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

To advance datacenter capabilities beyond what commodity server designs can provide, the authors designed and built a composable, reconfigurable fabric to accelerate large-scale software services. They deployed the reconfigurable fabric in a bed of 1,632 servers and field-programmable gate arrays in a production datacenter and used it to increase the throughput of the ranking portion of the Bing Web search engine by nearly a factor of two.

Datacenter operators have relied on continuous enhancements in server performance and efficiency to make both new and improved services economically viable. Largely owing to power limitations, however, servers are improving in performance and efficiency at ever slower rates. Although specializing servers for specific workloads running at scale can provide efficiency gains, it is problematic for two reasons. First, homogeneity in the datacenter is highly desirable to reduce management issues and provide a consistent platform for applications. Second, datacenter services evolve rapidly, making nonprogrammable hardware features impractical. Thus, datacenter providers face a conundrum: they need continued improvements in performance and efficiency, but can't obtain those improvements from standard 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 computational accelerators in either datacenter infrastructure or client devices. One challenge traditionally associated with FPGAs is the need to fit the accelerated function into the available reconfigurable area on one chip. In theory, FPGAs can be virtualized using runtime reconfiguration to support more functions than could fit on a single device. However, current reconfiguration times for standard FPGAs are too slow to make this approach practical. Multiple FPGAs in a single server provide more area but cost more, consume more power, and are wasteful when unneeded. On the other hand, being restricted to a single FPGA per server restricts the workloads that might be accelerated and could make the associated gains too small to justify the cost.

This article describes a reconfigurable fabric called Catapult, which balances these competing concerns. The Catapult fabric is embedded into each half-rack of 48 servers in

Published by the IEEE Computer Society

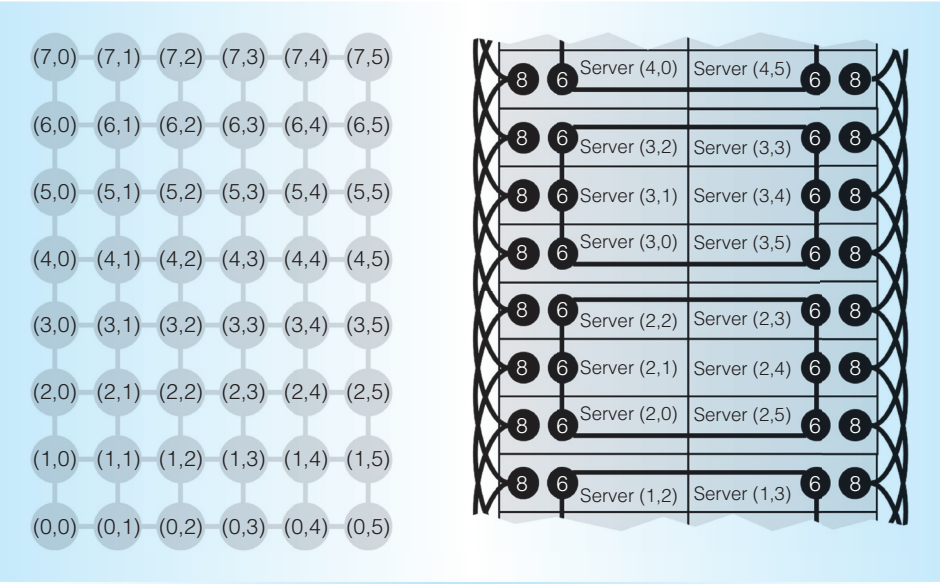

Figure 1. The logical mapping of the torus network, and the physical wiring on a pod of 2 × 24 servers. Each server has a single locally attached field-programmable gate array (FPGA).

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 6 × 8 2D torus, allowing services to allocate groups of FPGAs to provide the necessary reconfigurable area to implement the desired functionality.

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:

- preserve server homogeneity to avoid complex management of heterogeneous servers,
- scale to large workloads that might not fit into a single FPGA,
- avoid consuming too much power,
- avoid single points of failure,
- provide positive return on investment (ROI), and
- operate within the space and power confines of existing servers without hurting network performance or reliability.

These requirements guide the architectural choices we made throughout the Catapult system development.

Integration

There are several ways to integrate FPGAs into the datacenter. For example, they can be clustered into FPGA-only racks or installed into a few specialized servers per rack. Both of these methods violate datacenter homogeneity and are vulnerable to single points of failure, where the failure of a single FPGA-enabled rack or server can take down many conventional servers. In addition, communication to these specialized servers could potentially 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 PCI Express (PCIe), with the inter-server FPGAs tightly coupled via a secondary network.

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 shortcoming, we built a specialized network to facilitate FPGA-to-FPGA communication without degrading the existing Ethernet network. Figure 1 illustrates how the 48 FPGAs are organized at the rack level. Each server has a single locally attached

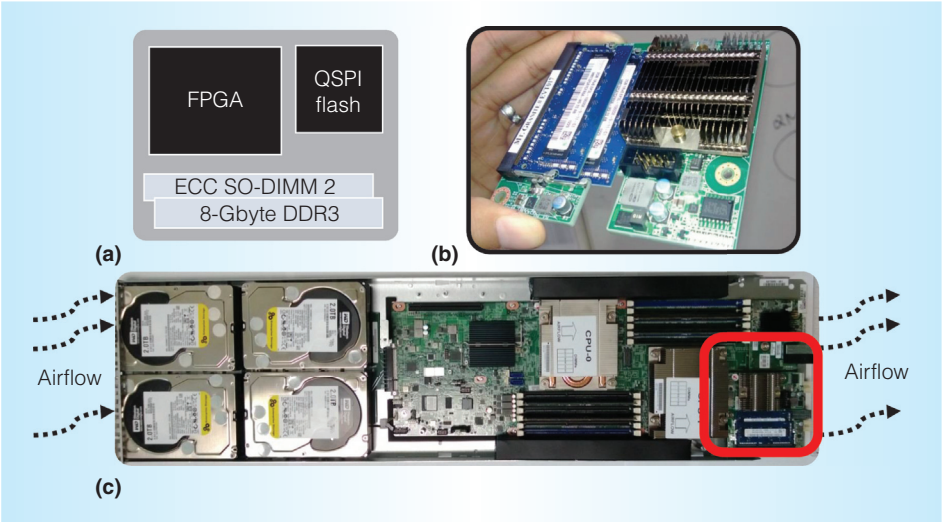

Figure 2. The FPGA board and the server into which it installs. (a) Block diagram of the FPGA board. (b) Picture of the manufactured board. (c) Diagram of the 1U, half-width server that hosts the FPGA board. The air flows from left to right, leaving the FPGA in the exhaust of both CPUs.

FPGA, and an inter-FPGA network lets services be logically mapped across multiple FPGAs. We selected a 2D, 6 × 8 torus topology, which balanced routability, resilience, and cabling complexity. Each inter-FPGA network link supports 20 Gbits per second of bidirectional bandwidth at submicrosecond latency, and comprises only passive copper cables, with no additional networking costs such as network interface cards or switches.

FPGA board design

Figure 2 shows the FPGA board and the server into which it installs.[1] The board incorporates an Altera Stratix V D5 FPGA[2] that has considerable reconfigurable logic, on-chip memory blocks, and digital signal processor units. The 8 Gbytes of DRAM comprises two dual-rank DDR3-1600 error-correcting code small outline DIMMs. The PCIe and inter-FPGA network traces are routed to a mezzanine connector on the bottom of the board that plugs directly into the motherboard. To avoid changes to the server itself, we custom designed the board to fit within a small 10 cm × 9 cm × 16 mm slot occupying the rear of a 1U (1.75" high), half-width server, which offered sufficient power and cooling for a 25-W PCIe peripheral device. In addition, the FPGA is placed downstream of both CPUs' exhausts, making it challenging to ensure that 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, remap services to recover from failures, and 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, including both increased capital costs and operating expenses, we set aggressive power and cost goals to achieve a positive ROI. Given the sensitivity of cost numbers on elements such as production servers, we can't give exact dollar figures; however, adding the FPGA card and network cost less than 30 percent in the total cost of

ownership, including a limit of 10 percent for total server power.

Datacenter deployment

To test this architecture on a critical production-scale datacenter service 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 two-socket EP motherboard, with dual 12-core Sandy Bridge CPUs, 64 Gbytes of DRAM, and two solid-state drives (SSDs) in addition to four hard-disk drives. The machines have a 10-Gbit network card connected to a 48-port top-of-rack switch, which in turn connects to a set of level-two switches. The daughter cards and cable assemblies were tested at manufacture and again at system integration. At deployment, we discovered that seven cards (0.4 percent) had a hardware failure and that one of the 3,264 links (0.03 percent) in the cable assemblies was defective. Since then, over several months of operation, we have seen no additional hardware failures.

Application case study

To drive the requirements of this new 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.

The 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 multiple 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 query processing and aggregates the final result. The TLA sends the same query (through midlevel aggregators) to many machines performing a selection service that finds documents (webpages) that match the query, and then narrows them down to a handful 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), which produces a score for that document query. The scores and document IDs are returned to the TLA, which sorts them, generates appropriate captions, and returns the results. These scores determine the final order of the webpages that the user sees.

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 SSD. 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 several other properties.

Many dynamic 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 (static, dynamic, and synthetic) 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 FFEs, and the machine-learned models on FPGAs. What remains in software is the SSD lookup, the hit vector computation, and a few software-computed features.

Software interface

To avoid sending unnecessary data, the ranking service compresses each document into a form that contains only the data

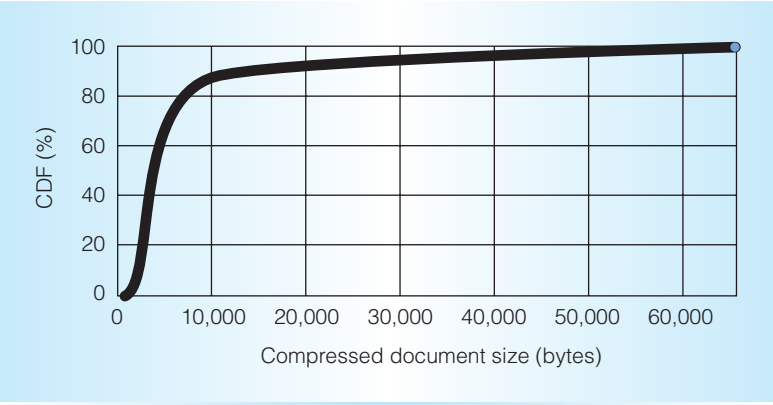

Figure 3. Cumulative distribution of compressed document sizes. Nearly all compressed documents are 64 Kbytes or less.

relevant for the given query. This encoded document-query request contains three sections: a header with basic request parameters, the set of software-computed features (static and dynamic), and the hit vector of query-match locations for each document's metastreams.

The header contains 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 either are not yet implemented on the FPGA or do not make sense to implement in hardware (such as static document features that are independent of the query and are stored alongside the document).

To save bandwidth, software-computed features and hit vector tuples are encoded in three different sizes using 2, 4, or 6 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.

Because of buffer limitations in the FPGA's direct memory access interface and the fact that the latency of feature extraction is proportional to the tuple count, we truncate compressed documents to 64 Kbytes. This limitation represents the only deviation of the accelerated ranker from the pure software implementation, but the effect on search relevance is small. Figure 3 shows a cumulative distributive function of all document sizes in an approximately 210,000-document document sample collected from real-world traces. As shown, nearly all of the compressed documents are less than 64 Kbytes (only 300 require truncation). On average, documents are 6.5 Kbytes, with the 99th percentile at 53 Kbytes.

For each request, the pipeline produces a single score (a 4-byte 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 direct memory access transfer moves the score, query ID, and performance counters back to the host.

Macropipeline

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

Queue manager and model reload

So far, the pipeline descriptions assumed a single set of features, FFEs, and machine-learned scorers. In practice, however, there are many different sets of features, free forms, and scorers. We call these different sets *models*. Different models are selected on the basis of each query and can vary for language (for example, Spanish, English, or Chinese) or query type. New experimental models are also frequently run in production.

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 the FPGA's local DRAM that 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 a timeout is reached, QM will switch to the next queue. When a new queue (that is, 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 a 20-Kbit capacity. Using the high-capacity DRAM configuration at DDR3-1333 speeds, model reload can take up to 250 μs. 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, although 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. An upper bound of 250 μs is reasonable so long as the number of queued documents per model is large enough to amortize the switching cost, which we believe will be the case in production.

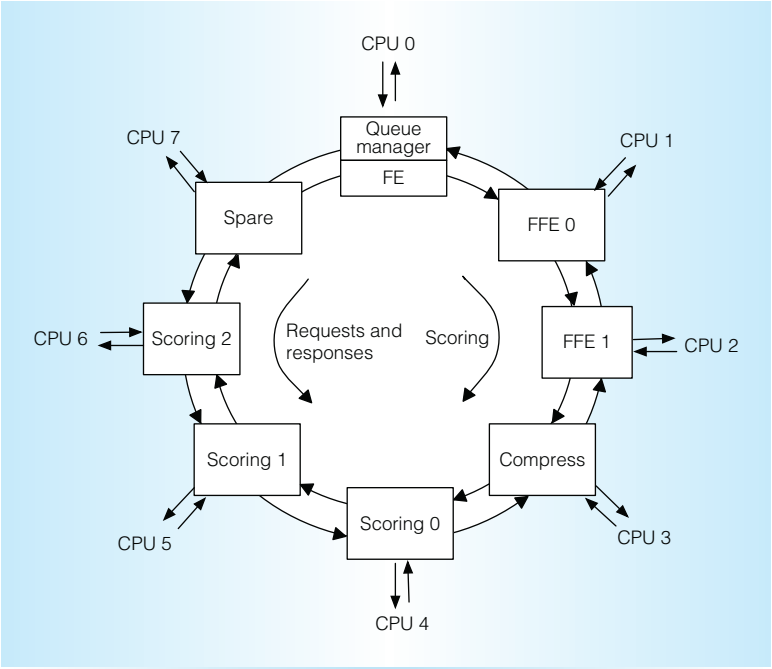

Figure 4. Mapping of ranking roles to FPGAs on the reconfigurable fabric. Data is sent from each server to the queue manager. It is then dispatched through the seven FPGA computation stages, and the results are sent back to the source server.

Feature extraction

The first stage of the scoring acceleration pipeline, feature extraction (FE), calculates numeric scores for various features on the basis of the query and document combination. Potentially thousands of unique features are calculated for each document, because each feature calculation produces a result for every stream in the request. Furthermore, some features also produce a result per query term. 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 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 nonzero 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 at a rate of one to two clock cycles per token. When a state machine finishes its computation, it emits one or more feature index and

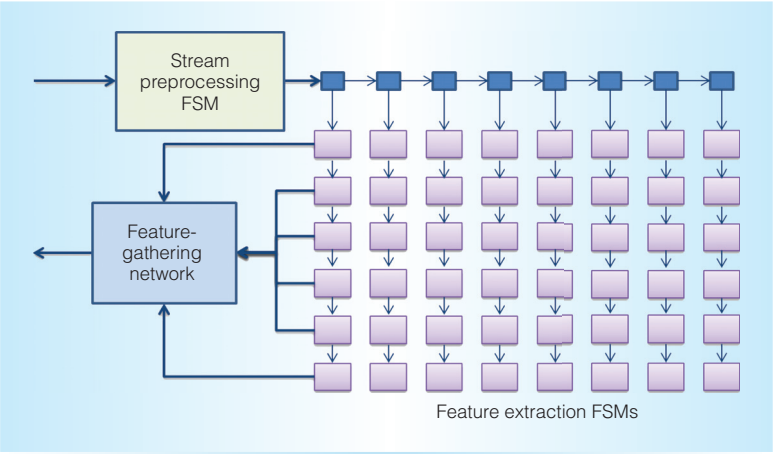

Figure 5. The first stage of the ranking pipeline. A compressed document is streamed into the stream processing state machine, split into control and data tokens, and issued in parallel to the 43 unique feature state machines. The feature-gathering network collects generated feature and value pairs and forwards them to the next pipeline stage.

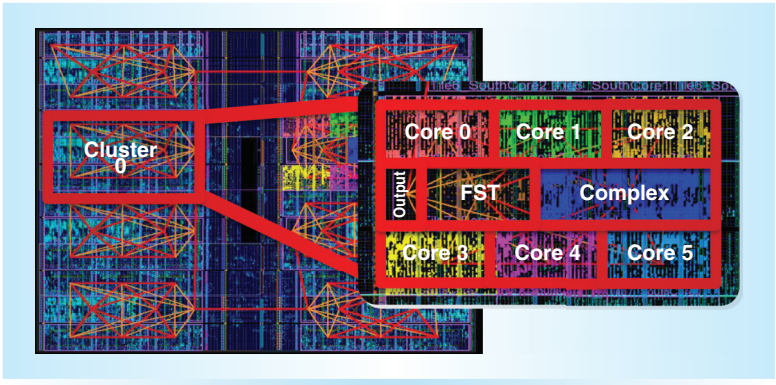

Figure 6. Free-form expressions (FFEs) placed and routed on an FPGA. Sixty cores fit on a single FPGA.

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.

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 simple (such as adding two features) to large and complex (with thousands of operations including conditional execution and complex floating-point operators such as *ln, pow,* and *fpdiv*). FFEs vary greatly across models, so it's impractical to synthesize customized datapaths for each expression.

One potential solution is to tile many off-the-shelf soft processor cores (such as Nios II), but these single-threaded cores are inefficient at processing thousands of threads with long-latency floating-point operations in the desired amount of time per macropipeline stage (8 μs). 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. The FFE microarchitecture is highly area efficient, letting us instantiate 60 cores on a single D5 FPGA.

The custom FFE processor has three key characteristics that make it capable of executing all of the expressions within the required deadline. First, each core supports four simultaneous threads that arbitrate for functional units on a cycle-by-cycle basis. When 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 expression with a simple feature read.

Because complex floating-point instructions consume a large amount of FPGA area, multiple cores (typically six) are clustered together to share a single complex block. Arbitration for the block is fair with round-robin priority. The complex block comprises

units for *ln, fpdiv, exp,* and *float-to-int.* The compiler translates *pow, intdiv,* and *mod* into multiple instructions to eliminate the need for expensive, dedicated units. In addition, the complex block contains the feature storage tile, which is double buffered, allowing one document to be loaded while another is processed.

Document scoring

The last stage of the pipeline is a machine-learned model evaluator that takes the features and FFEs as inputs and produces a 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.

Evaluation

We evaluated the Catapult fabric by deploying and measuring the Bing ranking engine described earlier 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 normalized target injection rate of 1.0 per server, the FPGA reduces the worst-case latency by 29 percent 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), whereas 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 percent 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: for equivalent ranking capacity, fewer servers can be purchased (in the target given earlier, by nearly a factor of two); or, new capabilities and features can be added to the software and/or hardware stack without exceeding the maximum allowed latency. Of course, a combination of the two is also possible.

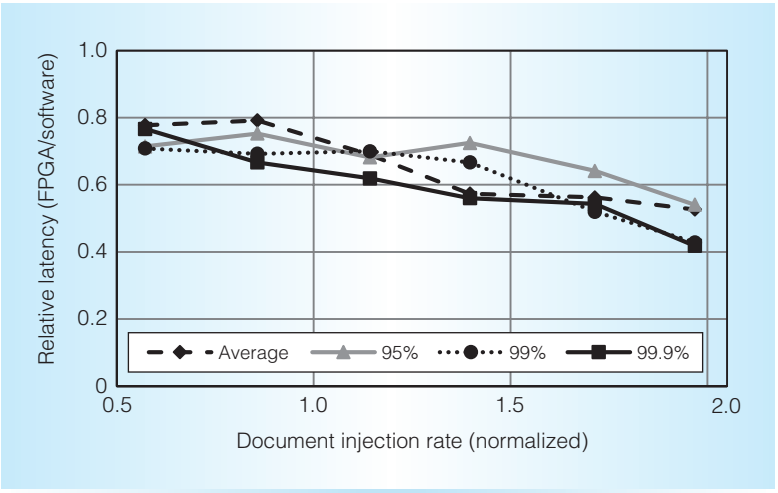

Figure 7. Relative latency of the FPGA-accelerated ranker. The FPGA ranker achieves lower average and tail latencies relative to software as the injection rate increases.

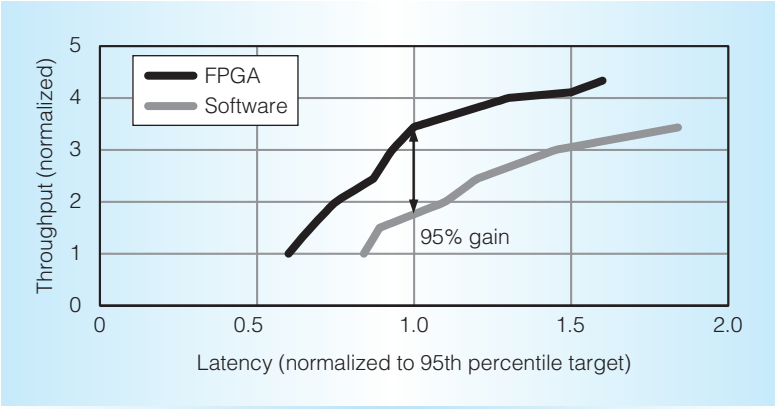

Figure 8. Achievable performance within a given latency bound. The points on the *x*-axis at 1.0 show the maximum sustained throughputs on both the FPGA and software while satisfying Bing's target for latency at the 95th percentile.

For over a decade, FPGAs have shown promise for accelerating many computational tasks (see the "Related Work in FPGA-Based Computation" sidebar), but they have

Related Work in FPGA-Based Computation

Many other groups have worked on incorporating field-programmable gate arrays (FPGAs) into CPUs to accelerate workloads in large-scale systems.

One challenge in 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[1,2] or Intel's Front Side Bus[3] and QuickPath Interconnect (QPI).[4] Although integrating the FPGA directly onto the processor bus would reduce direct memory access 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 server's overall utility for applications that can't use the FPGA.

IBM's Coherence Attach Processor Interface[5] and Convey's Hybrid-Core Memory Interconnect[6] enable advanced memory sharing with coherence between the FPGA and CPU. Because our ranking application requires only simple memory sharing, these mechanisms are not yet necessary but could 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,[6] Maxeler MPC series,[7] BeeCube BEE4,[8] and SRC MAPstation[9] are all examples of commercial FPGA acceleration appliances. Although 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 more difficult to guarantee. Finally, the appliance creates a single failure point 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,[10] Novo-G,[11] and QP.[12] These systems integrate the FPGA with the CPU, but the FPGA-to-FPGA communication must be routed through the CPU. Maxwell is the most similar to our design, as it directly connects FPGAs in a 2D torus using InfiniBand cables, although the FPGAs do not implement routing logic.[13] These systems are targeted to high-performance computing 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 high-performance computing 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,[14,15] compression and decompression,[16,17] *k*-means clustering,[18,19] and Web search. Researchers have used FPGAs to accelerate search,[20,21] but they 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 free-form expression (FFE) stage is a soft processor core, one of many available for FPGAs, including MicroBlaze[22] and Nios II.[23] 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,[24] CoRAM,[25] BORPH,[26] and LEAP.[27]

References

1. D. Slogsnat et al., "An Open-Source HyperTransport Core," *ACM Trans. Reconfigurable Technology and Systems*, vol. 1, no. 3, 2008, article 14.
2. *DRC Accelium Coprocessors Datasheet*, white paper, DRC, 2014.
3. L. Ling et al., "High-Performance, Energy-Efficient Platforms using In-Socket FPGA Accelerators," *Proc. ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays*, 2009, pp. 261–264.

not yet become mainstream in modern datacenters. 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. 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 because both have hundreds to thousands of arithmetic units available on each chip. We decided not to incorporate GPUs because the current

4. *An Introduction to the Intel Quickpath Interconnect*, Intel, Jan. 2009.
5. J. Stuecheli, "Next Generation POWER Microprocessor," Hot Chips 2013; www.hotchips.org/wp-content/uploads/hc_archives/hc25/HC25.20-Processors1-epub/HC25.26.210-POWER-Studecheli-IBM.pdf.
6. *The Convey HC-2 Computer*, white paper, Convey Computer, 2012; www.conveycomputer.com/files/4113/5394/7097/Convey_HC-2_Architectual_Overview.pdf.
7. O. Pell and O. Mencer, "Surviving the End of Frequency Scaling with Reconfigurable Dataflow Computing," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 4, 2011, pp. 60–65.
8. BEE4 Hardware Platform, BEEcube, 2011.
9. *MAPstation Systems*, white paper, SRC, 2014.
10. *Cray XD1 Datasheet*, Cray, 2004.
11. A. George, H. Lam, and G. Stitt, "Novo-G: At the Forefront of Scalable Reconfigurable Supercomputing," *Computing in Science Eng.*, vol. 13, no. 1, 2011, pp. 82–86.
12. M. Showerman et al., "QP: A Heterogeneous Multi-Accelerator Cluster," *Proc. 10th LCI Int'l Conf. High-Performance Clustered Computing*, 2009.
13. R. Baxter et al., "Maxwell—A 64 FPGA Supercomputer," *Eng. Letters*, vol. 16, no. 3, 2008, pp. 426–433.
14. M. Lavasani, H. Angepat, and D. Chiou, "An FPGA-Based In-line Accelerator for Memcached," *IEEE Computer Architecture Letters*, vol. 13, no. 2, 2013, pp. 57–60.
15. M. Blott and K. Vissers, "Dataflow Architectures for 10Gbps Line-Rate Key-Value Stores," Hot Chips 2013, 2013; www.hotchips.org/wp-content/uploads/hc_archives/hc25/HC25.50-FPGA-epub/HC25.27.510-Dataflow-Blott-Vissers-Xlinix-final_no_animation.pdf.
16. *IBM PureData System for Analytics N2001*, white paper, IBM, 2013.
17. A. Martin, D. Jamsek, and K. Agarawal, "FPGA-Based Application Acceleration: Case Study with GZIP Compression/Decompression Streaming Engine," *Proc. Int'l Conf. Computer-Aided Design*, 2013.
18. M. Estlick et al., "Algorithmic Transformations in the Implementation of *K*-Means Clustering on Reconfigurable Hardware," *Proc. ACM/SIGDA 9th Int'l Symp. Field Programmable Gate Arrays*, 2001, pp. 103–110.
19. H.M. Hussain et al., "Highly Parameterized *K*-means Clustering on FPGAs: Comparative Results with GPPs and GPUs," *Proc. Int'l Conf. Reconfigurable Computing and FPGAs*, 2011, pp. 475–480.
20. J. Yan et al., "Efficient Query Processing for Web Search Engine with FPGAs," *Proc. IEEE 20th Int'l Symp. Field Programmable Custom Computing Machines*, 2012, pp. 97–100.
21. W. Vanderbauwhede, L. Azzopardi, and M. Moadeli, "FPGA Accelerated Information Retrieval: High-Efficiency Document Filtering," *Proc. Int'l Conf. Field Programmable Logic and Applications*, 2009, pp. 417–422.
22. *MicroBlaze Processor Reference Guide*, 14th ed., Xilinx, 2012.
23. *Nios II Processor Reference Handbook*, 13th ed., Altera, 2014.
24. R. Kirchgessner et al., "VirtualRC: A Virtual FPGA Platform for Applications and Tools Portability," *Proc. ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays*, 2012, pp. 205–208.
25. E.S. Chung, J.C. Hoe, and K. Mai, "CoRAM: An In-Fabric Memory Architecture for FPGA-Based Computing," *Proc. 19th ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays*, 2011, pp. 97–106.
26. H.K.-H. So and R. Brodersen, "A Unified Hardware/Software Runtime Environment for FPGA-Based Reconfigurable Computers Using BORPH," *ACM Trans. Embedded Computing Systems*, vol. 7, no. 2, 2008.
27. M. Adler et al., "Leap Scratchpads: Automatic Memory and Cache Management for Reconfigurable Logic," *Proc. 19th ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays*, 2011, pp. 25–28.

power requirements of high-end GPUs are too high for conventional datacenter servers, and because it was unclear whether some latency-sensitive ranking stages (such as feature extraction) would map well to GPUs.

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 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 percent at comparable latency to a software-only solution. The added FPGA

computing boards increased power consumption by only 10 percent and did not exceed our 30 percent limit in an individual server's total cost of ownership, 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 long-term challenge is programmability. FPGA development still requires extensive hand-coding in Register Transfer Level 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 10 to 15 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.

MICRO

Acknowledgments

Many people across many organizations contributed to this system's construction, and although 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 also thank 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. K. Vaid, M. Shaw, and M. Drake, *How Microsoft Designs its Cloud-Scale Servers*, white paper, Microsoft, 2014.
2. *Stratix V Device Handbook*, 14th ed., Altera, 2014.

Andrew Putnam is a principal research hardware development engineer in Microsoft Research NExT. His research interests include reconfigurable computing, future datacenter design, and computer architecture. Putnam has a PhD in computer science and engineering from the University of Washington. Contact him at anputnam@microsoft.com.

Adrian M. Caulfield is a senior research hardware development engineer in Microsoft Research NExT. His research interests include computer architecture and reconfigurable computing. Caulfield has a PhD in computer engineering from the University of California, San Diego. Contact him at acaulfie@microsoft.com.

Eric S. Chung is a researcher in Microsoft Research NExT. His research focuses on the intersection of computer architecture and reconfigurable computing with FPGAs. Chung has a PhD in electrical and computer engineering from Carnegie Mellon University. Contact him at erchung@microsoft.com.

Derek Chiou is a principal architect at Bing and an associate professor (on leave) at the University of Texas at Austin. His research interests include FPGA-based acceleration, high-level descriptions of hardware, and high-performance computer simulation techniques. Chiou has a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology. Contact him at dechiou@microsoft.com.

Kypros Constantinides is a senior hardware development engineer at Amazon Web Services. His research interests include scalable

server platform architectures for datacenters and heterogeneous compute architectures. Constantinides has a PhD in computer science and engineering from the University of Michigan. He performed the research for this article while at Microsoft Research. Contact him at kypros@amazon.com.

John Demme is an associate research scientist at Columbia University. His research interests include programming paradigms for spatial computing and data-intensive computing. Demme has a PhD in computer science from Columbia University. Contact him at jdd@cs.columbia.edu.

Hadi Esmaeilzadeh is the Allchin Family Early Career Professor of Computer Science at the Georgia Institute of Technology. He founded and directs the Alternative Computing Technologies Lab. His research interests include developing new technologies and cross-stack solutions to build next-generation computer systems for emerging applications. Esmaeilzadeh has a PhD in computer science from the University of Washington. Contact him at hadi@cc.gatech.edu.

Jeremy Fowers is a research engineer in the Catapult team at Microsoft Research. His research focuses on creating FPGA accelerators and frameworks to better use custom FPGA logic and hard intellectual-property blocks. Fowers has a PhD in electrical engineering from the University of Florida. Contact him at jfowers@microsoft.com.

Gopi Prashanth Gopal is a senior engineering manager at Amazon. His research interests include computer vision, virtual reality, and machine learning. Prashanth Gopal has an MS in computational engineering from Mississippi State University. He performed the research for this article while at Bing. Contact him at gopiprashanth@gmail.com.

Jan Gray is a consultant, computer architect, and software architect. His research focuses on the design of FPGA-optimized soft processor arrays and computation accelerators. Gray has a BMath in computer science and electrical engineering from the University of Waterloo, Canada. Contact him at jsgray@acm.org.

Michael Haselman is a software engineer at Microsoft Applied Sciences Group (Bing). His research interests include FPGAs and computation accelerators. Haselman has a PhD in electrical engineering from the University of Washington. Contact him at mikehase@microsoft.com.

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. His research focuses on FPGAs and reconfigurable computing. Hauck has a PhD in computer science and engineering from the University of Washington. Contact him at hauck@uw.edu.

Stephen Heil is a principal program manager at Microsoft Research. His research focuses on the development of custom programmable hardware accelerators for use in Microsoft datacenters. Heil has a BS in electrical engineering technology and computer science from the College of New Jersey (formerly Trenton State College). Contact him at stephen.heil@microsoft.com.

Amir Hormati is a senior software engineer at Google. His research focuses on large-scale data analysis and storage platforms. Hormati has a PhD in computer science and engineering from the University of Michigan. He performed the research for this article while at Microsoft Research. Contact him at hormati@gmail.com.

Joo-Young Kim is a senior research hardware development engineer at Microsoft Research. His research focuses on high-performance accelerator design for datacenter workloads. Kim has a PhD in electrical engineering from the Korea Advanced Institute of Science and Technology. Contact him at jooyoung@microsoft.com.

Sitaram Lanka is a group engineering manager for Search Platform at Bing. His research interests include distributed systems, large-scale systems, and fault tolerance. Lanka

has a PhD in computer science from the University of Pennsylvania. Contact him at slanka@microsoft.com.

James Larus is a professor and the dean of the School of Computer and Communication Sciences at École Polytechnique Fédérale de Lausanne. His research interests include programming languages, compilers, and computer architecture. Larus has a PhD in computer science from the University of California, Berkeley. He is an ACM Fellow. Contact him at james.larus@epfl.ch.

Eric Peterson is a principal mechanical architect at Microsoft Research. His research interests include new datacenter energy systems and system integration. Peterson has a BS in mechanical engineering from Texas A&M University. Contact him at eric.peterson@microsoft.com.

Simon Pope is a program manager for Microsoft. His research focuses on the nexus of computing and psychology. Pope has a master's in cognitive science from the University of New South Wales. Contact him at simon.pope@microsoft.com.

Aaron Smith is a principal research software development engineer at Microsoft Research. His research focuses on the development of advanced optimizing compilers and microprocessors. Smith has a PhD in computer science from the University of Texas at Austin. Contact him at aaron.smith@microsoft.com.

Jason Thong is a hardware design engineer with Microsoft Research. His research interests include hardware acceleration, heterogeneous computing, and computer-aided design. Thong has a PhD in computer engineering from McMaster University. Contact him at a-jathon@microsoft.com.

Phillip Yi Xiao is a senior software engineer at Bing. His research focuses on distributed computing architecture and machine learning. Xiao has a master's in computer science and engineering from JiaoTong University. Contact him at phxiao@microsoft.com.

Doug Burger directed the Hardware, Devices, and Experiences Group at Microsoft Research NExT. His research interests include cloud architecture, efficient silicon architectures, and machine-learned personal services. Burger has a PhD in computer science from the University of Wisconsin–Madison. He is an IEEE and ACM Fellow. Contact him at dburger@microsoft.com.