers.

Several large systems have also been built with distributed FPGAs, including the Cray XD-1 [9], Novo-G [12], and QP [22]. These systems integrate the FPGA with the CPU, but the FPGA-to-FPGA communication must be routed through the CPU. Maxwell [4] is the most similar to our design, as it directly connects FPGAs in a 2-D torus using InfiniBand cables, although the FPGAs do not implement routing logic. These systems are targeted to HPC rather than datacenter workloads, but they show the viability of FPGA acceleration in large systems. However, datacenters require greater flexibility within tighter cost, power, and failure tolerance constraints than specialized HPC machines, so many of the design decisions made for these systems do not apply directly to the Catapult fabric.

FPGAs have been used to implement and accelerate important datacenter applications such as Memcached [17, 6] compression/decompression [14, 19], K-means clustering [11, 13], and web search.

Pinaka [29] and Vanderbauwhede, et. al [27] used FPGAs to accelerate search, but focused primarily on the Selection stage of web search, which selects which documents should be ranked. Our application focuses on the Ranking stage, which takes candidate documents chosen in the Selection stage as the input.

The FFE stage is a soft processor core, one of many available for FPGAs, including MicroBlaze [28] and Nios II [2]. Unlike other soft cores, FFE is designed to run a large number of threads, interleaved on a cycle-by-cycle basis.

The Shell/Role design is aimed at abstracting away the board-level details from the application developer. Several other projects have explored similar directions, including VirtualRC [16], CoRAM [7], BORPH [24], and LEAP [1].

6. Conclusions

FPGAs show promise for accelerating many computational tasks, but they have not yet become mainstream in commodity systems. Unlike GPUs, FPGAs' traditional applications, such as rapid ASIC prototyping and line-rate switching, are unneeded in high-volume client devices and servers. However, FPGAs are now powerful computing devices in their own right, suitable for use as fine-grained accelerators. This paper described a large-scale reconfigurable fabric intended for accelerating datacenter services. Our goal in building the Catapult fabric was to understand what problems must be solved to operate FPGAs at scale, and whether significant performance improvements are achievable for large-scale production workloads.

When we first began this investigation, we considered both FPGAs and GPUs as possible alternatives. Both classes of devices can support copious parallelism, as both have hundreds to thousands of arithmetic units available on each chip. We decided not to incorporate GPUs because the current power requirements of high-end GPUs are too high for conventional datacenter servers, but also because it was unclear that some latency-sensitive ranking stages (such as feature extraction) would map well to GPUs.

Our study has shown that FPGAs can indeed be used to accelerate large-scale services robustly in the datacenter. We have demonstrated that a significant portion of a complex datacenter service can be efficiently mapped to FPGAs using a low-latency interconnect to support computations that must span multiple FPGAs. Special care must be taken when reconfiguring FPGAs, or rebooting machines, so that they do not crash the host server or corrupt their neighbors. We implemented and tested a high-level protocol for ensuring safety when reconfiguring one or more chips. With this protocol and the appropriate fault handling mechanisms, we showed that a medium-scale deployment of FPGAs can increase ranking throughput in a production search infrastructure by 95% at comparable latency to a software-only solution. The added FPGA compute boards only increased power consumption by 10% and did not exceed our 30% limit in the total cost of ownership of an individual server, yielding a significant overall improvement in system efficiency.

We conclude that distributed reconfigurable fabrics are a viable path forward as increases in server performance level off, and will be crucial at the end of Moore’s Law for continued cost and capability improvements. Reconfigurability is a critical means by which hardware acceleration can keep pace with the rapid rate of change in datacenter services.

A major challenge in the long term is programmability. FPGA development still requires extensive hand-coding in RTL and manual tuning. Yet we believe that incorporating domain-specific languages such as Scala or OpenCL, FPGA-targeted C-to-gates tools such as AutoESL or Impulse C, and libraries of reusable components and design patterns, will be sufficient to permit high-value services to be productively targeted to FPGAs for now. Longer term, more integrated development tools will be necessary to increase the programmability of these fabrics beyond teams of specialists working with large-scale service developers. Within ten to fifteen years, well past the end of Moore's Law, compilation to a combination of hardware and software will be commonplace. Reconfigurable systems, such as the Catapult fabric presented here, will be necessary to support these hybrid computation models.

Acknowledgments

Many people across many organizations contributed to the construction of this system, and while they are too numerous to list here individually, we thank our collaborators in Microsoft Global Foundation Services, Bing, the Autopilot team, and our colleagues at Altera and Quanta for their excellent partnership and hard work. We thank Reetuparna Das, Ofer Dekel, Alvy Lebeck, Neil Pittman, Karin Strauss, and David Wood for their valuable feedback and contributions. We are also grateful to Qi Lu, Harry Shum, Craig Mundie, Eric Rudder, Dan Reed, Surajit Chaudhuri, Peter Lee, Gaurav Sareen, Darryn Dieken, Darren Shakib, Chad Walters, Kushagra Vaid, and Mark Shaw for their support.

References

[1] M. Adler, K. E. Fleming, A. Parashar, M. Pellauer, and J. Emer, "Leap Scratchpads: Automatic Memory and Cache Management for Reconfigurable Logic," in Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ser. FPGA '11, 2011.

[2] Nios II Processor Reference Handbook, 13th ed., Altera, 2014.

[3] Stratix V Device Handbook, 14th ed., Altera, 2014.

[4] R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. Mccormick, G. Smart, R. Smart, A. Cantle, R. Chamberlain, and G. Genest, "Maxwell - a 64 FPGA Supercomputer," Engineering Letters, vol. 16, pp. 426–433, 2008.

[5] BEE4 Hardware Platform, 1st ed., BEECube, 2011.

[6] M. Blott and K. Vissers, "Dataflow Architectures for 10Gbps Line-Rate Key-Value Stores," in HotChips 2013, August 2013.

[7] E. S. Chung, J. C. Hoe, and K. Mai, "CoRAM: An In-fabric Memory Architecture for FPGA-based Computing," in Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ser. FPGA '11, 2011.

[8] The Convey HC-2 Computer, Conv-12-030.2 ed., Convey, 2012.

[9] Cray XD1 Datasheet, 1st ed., Cray, 2005.

[10] DRC Accelium Coprocessors Datasheet, Ds ac 7-08 ed., DRC, 2014.

[11] M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski, “Algorithmic Transformations in the Implementation of K-Means Clustering on Reconfigurable Hardware,” in Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays, ser. FPGA ’01, 2001.

[12] A. George, H. Lam, and G. Stitt, “Novo-G: At the Forefront of Scalable Reconfigurable Supercomputing,” Computing in Science Engineering, vol. 13, no. 1, pp. 82–86, 2011.

[13] H. M. Hussain, K. Benkrid, A. T. Erdogan, and H. Seker, “Highly Parameterized K-means Clustering on FPGAs: Comparative Results with GPPs and GPUs,” in Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs, ser. RECONFIG ’11, 2011.

[14] IBM PureData System for Analytics N2001, WAD12353-USEN-01 ed., IBM, 2013.

[15] Intel, “An Introduction to the Intel Quickpath Interconnect,” 2009.

[16] R. Kirchgessner, G. Stitt, A. George, and H. Lam, “VirtualRC: A Virtual FPGA Platform for Applications and Tools Portability,” in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ser. FPGA ’12, 2012.

[17] M. Lavasani, H. Angepat, and D. Chiou, “An FPGA-based In-line Accelerator for Memcached,” Computer Architecture Letters, vol. PP, no. 99, pp. 1–1, 2013.

[18] L. Ling, N. Oliver, C. Bhushan, W. Qigang, A. Chen, S. Wenbo, Y. Zhihong, A. Sheiman, I. McCallum, J. Grecco, H. Mitchel, L. Dong, and P. Gupta, “High-performance, Energy-efficient Platforms Using In-socket FPGA Accelerators,” in International Symposium on Field Programmable Gate Arrays, ser. FPGA ’09, 2009.

[19] A. Martin, D. Jamsek, and K. Agarawal, “FPGA-Based Application Acceleration: Case Study with GZIP Compression/Decompression Streaming Engine,” in ICCAD Special Session 7C, November 2013.

[20] How Microsoft Designs its Cloud-Scale Servers, Microsoft, 2014.

[21] O. Pell and O. Mencer, “Surviving the End of Frequency Scaling with Reconfigurable Dataflow Computing,” SIGARCH Comput. Archit. News, vol. 39, no. 4, Dec. 2011.

[22] M. Showerman, J. Enos, A. Pant, V. Kindratenko, C. Steffen, R. Pennington, and W. Hwu, “QP: A Heterogeneous Multi-Accelerator Cluster,” 2009.

[23] D. Slogsnat, A. Giese, M. Nüssle, and U. Brüning, “An Open-source HyperTransport Core,” ACM Trans. Reconfigurable Technol. Syst., vol. 1, no. 3, Sep. 2008.

[24] H. K.-H. So and R. Brodersen, “A Unified Hardware/Software Runtime Environment for FPGA-based Reconfigurable Computers Using BORPH,” ACM Trans. Embed. Comput. Syst., vol. 7, no. 2, Jan. 2008.

[25] MAPstation Systems, 70000 AH ed., SRC, 2014.

[26] J. Stuecheli, “Next Generation POWER microprocessor,” in HotChips 2013, August 2013.

[27] W. Vanderbauwhede, L. Azzopardi, and M. Moadeli, “FPGA accelerated Information Retrieval: High-efficiency document filtering,” in Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on, Aug 2009, pp. 417–422.

[28] MicroBlaze Processor Reference Guide, 14th ed., Xilinx, 2012.

[29] J. Yan, Z.-X. Zhao, N.-Y. Xu, X. Jin, L.-T. Zhang, and F.-H. Hsu, “Efficient Query Processing for Web Search Engine with FPGAs,” in Proceedings of the 2012 IEEE 20th International Symposium on Field Programmable Custom Computing Machines, ser. FCCM ’12, 2012.

Emails

Andrew Putnam <anputnam@microsoft.com>
Adrian Caulfield <acaulfie@microsoft.com>
Eric Chung <erchung@microsoft.com>
Derek Chiou <dechiou@microsoft.com>
Kypros Constantinides <kypros@amazon.com>
John Demme <jdd@cs.columbia.edu>
Hadi Esmaeilzadeh <hadi@cc.gatech.edu>
Jeremy Fowers <jfowers@microsoft.com>
Gopi Prashanth Gopal <gopiprashanth@gmail.com>
Jan Gray <jsgray@acm.org>
Michael Haselman <mikehase@microsoft.com>
Scott Hauck <hauck@uw.edu>
Stephen Heil <Stephen.Heil@microsoft.com>
Amir Hormati <hormati@gmail.com>
Joo-Young Kim <jooyoung@microsoft.com>
Sitaram Lanka <slanka@microsoft.com>
James Larus <james.larus@epfl.ch>
Eric Peterson <Eric.Peterson@microsoft.com>
Simon Pope <Simon.Pope@microsoft.com>
Aaron Smith <aaron.smith@microsoft.com>
Jason Thong <a-jathon@microsoft.com>
Phillip Yi Xiao <phxiao@microsoft.com>
Doug Burger <dburger@microsoft.com>

Biographies

Andrew Putnam is a Principal Research Hardware Development Engineer in the Microsoft Research NExT. He received a dual B.A/B.S. from the University of San Diego in 2003, and an M.S. and Ph.D. in Computer Science and Engineering from the University of Washington in 2006 and 2009 respectively. His research focuses on reconfigurable computing, future datacenter design, and computer architecture, with an emphasis on seeing research through from concept to prototype to technology transfer.

Adrian Caulfield is a Senior Research Hardware Development Engineer at Microsoft Research. Prior to joining Microsoft Research in 2013, Caulfield received his Ph.D. in Computer Engineering from the University of California, San Diego. His current research interests focus on computer architecture and

reconfigurable computing. Caulfield has also published extensively on non-volatile memories and storage systems design.

Eric Chung is a Researcher in Microsoft Research NExT. His research interests lie at the intersection of computer architecture and reconfigurable computing with FPGAs. He holds a Ph.D. in Electrical and Computer Engineering from Carnegie Mellon University and a B.S. from UC Berkeley.

Derek Chiou is a Principal Architect in Bing and an associate professor (on leave) at The University of Texas at Austin.

Kypros Constantinides is a Senior Hardware Development Engineer at Amazon Web Services. Before joining Amazon Web Services he worked at Microsoft Research where he contributed to the work published in this paper. Constantinides has a PhD degree in Computer Science and Engineering from the University of Michigan.

John Demme is an associate research scientist at Columbia University. His research interests include programming paradigms for spatial computing and data-intensive computing. He holds a PhD in computer science from Columbia University.

Hadi Esmaeilzadeh is the Allchin Family Early Career Professor of Computer Science at Georgia Institute of Technology. He has founded and directs the Alternative Computing Technologies (ACT) Lab. He is working on developing new technologies and cross-stack solutions to build the next generation computer systems for emerging applications. Esmaeilzadeh has a PhD in computer science from the University of Washington.

Jeremy Fowers is a research engineer in the Catapult team at Microsoft Research. He specializes in creating FPGA accelerators and frameworks to better utilize accelerators and IP. Jeremy received his Ph.D. from the University of Florida in 2014.

Gopi Prashanth Gopal has a Masters degree in computational engineering with over thirteen years of technical experience with increasing leadership responsibilities. Proven track record of managing and delivering mission critical programs for various blue-chip organizations in the space of large scale distributed systems, low latency application infrastructures, search technologies, online advertising, social games and computer vision.

Jan Gray is a consultant, computer architect, and software architect. He worked at Microsoft for 20 years on developer tools including the Visual C++ compiler, COM+, Common Language Runtime, and Parallel Computing Platform, and now focuses on design of FPGA-optimized soft processor arrays and compute accelerators. He has a B.Math. (CS/EE) from the University of Waterloo, Canada.

Michael Haselman is a Software Engineer in Microsoft ASG (Bing). Prior to joining Microsoft in 2013, he was a Senior Member of the Technical Staff at Sandia National Laboratories. He holds a Ph.D. in Electrical Engineering from the University of Washington.

Scott Hauck is a Professor in the Department of Electrical Engineering at the University of Washington, and a consultant with Microsoft on the Catapult project. Dr. Hauck's research focuses on FPGAs and reconfigurable computing.

Stephen Heil is a Principal Program Manager in Microsoft Research at Microsoft Corporation. In recent years his focus has been on the development of custom programmable hardware accelerators for use in Microsoft datacenters. His interests include field programmable gate arrays, application accelerators, and rack-scale system design. He has been involved in various aspects of computer system design and standards while working at Microsoft, Compaq, Panasonic and Unisys. Stephen has BS in Electrical Engineering Technology and Computer Science from The College of New Jersey (formerly Trenton State College).

Amir Hormati is currently a senior software engineer at Google working on large scale data analysis and storage platforms. Prior to joining Google in 2013, He worked at Microsoft Research as a research design engineer for 2 years. Amir received his M.S. and Ph.D. in Computer Science and Engineering from the University of Michigan in 2011, working on programming languages and compilation strategies for Heterogeneous architectures.

Joo-Young Kim is a Senior Research Hardware Development Engineer at Microsoft Research. Prior to joining Microsoft Research in 2012, Joo-Young received his Ph.D. in Electrical Engineering from Korea Advanced Institute of Science and Technology (KAIST). His current research interests focus on high performance accelerator design for data center workloads.

Sitaram Lanka is a Group Engineering Manager in Search Platform in Bing. Prior to joining Microsoft he was at Amazon.com and Tandem computers. He received a Ph.D. in Computer Science from University of Pennsylvania.

James Larus is Professor and Dean of the School of Computer and Communication Sciences (IC) at EPFL (École Polytechnique Fédérale de Lausanne). Prior to that, Larus was a researcher, manager, and director in Microsoft Research for over 16 years and an assistant and associate professor in the Computer Sciences Department at the University of Wisconsin, Madison. Larus has been an active contributor to the programming languages, compiler, software engineering, and computer architecture communities. He published over 100 papers (with 9 best and most influential paper awards), received 30 US patents, and served on numerous program committees and NSF, NRC, and DARPA panels. His book, Transactional Memory (Morgan Claypool) appeared in 2007. Larus received a National Science Foundation Young Investigator award in 1993 and became an ACM Fellow in 2006.

Eric Peterson is a Principal Mechanical Architect in the Microsoft Research Redmond Lab. His research interests are in new data center energy systems and system integration. He has over 30 years in high end computer system design with numerous resulting patents. He received his B.S. in Mechanical Engineering from Texas A&M University.

Simon Pope is a program manager for Microsoft. In his career, he has worked in government, enterprise, and academia and has held science, technology, and business roles across a variety of institutions and enterprises including Australia's DSTO and IBM T. J. Watson Research. He has degrees in computer science, cognitive science, and management, and his specialization is at the nexus of computing and psychology.

Aaron Smith is a Principal Research Software Development Engineer at Microsoft Research. He has over 15 years of experience in the research and development of advanced optimizing compilers and microprocessors. He received his Ph.D. in Computer Science from the University of Texas at Austin and

was General Chair of the 2015 IEEE/ACM International Symposium on Code Generation and Optimization.

Jason Thong received a Ph.D. in computer engineering from McMaster University, Canada in 2014. His academic research interests have included hardware acceleration, heterogeneous computing, and computer-aided design. Jason is currently a hardware design engineer with Microsoft Research in Canada. He develops low-latency PCIe interfaces jointly in hardware and software.

Phillip Yi Xiao is a Senior Software Engineer in Bing focusing on distributed computing architecture and machine learning.

Doug Burger is Director of the Hardware, Devices, and Experiences group in Microsoft Research NExT. Prior to joining Microsoft in 2008, he spend ten years on the faculty at the University of Texas at Austin, where he co-invented EDGE architecture and NUCA caches. He is an IEEE and ACM Fellow.

Figures

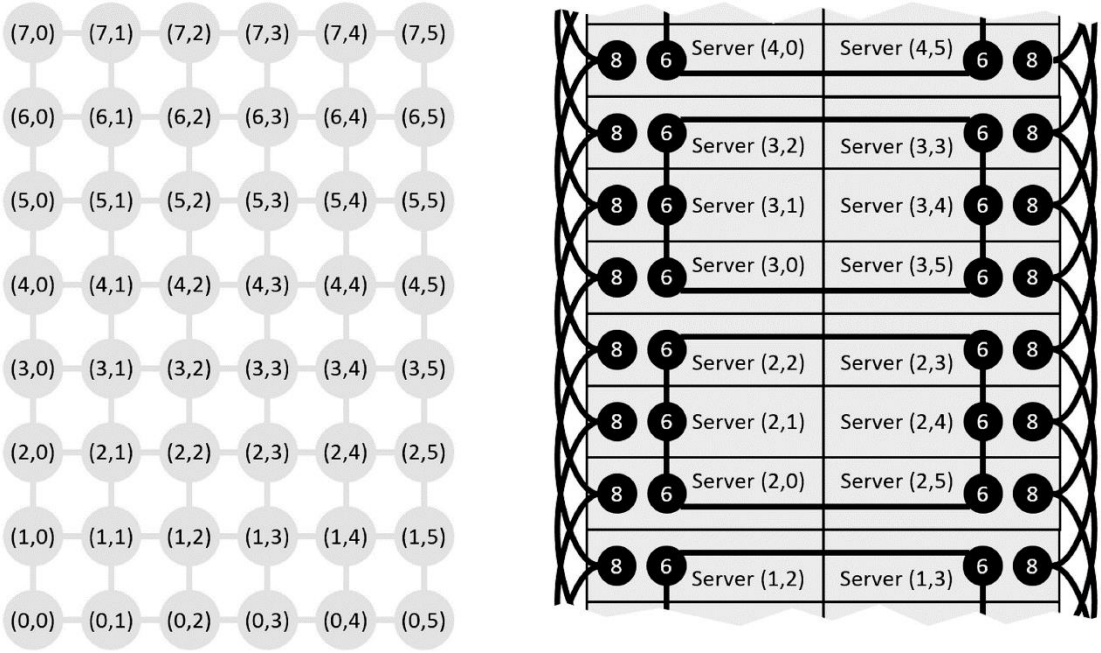

Figure 1: The logical mapping of the torus network, and the physical wiring on a pod of 2 x 24 servers

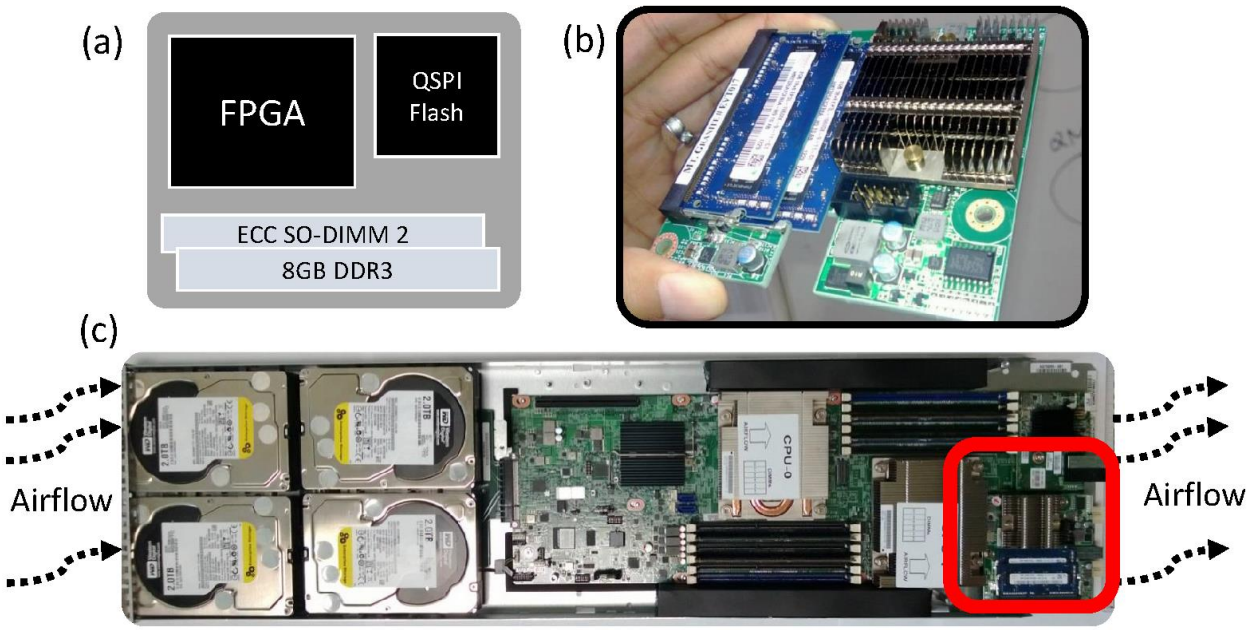

Figure 2: (a) A block diagram of the FPGA board. (b) A picture of the manufactured board. (c) A diagram of the 1~U, half-width server that hosts the FPGA board. The air flows from the left to the right, leaving the FPGA in the exhaust of both CPUs.

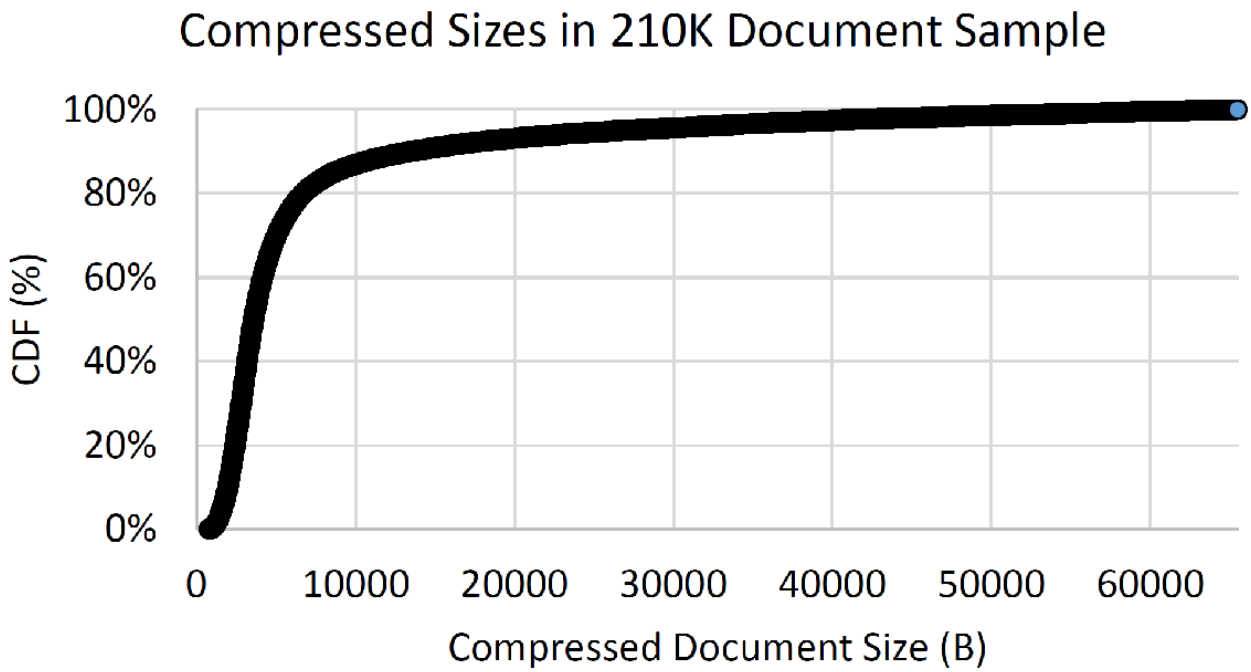

Figure 3: Cumulative distribution of compressed document sizes. Nearly all compressed documents are 64 kB or less.

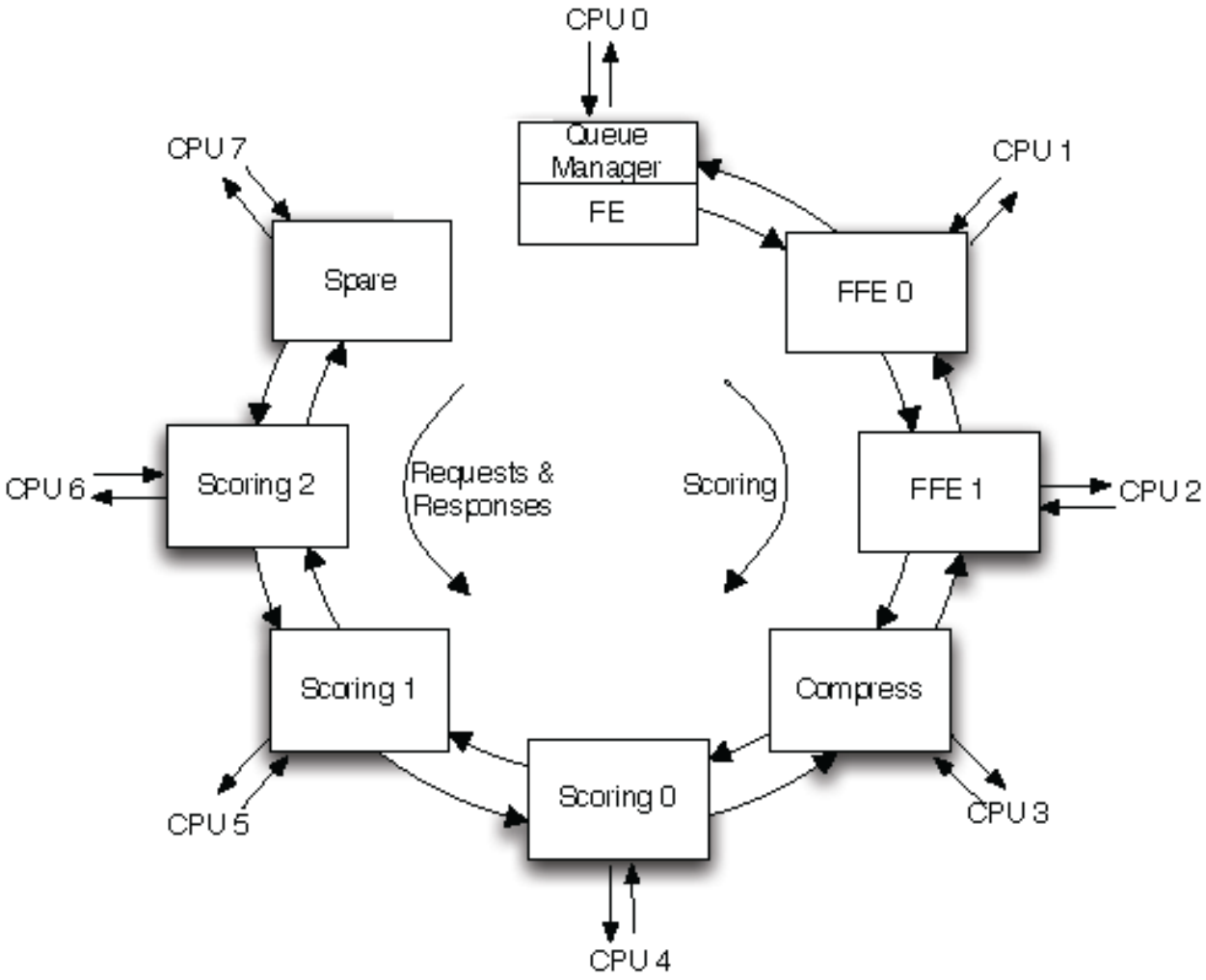

Figure 4: Mapping of ranking roles to FPGAs on the reconfigurable fabric.

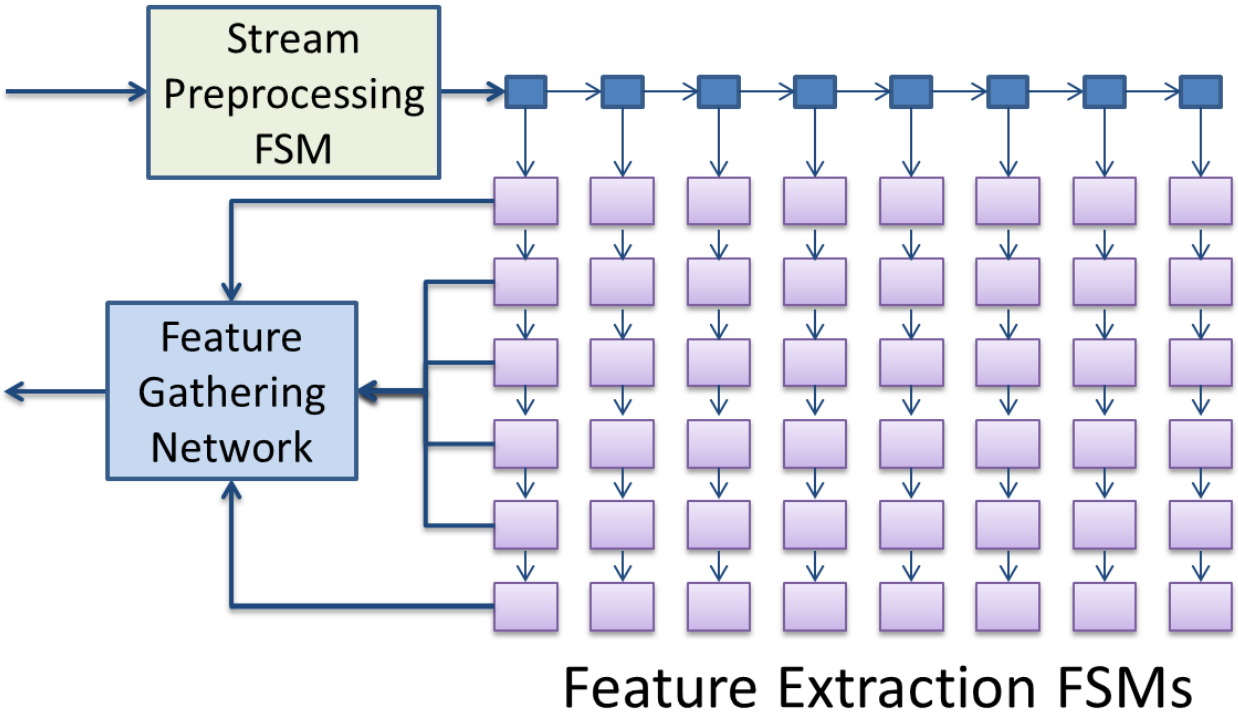

Figure 5: The first stage of the ranking pipeline. A compressed document is streamed into the Stream Processing FSM, split into control and data tokens, and issued in parallel to the 43 unique feature state machines. Generated feature and value pairs are collected by the Feature Gathering Network and forward on to the next pipeline stage.

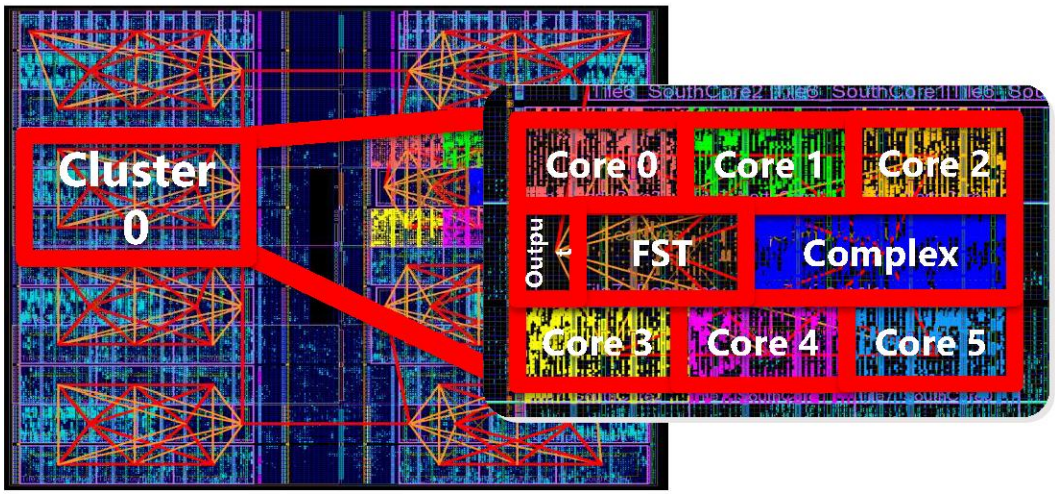

Figure 6: FFE Placed-and-Routed on FPGA.

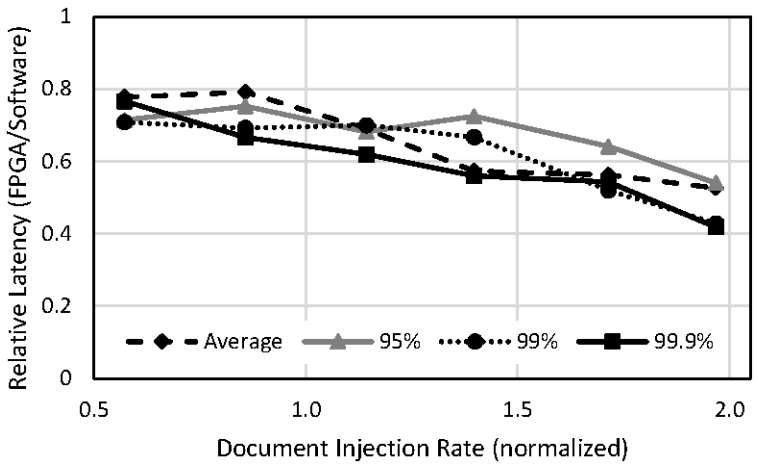

Figure 7: The FPGA ranker achieves lower average and tail latencies relative to software as the injection rate increases.

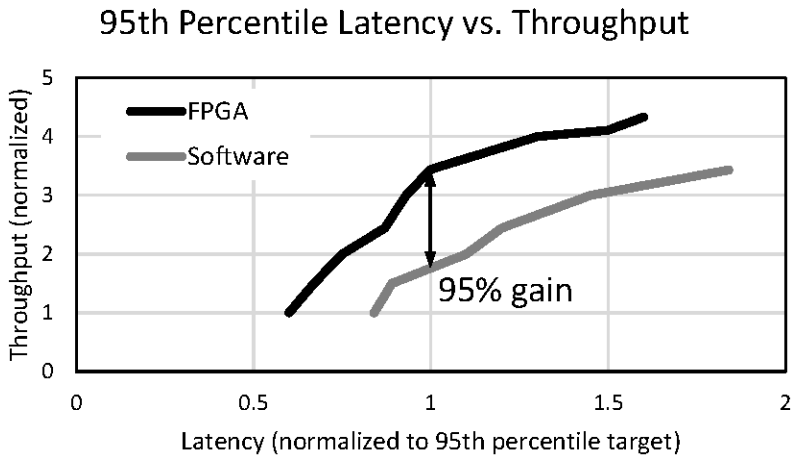

Figure 8: The points on the x-axis at 1.0 show the maximum sustained throughputs on both the FPGA and software while satisfying Bing's target for latency at the 95th percentile.