Appears in The Exascale Evaluation and Research Techniques Workshop (EXERT 2011) at ASPLOS 2011

Datacenter-Scale Network Research on FPGAs

Zhangxi Tan
Computer Science Division
UC Berkeley, CA
xtan@eecs.berkeley.edu

Krste Asanović
Computer Science Division
UC Berkeley, CA
krste@eecs.berkeley.edu

David Patterson
Computer Science Division
UC Berkeley, CA
pattrsn@eecs.berkeley.edu

ABSTRACT
We describe an FPGA-based datacenter network simulator to allow researchers to rapidly experiment with $O(10,000)$ node datacenter network architectures. We configure the FPGA hardware to implement abstract models of key datacenter building blocks including servers and all levels of switches. We discuss design and implementation issues of our FPGA models and show that it is practical to prototype and scale the testbed with a few low-cost FPGA boards.

1. INTRODUCTION
Massive warehouse-scale computers (WSCs) [5] are the foundation of widely used Internet services; e.g., search, social networking, email, video sharing, and online shopping. The tremendous success of these services has led to the rapid growth of datacenters to keep up with the increasing demand. Recent advances such as modularized container-based datacenter construction and server virtualization have allowed modern datacenters to scale up from 10,000 servers to 100,000 servers or more [11].

At this extreme scale, network infrastructure has become one of the most critical data center components [8, 20] for several reasons:

1. Current networks are extremely complex, and are difficult to scale out to larger configurations without complete redesign.

2. Existing networks have many different failure modes. Occasionally, correlated failures are found in replicated million-dollar units.

3. Networking infrastructure has a significant impact on server utilization, which is an important factor in datacenter power consumption and cost-effectiveness.

4. Network infrastructure is crucial for supporting data intensive Map-Reduce jobs.

5. Network infrastructure accounts for 18% of the monthly datacenter costs, which is the third largest contributing factor [8]. In addition, existing large commercial switches and routers command high margins and charge a great deal for features that are rarely used in datacenter.

As a result, datacenter network architecture has become an active area of research, often focusing on new switch designs [9, 10, 16, 20]. However, warehouse-scale network research is very difficult to perform. Previous work [17] notes that most of these new proposals are based on observations of existing datacenter infrastructure and applications, and lack a sound methodology to evaluate new designs. Moreover, most proposed designs have only been tested with a very small testbed running unrealistic microbenchmarks, often built using off-the-shelf devices [7] that have limitations when exploring proposed new features. The behavior observed by running a test workload over a few hundred nodes bears little relationship to the behavior of production runs completed over thousands or tens of thousands of nodes. The topology and switches used for small test clusters are very different from those in a real environment. Dedicating tens of thousands of nodes to network research is impractical even for large companies like Amazon and Microsoft, let alone academic researchers.

For systems research, one possible solution is to use cloud computing, such as Amazon EC2, when extreme scale is needed. But EC2 nodes are already fully provisioned with networking, and building an overlay network and testing at scale in EC2 is only adequate for some areas of network research, such as network configuration and management. Cloud computing cannot be readily applied to hardware network switch studies, as the network switches have to be emulated at high fidelity using software models.

Unfortunately, simulating target devices using software is prohibitively slow [19], given the growing complexity in target designs. To mitigate this software simulation gap, many techniques have been proposed to reduce simulation time, such as statistical sampling and parallel simulation with relaxed synchronization. These techniques assume the workload is static and independent of target architecture, but datacenter networks exhibit highly dynamic target-dependent behavior, as they are tightly coupled with computation servers running very adaptive software networking stacks.

To address the above issues, we propose using Field Programmable Gate Arrays (FPGAs) to build a reconfigurable simulation testbed at the scale of O(10,000) nodes. Each node in the testbed is capable of running real datacenter applications on a full operating system. In addition, our network elements are heavily instrumented. This research testbed will allow us to record the same behaviors administrators observe when deploying equivalently scaled datacenter software. We build our testbed on top of a cost-effective FPGA-based full-system manycore simulator, RAMP Gold [18]. Instead of mapping the real target hardware directly, we build several abstract models with runtime configurable parameters of key datacenter components and compose them together in FPGAs. In this paper, we show how to construct a 10,000-node system model from several low-cost FPGA boards connected with multi-gigabit serial links. The prototype testbed has been successfully applied to evaluating a novel network proposal based on circuit-switching technology [21] running application kernels taken from the Microsoft Dryad Terasort program.

2. DATACENTER NETWORK INFRASTRUCTURE BACKGROUND

Datacenters use a hierarchy of local-area networks (LAN) and off-the-shelf switches. Figure 1 shows a typical datacenter network arranged in a Clos topology with three networking layers. At the bottom layer, each rack typically holds ~20–40 servers, each singly connected to a commodity *Top-of-Rack (ToR)* switch with a 1 Gbps link. These ToR switches usually offer two to eight uplinks, which leave the rack to connect up to several *array switches* to provide redundancy and bandwidth. At the top of the hierarchy, *datacenter switches* carry traffic between array switches usually using 10 Gbps links. All links use Ethernet as the physical-layer protocol, with either copper or fiber cabling depending on the connection distance.

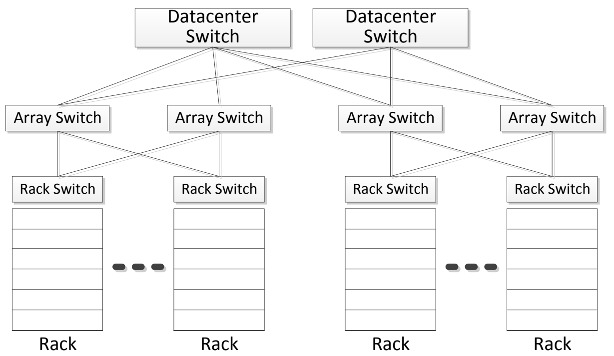

Figure 1: A typical datacenter network architecture.

As we move up the hierarchy, one of the most challenging problems is that the bandwidth "over-subscription" ratio (i.e. bandwidth entering from below versus bandwidth to the level above) gets worse rapidly. This imbalance is due to the cost of switch bandwidth, which grows quadratically in the number of switch ports. The resulting limited datacenter bisection bandwidth significantly affects the design of software and the placement of services and data, hence the current active interest in improving network switch designs.

3. FPGA-BASED DATACENTER-SCALE EMULATION

FPGA Architecture Model Execution (FAME) [19] has become a promising vehicle for architectural investigation of parallel computer systems. Groups in both academia and industry have built various types of FAME simulators, which can be classified in five levels that are analogous to different RAID levels [19]. Higher FAME levels lower the simulator cost and improve performance over the lower levels, while moving further away from the concrete RTL design of the simulation target. As pointed out in our previous work [17], datacenter-scale simulation requires simulators to handle hundreds of thousands of concurrent events synchronized to within 50 ns of simulated time across $O(10,000)$ server nodes. We propose building a FAME-7 simulator based on modules that fit in low-cost single-FPGA boards, using the following three key techniques to improve efficiency [19]:

1. *Abstracted Models* A full-fledged implementation of any datacenter component requires considerable design effort and hardware resources. Instead, we employ high-level abstract models that greatly reduce these requirements and build a simplified version of each target component, capturing important features but simplifying or removing features that are rarely used in practice. We also separate functional models from timing models to simplify parameterization of target timing.

2. *Decoupled Design* A function computed within a single target clock cycle can be implemented with a variable number of FPGA host clock cycles. For example, a simple ring network on the FPGA can model an expensive multiport switch crossbar in datacenter switches.

3. *Host Multithreading* Instead of replicating hardware models to simulate multiple instances in the target, we use multiple model threads running in a single host hardware model to simulate different target instances. Multithreading significantly improves FPGA resource utilization and hides simulation latencies, such as those from host DRAM access and timing-model synchronization across different FPGAs.

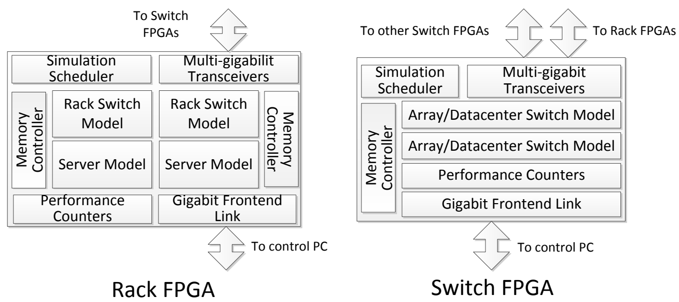

Figure 2: FPGA Simulator Architecture.

Figure 2 shows the high-level simulator architecture for the typical target datacenter configuration presented in Figure 1. We map all server models along with the ToR switch models into *Rack FPGAs*, and array and datacenter switch models to separate *Switch FPGAs*. This partition enables a more modularized model design that eases experimentation with new array and datacenter switch designs. It also makes it easy to scale up the size of the emulated datacenter. To further simplify switch model design, we keep any switch model within a single FPGA. Following the physical topology of the target system, we connect Rack FPGAs to Switch FPGAs through several time-shared multi-gigabit serial transceivers using low-cost copper cables, such as standard SATA cables. Each FPGA has its own simulation scheduler that synchronizes with adjacent FPGAs over the serial links at a very fine granularity to satisfy simulation accuracy requirements. For example, a 10 Gbps switch with a minimum flit size of 64 bytes requires a maximum synchronization interval of 51.2 ns. We reduce host communication latency by using our own protocol over the serial links, achieving FPGA-FPGA communication latencies of around 20 FPGA logic cycles, which is roughly the latency for a host DRAM access on the FPGA. In addition, the host-multithreaded design further helps to hide the simulator communication latency, removing model synchronization latency as a simulator performance bottleneck.

Every model in the design has numerous hardware performance counters that periodically send performance statistics to a control workstation over a separate gigabit Ethernet link to avoid interrupting the simulation. Rack FPGAs support more memory controllers than switch FPGAs to provide the larger memory capacity and bandwidth required to simulate software running on the server models. We statically partition the host DRAM resources on each FPGA between different server models and the ToR switch models.

We select multi-gigabit serial transceivers as the only inter-FPGA connection instead of the high-speed parallel LVDS links often seen on multi-FPGA boards to make the design simpler and more modular. Specifically, parallel LVDS links increase design complexity. To ensure reliable transmission, designs require complicated dynamic calibration and special eye-opening monitoring circuit on groups of I/O signals. In addition, designs with LVDS links are less portable because of varying I/O layouts on different boards, making connections between Rack FPGAs and Switch FPGAs less flexible. Moreover, LVDS links increase both PCB board and FPGA cost because they require more FPGA I/O pins and link wires for a given link bandwidth. Finally, we found that the multi-gigabit serial transceivers provide enough bandwidth between FPGAs considering our overall simulation slowdown of between 250× and 1000× of real time. For example, 2.5 Gbps transceivers are common on three-year-old Xilinx Virtex 5 FPGAs. The bandwidth of a single transceiver translates to 500 Gbps to 2.5 Tbps in the target, which far exceeds the bandwidth between a few racks and several array switches. Moreover, recent FPGAs have significantly enhanced serial transceiver performance, supporting up to 28 Gbps bandwidth [4] in the 28 nm generation.

3.1 Server Models

We build the server models on top of RAMP Gold, which is an open-source cycle-accurate full-system FAME-7 architecture simulator. RAMP Gold supports the full 32-bit SPARC v8 ISA in hardware, including floating-point instructions and precise exceptions. It also models sufficient hardware to run an operating system, including MMUs, timers, and interrupt controllers. Currently, we can boot the Linux 2.6.21 kernel and a manycore research OS [14]. We map one server to one hardware thread in RAMP Gold. One 64-thread RAMP Gold hardware pipeline simulates up to two 32-server datacenter racks. Each simulated server uses a simplified fixed-CPI timing model. A more detailed timing model could be implemented, but it would reduce simulation scale as each server model would require additional host hardware resources.

The simulation performance of a 64-thread configuration is two orders of magnitude faster than state-of-the-art software simulators for parallel processors, or a slowdown of ∼1000× compared to real hardware. The server models are currently the simulation bottleneck for the whole system, but performance could be improved by reducing the number of threads per hardware model as discussed in [18]. When packing more pipelines onto a single FPGA to simulate more racks, another potential concern is the host memory controller bandwidth. However, our earlier analysis shows this is not a problem for RAMP Gold [18]. A single pipeline only consumes a maximum of 15% of the total bandwidth of a single host DRAM channel when running real-world applications on a research OS

3.2 Switch Models

There are two broad categories of datacenter switches: connectionless packet switching, also known as datagram switching, and connection-oriented virtual circuit switching. In the first case, each packet includes complete routing information, and is routed by network devices individually. The second case requires a pre-allocated virtual circuit path before transferring any packet. To demonstrate the flexibility of our approach, we build FAME-7 models for both types of switches.

The real challenges for modeling the packet switches used in existing production datacenters arise from design complexity and proprietary architecture specifications. To work around these barriers, we build abstract models by simplifying features that are seldom used in a datacenter. Here are the abstractions we employed and the rationale behind our choice:

1. Ignore Ethernet QoS related features (e.g. support of IEEE 802.1p class of service (CoS)): Although QoS features are available on almost every switch today, datacenters only utilize switches for basic connectivity without turning on QoS features.
2. Use simplified source routing: Many packet switches use a large ternary CAM to hold flow tables and look up the destination address for each packet. When an unknown MAC address is seen, the forwarding engine sends an interrupt to a slow-path control processor that updates the table using software. Many switches [2, 3] already support flow tables that have at least 32K entries. Given the total number of machines in datacenters, the slow-path flow-table update is rarely executed, making the TCAM lookup time constant in practice. Besides, datacenter topologies do not change frequently, and routes can be pre-configured statically. We use source routing to simplify modeling of packet routing, and we note that source routing is actually a component of many datacenter-switch research proposals. To emulate more complicated flow table operations, we could implement d-left hash tables [15] using host DRAM. Recent datacenter switches that implement large flow tables, such as the Cisco Nexus 5000, use similar techniques instead of TCAMs.

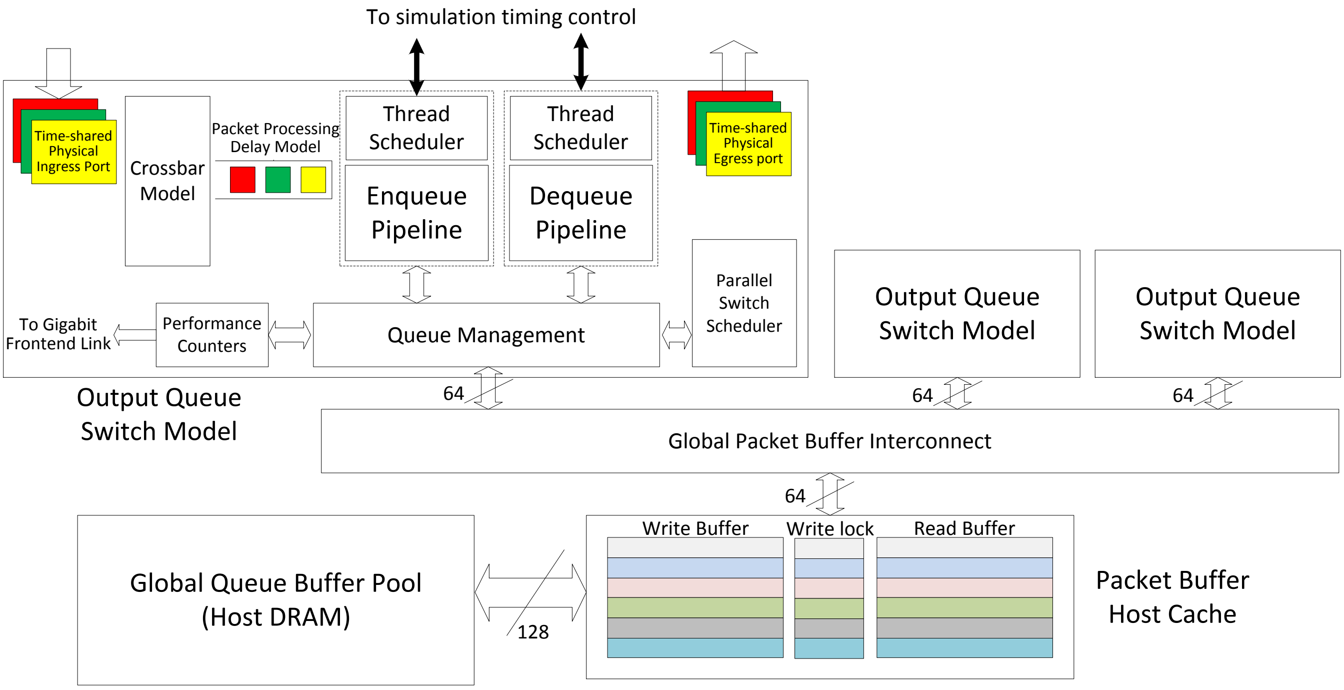

Figure 3: FAME model for virtual output queue switches.

3. Abstract packet processors: Commercial datacenter switches include many pipelined packet processors that handle different tasks such as MAC address learning, VLAN membership, and so on. The processing time of each stage is relatively constant regardless of packet size, and the time can be as short as a few hundred nanoseconds [6] to a few microseconds [2]. We simply employ FIFOs with runtime-configurable delays to model packet processing.

Although commercial switch implementation details are generally not publicly available, the fundamentals of these switch architectures are well known. Examples include the architecture of a virtual output queue switch and common scheduling algorithms. We build our abstracted model focusing on these central well-known architectural features, and allow other parts that are unclear or of special interest to researchers (e.g. packet buffer layout) to be configurable during simulation.

Figure 3 shows the architecture of our abstracted simulation model for output-queue switches, such as the Fulcrum FocalPoint FM4000 [6]. One of the biggest differences between existing commercial packet switches is the packet buffer size. For instance, the Force 10 S60 switch has 1280 MB of packet buffering, the Arista Network's 7048 switch has 768 MB, and the Cisco Systems' 4948-10GE switch has 16 MB. According to our conversations with datacenter networking researchers in industry, switch buffer management and configurations have also become an active area for packet switching researchers. To provide maximum flexibility for simulating a wide range of switch buffer sizes and to keep the overall design simple, we place the physical storage of simulated switch buffers in the host DRAM and all virtual queue pointers in BRAMs on the FPGA.

To make efficient use of host DRAM burst accesses, we designed a shared host cache connected by a ring-like interconnect to all switch models using the same host DRAM channel. The host cache is composed of two simple buffers, one for write and one for read, partitioned equally among all physical ports. Due to the limited size of on-chip FPGA BRAM, the write and read buffers only hold 64 bytes for every physical port, which is the minimum flit size for many packet switches. In addition, the write and read buffers for each port have a write-lock to ensure they are kept coherent.

Inside each switch model, a key component is a queue management model responsible for all virtual queue pointer operations. It also keeps track of queue status and performs packet drops when necessary. The length of every simulated virtual queue can be configured dynamically without requring another FPGA CAD flow run before a simulation starts. We select these configurable parameters according to a Broadcom switch design [13]. Along with this module, a performance-counter module, implemented with a collection of BRAMs and LUTRAMs, maintains all statistics for every virtual queue. The performance counter module reports all of its content periodically to a remote PC through the gigabit frontend link, with which we can construct queue length dynamics offline for every virtual queue running any workload. To send unicast statistics every 6.4 μs in target time, a 10 Gbps 32-port output-queue switch model demands a bandwidth of approximately 40 Mbps on the frontend link.

Each model has an independent 3-stage enqueue pipeline and a 4-stage dequeue pipeline that are controlled and synchronized by global simulation timing control logic. Ideally, the two pipelines send commands to the queue management

FAME Model	Registers		LUTs		BRAMs		Lines of Code
64-server model	9,981	(14%)	6,928	(10%)	54	(18%)	35,000
Circuit-swiched model	859	(1%)	1,498	(2%)	28	(9%)	2,550
Packet-switched model	814	(1%)	1,260	(2%)	24	(8%)	2,925

Table 1: FPGA resource usage and lines of code for different FAME models.

model through simple FIFOs in order to simplify and decouple the control logic design. However, this appears to be a significant area and performance overhead on FPGAs, consuming a large amount of distributed LUTRAM and making routing very hard. Instead, we implement two independent static thread schedulers for the two pipelines and replay commands that were deferred due to dependencies.

To guarantee good simulation performance, the switch scheduler model processes scheduling decisions for multiple virtual queues in every host FPGA clock cycle. To further improve performance, given the hundreds or thousands of virtual queues existing in our simulated switches, the parallel scheduler model only processes active events that happen between two scheduling quanta instead of naively scanning all virtual queues. Overall, the simulation performance of a single switch model has a slowdown of 150× compared to real hardware. This is four times faster than a software single-threaded network simulator used at Google [1], which does not simulate packet payloads or support full software stack scaling to 10,000 nodes as does our system.

Our circuit-switching models are based on a recent proposal for container-based datacenters [21], which has two levels of switch. In contrast to the complexity of the packet-switched approach, the proposed circuit-switched model is simple enough to be directly implemented on entry-level FPGAs. By employing a host-multithreaded architecture that time-multiplexes packets from multiple target switch ports, we can model several 16-port rack-level switches along with a 128-port 10-Gbps high-radix array/datacenter switch with full architecture details on a single FPGA.

4. IMPLEMENTATION

We code all models in SystemVerilog and map them to a three-year old Xilinx Virtex-5 LX110T-1 target device. All models run well at 100 MHz on FPGA hardware. To verify the correctness of our FPGA model, we use SystemVerilog assertions and code coverage groups extensively. The workload we run are traffic patterns sampled from the Microsoft Dryad Terasort application. Table 1 lists FPGA resource consumption as well as design effort, measured in number of lines of code. To validate our FPGA design, we also developed a cycle-accurate C software simulation model for each FPGA model. In addition, we use micro benchmarks and the TCP incast application [17] to verify the correctness of our abstracted switch models.

Overall, the model of a rack of 64 servers consumes the most FPGA resources, and also requires the most design effort to support the full SPARC v8 standard. Abstract switch models for both circuit switching and packet switching have a very small logic resource requirement, with $< 1\%$ resource utilization on the FPGA we chose. For all three models, BRAMs are still the limiting factor for overall simulation density. On a four-FPGA board, such as BEE3, we can easily simulate up to 512 servers with 16 32-port ToR switches.

To scale to 10,000 servers, we could use a system with 20 BEE boards, as has been built before [12]. We would also need two BEE3 boards to simulate a datacenter switch together with several array switches to connect rack switches together. Note that the BEE3 board was designed for a different usage model three years ago, and uses older Virtex 5 FPGAs and is equipped with limited high-speed serial connectors. If we used a board that has the latest 28 nm FPGAs and optimized for high-radix FPGA–FPGA serial communications, we could greatly reduce the number of FPGAs used and lower the overall system cost of a 10,000-node system simulator from 88 FPGAs on 22 boards to 11 FPGAs on 11 boards.

5. COMPARING TO SOFTWARE MODELS

Both switch models are relatively easy to build with only around 3,000 lines of SystemVerilog code. The circuit-switching model is more straightforward and requires less design effort due to a simpler target design. Although FAME models allow us to conduct datacenter experiments at enormous scale with greater architecture detail, they do require more design effort. For example, a comparable cycle-accurate software packet-switch model only requires 500 lines of C++ code. However, this does not mean a software C model is easier to verify. We always develop the software model along with the FAME model, and verify the correctness of the two models against each other. The debugging of the software model has never been accomplished well ahead of the corresponding FAME model. On the other hand, the more detailed FAME model helps to find many timing-related bugs in our software simulator.

As a comparison, we also optimized and parallelized the equivalent C++ simulation model using *Pthreads*. We compiled the C++ module using 64-bit GCC4.4 with '-O3 -mtune=native -march=native', and measured the software simulator in a trace-replay mode with the CPU cache already warmed up. We ran the software simulator on an 8-Core Intel Xeon X5550 machine with 48 GB memory running the latest Linux 2.6.34 kernel. Figure 4 shows slowdowns of the multithreaded software model simulating different size 10 Gbps switches under two types of workload, i.e. full load with 64-byte packets and random load with random-size packets. When simulating a small 32-port switch, the single-thread software model has better performance than our threaded 100 MHz FAME-7 FPGA model. However, the simulation performance drops quickly when increasing the number of switch ports. Due to many fine-grained synchronizations (approximately every 50 ns in target time), software multithreading helps little when simulating small switches. When simulating a large switch configuration, we saw small sublinear speedups using two, or sometimes four, threads but the benefit of using more threads diminishes quickly. Profile results show that crossbar scheduling, which scans multiple virtual queues, accounts for a large fraction of the total simulation time. Other large overheads include cache misses for first time accesses to virtual queue structures as well as updating in-memory performance counters for each simulation quanta. On the other hand, CPU memory bandwidth is not at all a limiting factor, even when simulating a large switch configuration. Moreover, Figure 4 also illustrates that the workload significantly affects the simulation performance for large switch configurations.

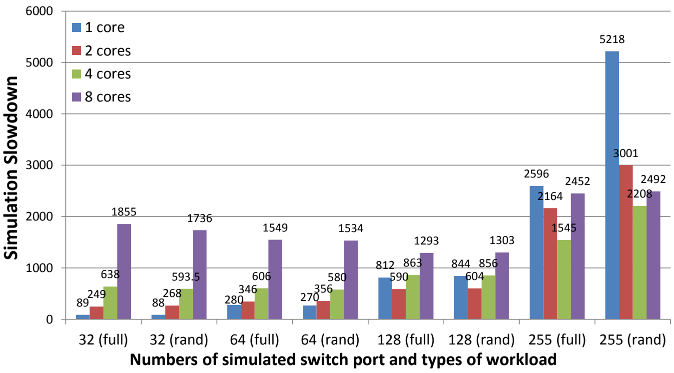

Figure 4: Parallel software switch simulation performance.

Note that we measured the software simulation performance under an unrealistic setting. In a real usage scenario, switch traffic will be generated dynamically by other models connected to the switch, which requires many more synchronizations over the input and output ports of the simulated switch. When simulating a large system containing many switches and servers, we believe it will be difficult to see any performance benefit by partitioning the software model across a high-performance cluster. Besides, future datacenter switches are very likely to be high-radix switches. Simulating architectures in even greater detail could also easily render the software approach impractical.

6. CONCLUSIONS

Our initial implementation and experience with applying the prototype to some real-world datacenter network research shows our FPGA-based approach is promising. Our future work primarily involves improving system capability using multiple FPGAs and scaling the software infrastructure running on our server model. We plan to quantitatively compare both circuit-switching and packet-switching datacenter network proposals using more real applications.

7. ACKNOWLEDGEMENT

This research is supported in part by gifts from Sun Microsystems, Google, Microsoft, Amazon Web Services, Cisco Systems, Cloudera, eBay, Facebook, Fujitsu, Hewlett-Packard, Intel, Network Appliance, SAP, VMWare and Yahoo! and by matching funds from the State of California's MICRO program (grants 06-152, 07-010, 06-148, 07-012, 06-146, 07-009, 06-147, 07-013, 06-149, 06-150, and 07-008), the National Science Foundation (grant #CNS-0509559), and the University of California Industry/University Cooperative Research Program (UC Discovery) grant COM07-10240.

8. REFERENCES

[1] Glen Anderson, private communications, 2009.

[2] Cisco Nexus 5000 Series Architecture: The Building Blocks of the Unified Fabric . . `http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/white_paper_c11-462176.html`, 2010.

[3] Force10 S60 High-Performance 1/10 GbE Access Switch, http://www.force10networks.com/products/s60.asp, 2010.

[4] Xilinx Virtex 7 Series FPGAs, http://www.xilinx.com/technology/roadmap/7-series-fpgas.htm, 2010.

[5] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[6] U. Cummings, D. Daly, R. Collins, V. Agarwal, F. Petrini, M. Perrone, and D. Pasetto. Fulcrum's FocalPoint FM4000: A Scalable, Low-Latency 10GigE Switch for High-Performance Data Centers. In *Proceedings of the 2009 17th IEEE Symposium on High Performance Interconnects*, pages 42–51, Washington, DC, USA, 2009. IEEE Computer Society.

[7] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *SIGCOMM '10*, pages 339–350, 2010.

[8] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, 2009.

[9] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *SIGCOMM '09*, pages 51–62, New York, NY, USA, 2009. ACM.

[10] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM '09*, pages 63–74, New York, NY, USA, 2009. ACM.

[11] R. Katz. Tech titans building boom: The architecture of internet datacenters. *IEEE Spectrum*, February 2009.

[12] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz. RAMP Blue: A Message-Passing Manycore System In FPGAs. In *Proceedings of International Conference on Field Programmable Logic and Applications*, pages 54–61, Amsterdam, The Netherlands, 2007.

[13] B. Kwan, P. Agarwal, and L. Ashvin. Flexible buffer allocation entities for traffic aggregate containment. US Patent 20090207848, August 2009.

[14] R. Liu et al. Tessellation: Space-Time partitioning in a manycore client OS. In *HotPar09*, Berkeley, CA, 03/2009 2009.

[15] M. Mitzenmacher, A. Broder, A. Broder, M. Mitzenmacher, and M. Mitzenmacher. Using multiple hash functions to improve ip lookups. In *In Proceedings of IEEE INFOCOM*, pages 1454–1463, 2000.

[16] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM '09*, pages 39–50, New York, NY, USA, 2009. ACM.

[17] Z. Tan, K. Asanović, and D. Patterson. An FPGA-based Simulator for Datacenter Networks. In *The Exascale Evaluation and Research Techniques Workshop (EXERT 2010), at the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010)*, March 2010.

[18] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson, and K. Asanović and. RAMP gold: An FPGA-based architecture simulator for multiprocessors. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 463 –468, 2010.

[19] Z. Tan, A. Waterman, H. Cook, S. Bird, K. Asanović, and D. Patterson. A case for FAME: FPGA architecture model execution. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 290–301, New York, NY, USA, 2010. ACM.

[20] C. Thacker. Rethinking data centers. October 2007.

[21] C. Thacker. A data center network using FPGAs, May 2010.