Piton: A Manycore Processor for Multitenant Clouds

Piton is a 25-core manycore processor that reimagines the datacenter architecture, breaking down barriers between chips, nodes, and racks, and enabling flexibility, performance, and energy efficiency at scale. It is designed not only as a single chip, but as a large-scale system. Piton supports shared memory across arbitrary cores in the system and is tailored to infrastructure-as-a-service clouds.

Michael McKeown
Yaosheng Fu
Tri Nguyen
Yanqi Zhou
Jonathan Balkind
Alexey Lavrov
Mohammad Shahrad
Samuel Payne
David Wentzlaff
Princeton University

Datacenters are expanding rapidly, fueled by cloud computing and the insatiable need for server computing power to drive web-hosted services. The conventional architecture of such large-scale systems is relatively hierarchical and rigid, with boundaries between chips, boards, nodes, and racks, and it follows a scale-out approach. Piton is a manycore processor that looks to break down these boundaries with a flat and flexible architecture. It is designed not only as a single chip, but also as a large, scalable system of up to 8,192 Piton chips (204,800 cores) connected together. Shared memory is maintained among arbitrary cores in the system, both intrachip and interchip, to support flexibility in shared systems and facilitate a scale-up approach with fine-grained control over resources. Piton is designed to exploit similar or identical code executing in a datacenter, as can be found in shared cloud systems, and enables novel infrastructure-as-a-service (IaaS) economic models.

Piton is a 25-core manycore processor that targets multitenant clouds and datacenters. It implements the 64-bit SPARC V9 instruction set architecture (ISA), which lets it support standard operating systems (OSs) and toolchains. Piton uses a modern tiled design and three 64-bit on-chip networks (NoCs) configured in a 2D mesh topology to facilitate seamless scalability. Cache coherence is maintained using a directory-based MESI (modified, exclusive, shared, invalid) cache coherence protocol at the shared, distributed L2 cache. The NoCs and coherence protocol extend off-chip to support system-wide shared memory, enabled by the use of coherence domains[1] to reduce the directory storage overhead and communication latency. A memory bandwidth provisioning mechanism improves utility for cloud users and providers,[2] allowing providers to charge based on the memory bandwidth an application or virtual machine needs and enforce a particular memory traffic distribution.

Energy efficiency and throughput are Piton's primary design goals, as these are important metrics in datacenters. Thus, we use an efficient multithreaded core and implement an energy-saving drafting mode[3] in each core to reduce the switching activity and instruction cache accesses.

Published by the IEEE Computer Society

Piton was taped-out on IBM's 32-nm silicon-on-insulator process with a 36 mm^2 die size, 6 mm × 6 mm. It contains more than 460 million transistors and has a target clock frequency of 1 GHz at the nominal 0.9 V supply voltage. This puts it among the largest chips built in academia to date. We have received silicon back from IBM and have tested it working in the lab, running multiple microbenchmarks and booting full-stack Debian Linux. The die contains 331 total pads and is packaged in a 208-pin ceramic quad flat package (QFP) with an epoxy encapsulation. Figure 1 shows the Piton die, wirebonds, and package (without the epoxy encapsulation).

Piton has also been open sourced as a research framework called OpenPiton (http://openpiton.org),[4] to facilitate at scale, realistic manycore research in many domains.

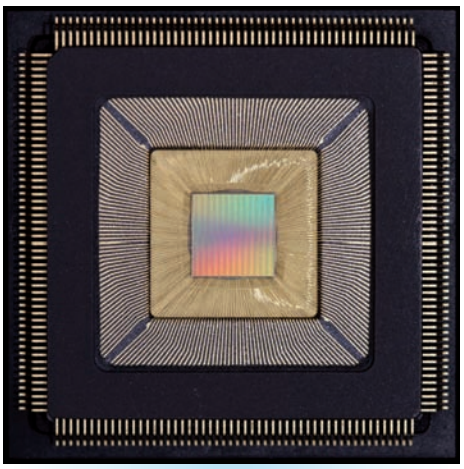

Figure 1. Piton die, wirebonds, and package without epoxy encapsulation.

Piton Architecture

Figure 2a shows Piton's high-level architecture. Piton uses a tile-based architecture, with 25 tiles connected in a 5-×-5 2D mesh topology—a topology used by other manycore designs.[5,6] The tiles are interconnected with three full-duplex 64-bit NoCs (128 bits per link per NoC), providing ample bandwidth for intercore communication. We use three separate NoCs to avoid protocol-level deadlock. The three NoCs are physically implemented and use dimension-ordered, wormhole routing for deadlock avoidance. We chose to implement the NoCs physically, as opposed to virtually, because we found wiring resources to be plentiful on chip. This made for a good tradeoff compared to the extra area required for virtual NoCs. The networks have a latency of one cycle per hop, plus an additional cycle for turns; use credit-based flow control; and provide a point-to-point ordering guarantee. The tiles maintain cache coherence at the shared, distributed L2 cache, using the NoC for communication.

The top of Figure 2a shows Piton's primary off-chip interface, the *chip bridge*. It connects the tile array to off-chip logic, namely the chipset, and to other Piton chips. The NoC routers route off-chip traffic to tile0, where the chip bridge connects to the tile array. The chip bridge comprises two 32-bit unidirectional links and multiplexes the three 64-bit physical NoCs using logical channels for communication over the pin-limited off-chip channel. This effectively extends the NoCs and coherence protocol off-chip, enabling shared-memory support across up to 8,192 chips or 204,800 cores. At the target off-chip clock frequency, 350 MHz, the chip bridge provides 2.8 Gbytes per second (GBps) of bandwidth. Compared to other manycore processors,[5,6] Piton's off-chip bandwidth is relatively low due to pin limitations. The low pin count was primarily a cost and design risk-reduction decision, but it lets us explore issues in future manycore designs, in which the number of cores scales faster than the available memory bandwidth, as a research exercise. For instance, we are interested in exploring possible solutions such as cache compression.[7]

Figure 2b shows the CAD tool layout screenshot of Piton, highlighting the 25 tiles and the chip bridge.

Tile Architecture

Figure 3 depicts a Piton tile's architecture and CAD tool layout screenshot. A tile is made up of a modified OpenSPARC T1 core[8]; an L1.5 cache; a slice of the shared, distributed L2 cache; three NoC routers; a floating-point unit (FPU); a cache-CPU crossbar (CCX) arbiter; and a memory traffic shaper. The tile's physical area is 1.17 mm^2.

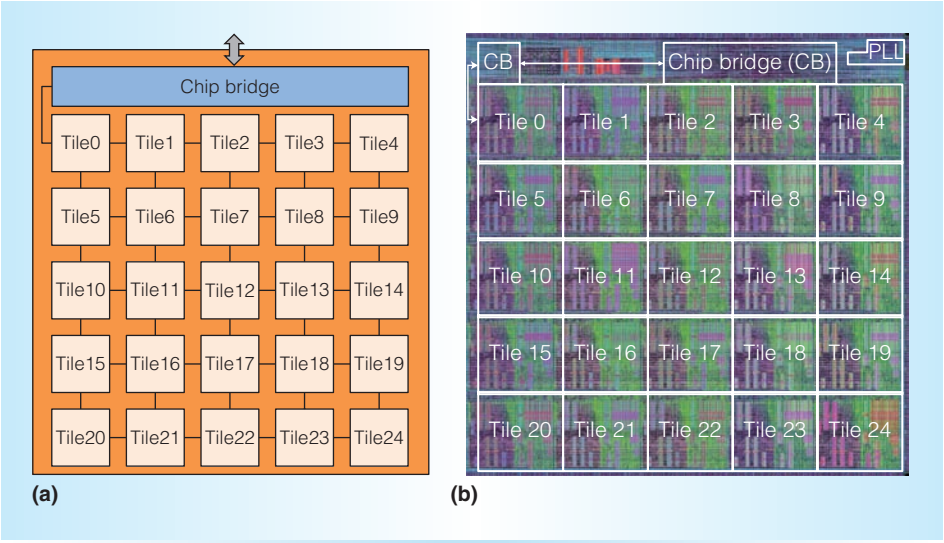

Figure 2. Piton chip architecture. (a) High-level architecture diagram. (b) Annotated CAD tool layout screenshot. The figure shows 25 tiles interconnected with networks on chip (NoCs) in a 2D mesh, as well as Piton's off-chip interface, the chip bridge.

The core contains two-way multithreading and implements an energy-efficient drafting mode. The L1.5 cache is an 8-Kbyte, inclusive, private data cache, implemented with a three-stage pipeline, and it has two primary purposes:

- to encapsulate the write-through L1 data cache in the core with a write-back cache, which reduces the bandwidth requirement to the shared, distributed L2 cache; and
- to transduce between the OpenSPARC T1 core (CCX protocol) and Piton's cache coherence NoC protocol.

The L1.5 connects to the slice of the distributed L2 cache through three NoC routers that implement the routing for the three physical networks. The NoC routers also connect to other tiles in the system. The L2 cache slice contains an integrated directory cache for Piton's MESI coherence protocol.

The FPU located in each tile is taken from the OpenSPARC T1. It is IEEE 754 compliant and is fully pipelined except for multiply and divide operations. Simple floating-point operations, such as move-type instructions, are implemented in the core, and the floating-point registers are also stored in the core. The FPU interfaces through the CCX protocol, so a simple CCX arbiter is included in each tile to arbitrate over the single, shared core CCX interface. We include an FPU per core, as opposed to the original OpenSPARC T1, which had one FPU for eight cores, to boost floating-point performance and simplify the design.

Finally, a memory traffic shaper that provisions memory bandwidth on a per-core basis communicates with the L1.5 cache. The memory traffic shaper shapes traffic on the basis of hit and miss information from the shared, distributed L2 cache.

Core and Drafting Mode

Piton repurposes a modified OpenSPARC T1 core. It is implemented with a six-stage in-order pipeline (see Figure 4a). The core is two-way multithreaded, which facilitates increased throughput and hiding of memory latency, which is important for many multitenant cloud applications. It contains a 16-Kbyte L1 instruction cache with a 32-byte line size and an 8-Kbyte L1 data cache with a 16-byte line size. Both caches are four-way set associative. The cache configurations are maintained from the OpenSPARC T1. The core implements the SPARC V9 64-bit ISA, which enables support for standard toolchains and OSs.

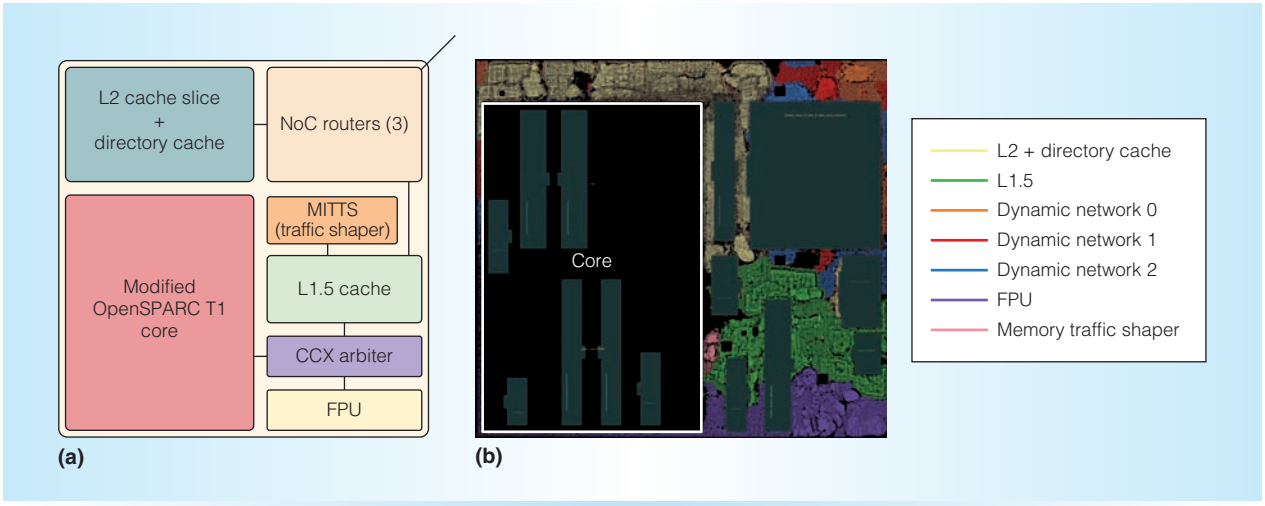

Figure 3. Piton tile architecture. (a) Architecture diagram. (b) Annotated CAD tool layout screenshot.

Figure 4b shows the annotated CAD tool layout screenshot of the core. The core's physical area is 0.55 mm^2.

Each core is augmented with an energy-efficient drafting mode.[3] The drafting mode aims to exploit identical or similar code executing in a datacenter—for example, Amazon's Elastic Compute Cloud (EC2), in which multiple tenants might run the same or similar versions of an application such as Apache. Another example can be found in Facebook, wherein many users simultaneously upload images that need to be resized and compressed.

Once duplicate code is scheduled to one of Piton's drafting-enabled multithreaded cores, the drafting hardware aligns the execution points of the threads in time to identical instructions using active synchronization. Piton's drafting mode has three synchronization methods that are configurable by software. This active synchronization can result in a small performance overhead, but once the threads are aligned to identical instructions, significant energy can be saved.

One source of energy savings is what we refer to as *drafting,* or issuing identical instructions from different threads consecutively down the core pipeline. Because decoding of identical instructions results in the same control signals, the activity factor is reduced in the core. Furthermore, if the data happens to be identical, the activity factor is reduced even more. Another source of energy savings comes from disabling the core's fetch stage when it knows the threads' code is identical. In effect, the drafting mode dynamically exploits the energy savings that single-instruction, multiple-data (SIMD) execution exploits, but for applications that are otherwise difficult to execute in a SIMD fashion. The drafting mode trades a small performance overhead for larger energy savings.

Figure 4a highlights the modifications to the core for the drafting mode. Piton's drafting mode area overhead is only 2.12 percent of the core.

Simulation results for the drafting mode executing Apache hosting different websites show a dramatic increase in throughput divided by energy, up to 20 percent in many cases and 8.57 percent on average, with minimal impact on single-threaded performance (0.13 percent on average), even in cases where there is little to no gain in throughput divided by energy.

L2 Cache, Coherence Protocol, and Coherence Domains

Piton's L2 cache is a distributed cache shared by all tiles in the system. Each tile contains a 64-Kbyte slice of the L2 cache, providing 1.6 Mbytes of aggregate L2 cache per chip. Although the L2 cache in aggregate is comparably small with respect to

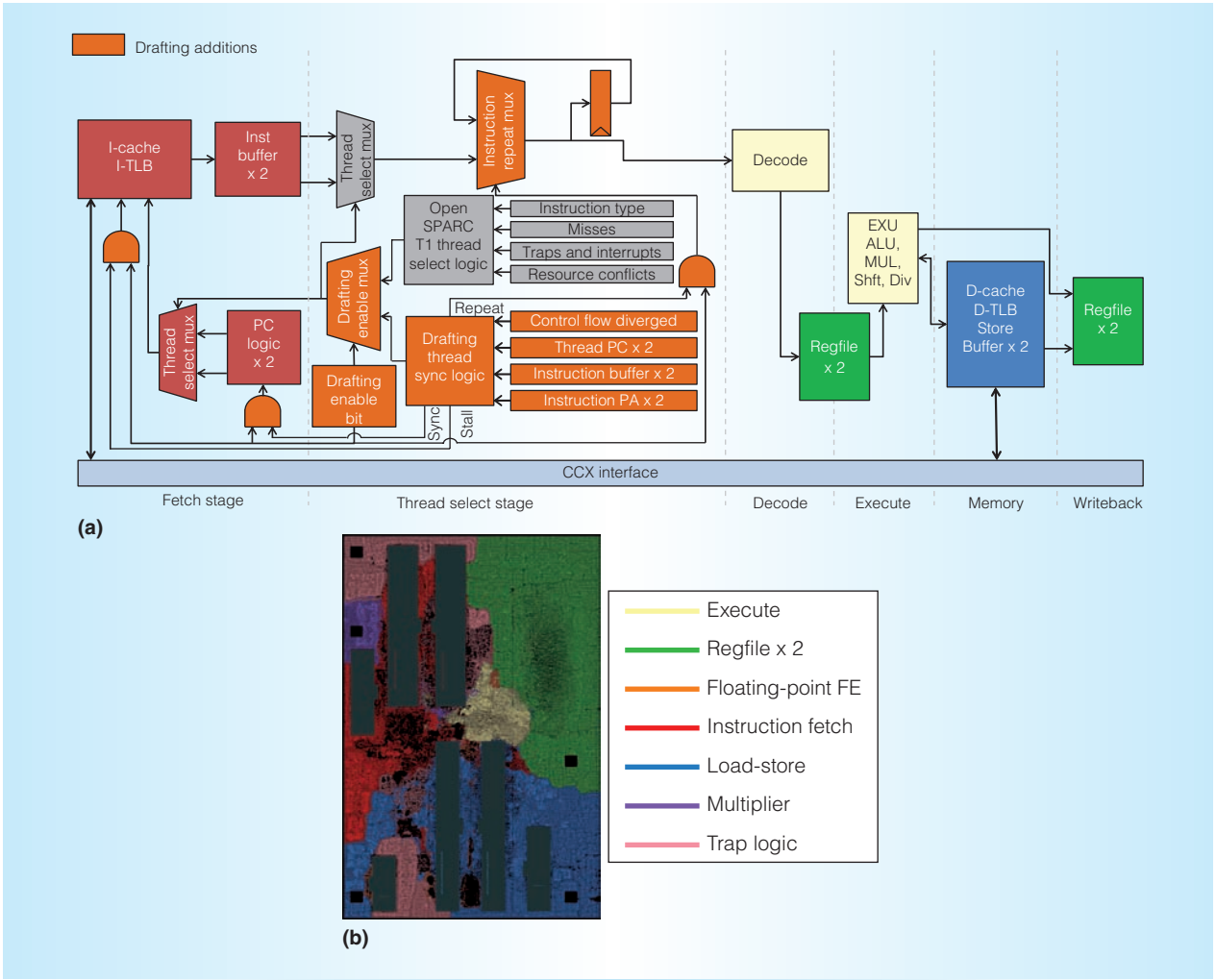

Figure 4. Modified OpenSPARC T1 core. (a) Six-stage in-order pipeline with drafting mode additions highlighted. (b) Annotated core CAD tool layout screenshot. The pipeline diagram is adapted from the *OpenSPARC T1 Microarchitecture Specification*.[9]

other manycore processors,[5,6] we target a multithreaded, throughput-oriented design to cover memory latency. Moreover, the shared, distributed nature of the cache, in contrast to other manycore processors with larger private caches,[5] facilitates sharing in cloud workloads and enables applications to dynamically use as much of the cache as needed. The L2 cache is four-way set associative and has a 64-byte line size. The cache is implemented with dual four-stage pipelines. One pipeline handles requests from the L1.5 and the other manages memory and L1.5 responses. Figure 5 illustrates the L2 cache pipelines, with the memory blocks implemented as SRAM arrays outlined in bold. This results in a five-cycle hit latency from when a request enters the cache to when a response leaves. The mapping of a cache line to an L2 cache slice (home placement) is configurable by software, permitting low-, middle-, or high-order address bit interleaving or the bitwise AND of the low- and middle-order address bits.

As Figure 5 shows, a directory-cache for the MESI coherence protocol is integrated into the L2 cache. The coherence protocol is implemented using 35 message types. The protocol uses four-hop message communication—that is, there is no direct communication between private L1.5 caches—as shown in Figure 6, which depicts the interfaces over which different blocks in the memory system

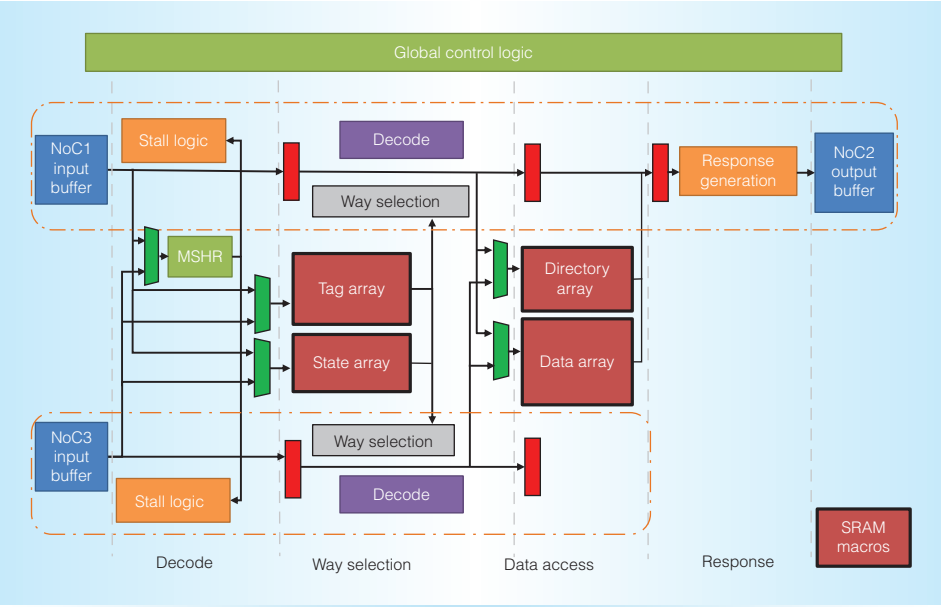

Figure 5. Shared, distributed L2 cache four-stage dual pipelines. Blocks outlined in bold are implemented as SRAM macros.

communicate. A cycle does exist in the coherence protocol communication graph, in which memory returns data to the L2 on NoC3 and the L2 returns data to the L1.5 on NoC2. We break this cycle using preallocation and prioritization in the L2 cache.

A unique feature of Piton's coherence protocol is the concept of coherence domains.[1] Coherence domains allow cores in the system, both intrachip and interchip, to be grouped by virtual machines, applications, or page accesses. Coherence needs to be maintained only within a coherence domain, not system-wide. Restricting the maximum coherence domain size, 64 cores in Piton, limits the directory storage to a constant, independent of the number of cores in the system. We chose 64 for the maximum coherence domain size because we found most cloud workloads do not scale beyond that. If a coherence domain grows larger than 64 cores, Piton reverts to a broadcast policy that allows for system-wide shared memory. Coherence domains are key to enabling Piton's 8,192-chip scalability with support for shared memory across arbitrary cores in the system. Without them, the directory storage overhead would be intolerable. In addition, coherence domains facilitate the dynamic relocation of L2 home placement to optimize for communication latency.

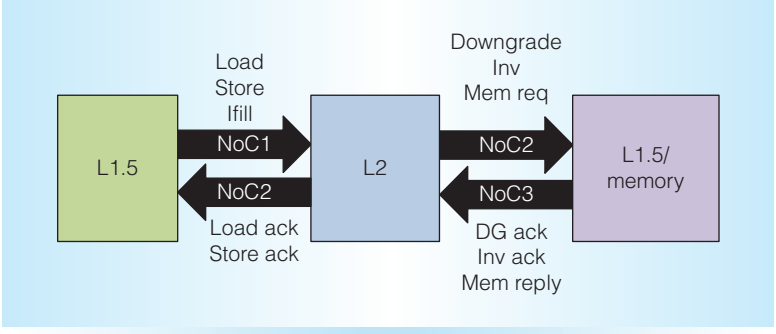

Figure 6. Coherence protocol interface diagram. Three physical networks along with preallocation and prioritization in the L2 cache are used to avoid protocol-level deadlock.

Figure 7 shows the hardware used to implement coherence domains. An additional indirection layer, called the *sharer map cache* (SMC), is used to map a small space of logical sharers to a larger space of physical cores. The translation lookaside buffer entries in the core are extended with coherence domain IDs, which are managed by software. The coherence domain IDs are transmitted over the NoC along with coherence requests. The coherence domain ID and logical sharer ID from the coherence directory's sharer vector are used as an index into the SMC to look

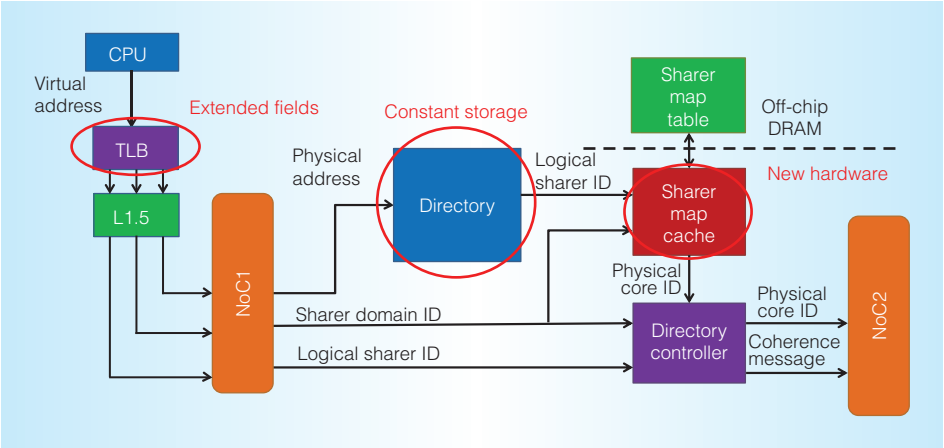

Figure 7. Coherence domain hardware implementation. An additional indirection layer maps a small space of logical sharers to a larger space of physical cores.[1]

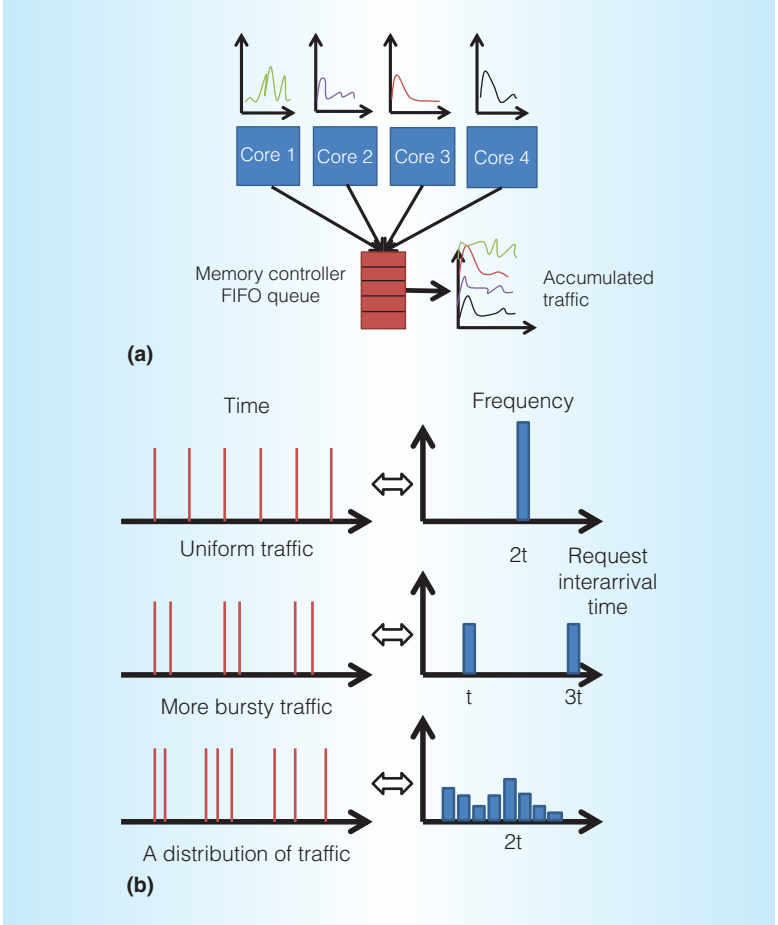

Figure 8. Memory traffic shaper motivating example. (a) Memory bandwidth problem in today's cloud systems. (b) Example memory interarrival time distributions.[2]

up the logical sharers' physical core IDs. These IDs can subsequently be used in coherence messages. The SMC is backed up by a full-size sharer map table in off-chip DRAM. Home relocation is accomplished in a similar way to restricting sharers by using an additional level of indirection on L2 home placement.

Memory Traffic Shaping

Piton includes a memory traffic shaper in each tile. The problem the memory traffic shaper addresses is that off-chip memory bandwidth is a significant limited resource and applications currently do not share it well. In today's multitenant cloud-based systems, cores contend for memory bandwidth with little to no restrictions (see Figure 8a). One core can consume a large portion of the memory bandwidth and starve other cores.

The solution implemented in Piton is to restrict core or application memory bandwidth to fit a particular distribution based on the temporal distance between memory requests, or the memory interarrival time. Thus, Piton's memory traffic shaper is called the Memory Interarrival Time Traffic Shaper (MITTS).[2] Figure 8b illustrates a few examples of different memory traffic distributions, with the left side representing time on the *x*-axis and memory requests as vertical lines. These plots effectively translate into histograms of

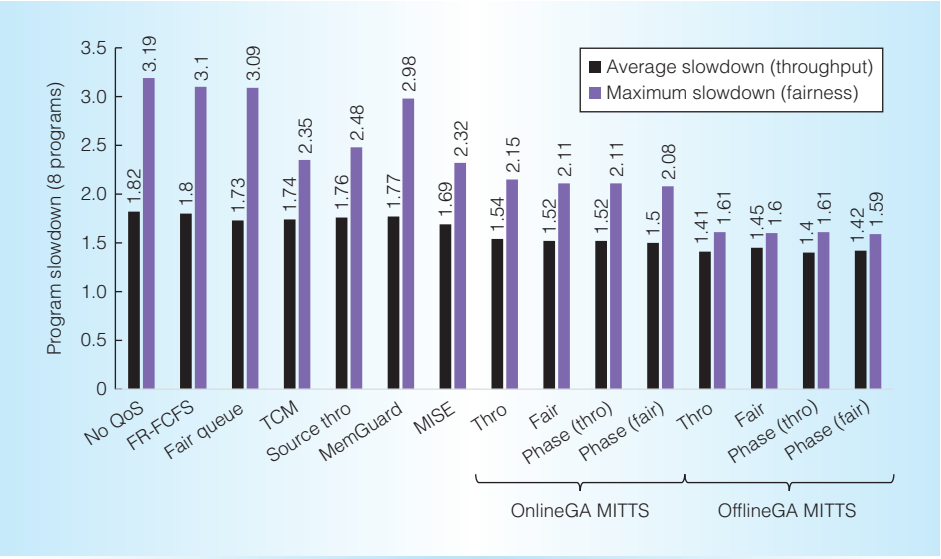

Figure 9. MITTS simulation results showing application slowdown when running a mix of Apache servers, mail servers, and SPECint applications contending for the same memory bandwidth. The left side shows MITTS versus other state-of-the-art memory schedulers. Lower is better.[2]

request interarrival times (shown on the right side of Figure 8b), the basis for shaping traffic with MITTS.

MITTS is implemented using a token bucket design. An array of bins, each representing different ranges of interarrival times, contains credits for requests. If a memory request enters the traffic shaper with an interarrival time corresponding to a bin with zero credits, MITTS stalls the request either until it falls into a bin with credits or until the credits are replenished, which occurs periodically. Users can specify a memory traffic distribution that fits their application by specifying appropriate bin credit replenishment configurations. Credits corresponding to different bins can be priced differently, according to an economic model, permitting users to pay commensurately for the memory bandwidth and distribution (bursty versus bulk) an application needs.

Alternatively, MITTS can be used as a traditional memory traffic scheduler. Figure 9 shows application slowdown (lower is better) simulating a mix of cloud workloads and SPECint applications contending for the same memory bandwidth. The bars on the left show average slowdown, a proxy for throughput, and the bars on the right show maximum slowdown, a proxy for fairness. The results from state-of-the-art memory schedulers[10,11] are shown on the left side of the graph. MITTS increases overall throughput while maximizing fairness compared to no traffic shaping and the state of the art.

System Architecture

A single-chip Piton system consists of a custom Piton test board accompanied by a chipset field-programmable gate array (FPGA) (see Figure 10). The custom Piton test board was designed at Princeton University and is based on the open source University of California, San Diego, Double Trouble Daughterboard (http://bjump.org). It comprises a 208-pin QFP socket to house Piton, which connects to a gateway FPGA, a Xilinx Spartan-6 that connects Piton's chip bridge interface over a FPGA Mezzanine Connector to a chipset FPGA board. The Piton board also includes voltage regulation to step-down a 12-V ATX power supply to the appropriate voltages to drive Piton and the interfacing devices and provides access to the Piton JTAG interface and configuration signals.

The chipset FPGA implements Piton's I/O and DRAM memory controller. We implemented the DRAM controller off-chip to reduce design risk. In the chipset FPGA, a

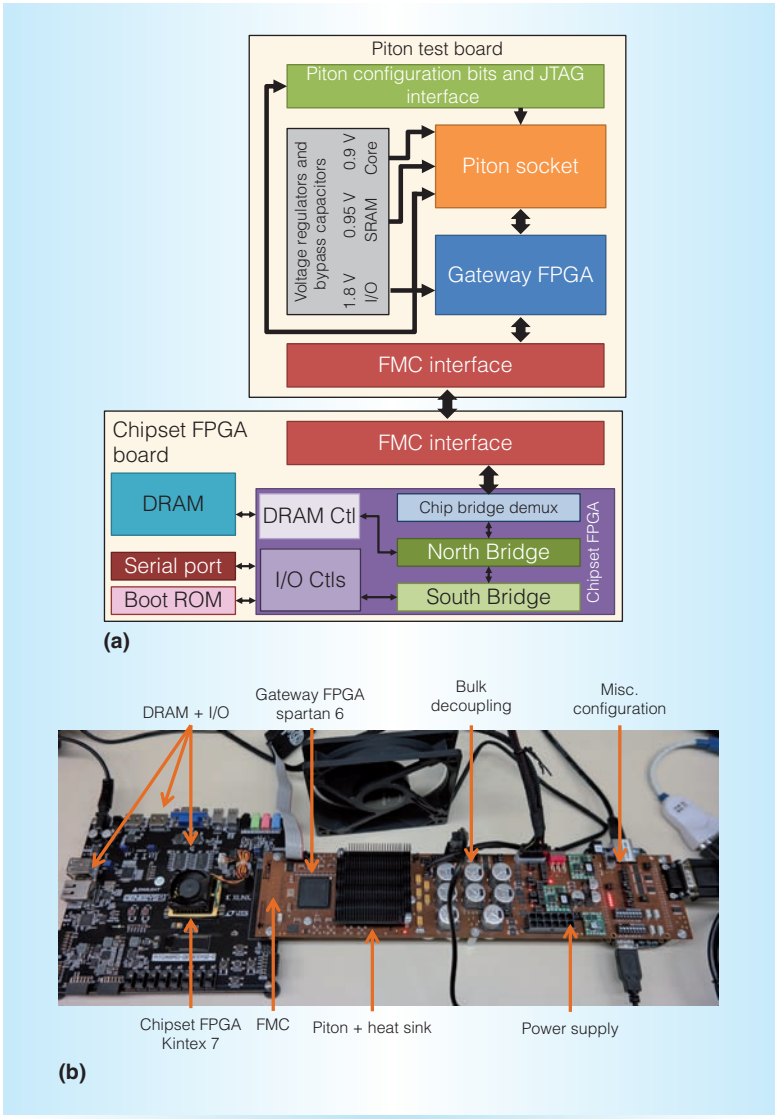

Figure 10. Piton system architecture. (a) Piton single-chip system block diagram and (b) photo.

chip bridge decoder demultiplexes the off-chip logical channels to the same three physical networks that exist in Piton. The network traffic is sent to the north and south bridges, which separate memory and I/O traffic, respectively, based on physical address values and direct it to the corresponding device controllers, also implemented in FPGA. This stand-alone system can boot standard OSs and has been successfully tested booting full-stack Debian Linux.

Piton can also be configured in a multichip system capable of supporting up to 8,192 Piton chips, enabled by Piton's coherence domains, connected with a packet-switched network similar to Piton's NoCs. The primary difference between the single-chip and multichip systems is the addition of interchip routers implemented in the chipset FPGA. Multiple interchip network topologies are possible. Logically, this system can be viewed as a large, flat pool of Piton tiles providing a high degree of flexibility, as memory can be shared among arbitrary cores in the system. This flexibility is ideal in shared cloud-based systems in which core fragmentation can become an issue during resource allocation. This system also promotes a scale-up paradigm, providing fine-grained control over resources similar to traditional distributed shared memory systems. This contrasts with current datacenters that have rigid boundaries between chips, boards, and racks; follow a scale-out approach; and are interconnected with networks that require traversing a network stack to a network interface card to communicate with other nodes in the system.

As the number of chips increases, the total number of cores available in the system also increases proportionally. However, to optimize for communication latency and bandwidth, we should consider the physical placement of cores. The bisection bandwidth for Piton's NoCs on a single chip is 288 GBps, which is two orders of magnitude higher than the off-chip bandwidth, so it is still much more efficient for an application to run on cores within the same chip rather than across multiple chips. The limited off-chip bandwidth has clearly become the bottleneck for multichip scalability, and we believe increasing off-chip bandwidth can alleviate this issue and enable more flexible multichip resource scheduling.

OpenPiton

OpenPiton is the open source release of Piton.[4] It includes everything used to design the chip: register transfer level (RTL) code, simulation infrastructure, test and validation suite, FPGA synthesis scripts, and application-specific integrated circuit (ASIC) synthesis and back-end scripts. The framework is modified from the original Piton chip framework to be highly configurable. The design can scale up to 536 million cores (65,536 cores intrachip with up to 8,192

chips), and the cache sizes and NoC topology are configurable. The FPGA synthesis scripts target multiple FPGA boards at different price points. The design can run at tens of megahertz on FPGA and boots full-stack Debian Linux, enabling users to run real applications.

OpenPiton is great for manycore research in multiple domains, from applications down to architecture. Researchers can make modifications to OpenPiton, including the processor RTL, compiler, and OS; seamlessly synthesize the design to FPGA; boot Debian Linux; and run real applications to evaluate their research ideas at scale and at speed. The large test and validation suite provides a safety net to ensure major functionality is maintained. The ASIC synthesis and back-end scripts aid in taping-out OpenPiton-based research chips, similar to Piton. OpenPiton immensely reduces the barrier of entry to at-scale, accurate, real-system manycore research. OpenPiton is also great for education and has already been used in the classroom. OpenPiton can be downloaded at http://openpiton.org.

In this article, we presented Piton, a 25-core manycore processor designed for IaaS clouds. We hope this work will encourage others to rethink the conventional datacenter design and build real systems in architecture research. We plan to build larger systems involving Piton and continue to explore the challenges in datacenter architectures, including I/O limitations, communication latency, energy efficiency, network topologies, fault tolerance, and continued performance scaling in the post-Moore's law era. We will also continue to support and improve OpenPiton to aid other researchers in doing real system manycore research. MICRO

References

1. Y. Fu, T. Nguyen, and D. Wentzlaff, "Coherence Domain Restriction on Large Scale Systems," *Proc. 48th Int'l Symp. Microarchitecture*, 2015, pp. 686–698.
2. Y. Zhuo and D. Wentzlaff, "MITTS: Memory Inter-Arrival Time Traffic Shaping," *Proc. 43rd Int'l Symp. Computer Architecture*, 2016, pp. 532–544.
3. M. McKeown, J. Balkind, and D. Wentzlaff, "Execution Drafting: Energy Efficiency through Computation Deduplication," *Proc. 47th Int'l Symp. Microarchitecture*, 2014, pp. 432–444.
4. J. Balkind et al., "OpenPiton: An Open Source Manycore Research Framework," *Proc. 21st Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2016, pp. 217–232.
5. A. Sodani, "Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor," *Proc. Hot Chips 27th Symp.*, 2015; doi:10.1109/HOTCHIPS.2015.7477467.
6. C. Ramey, "Tile-GX100 Manycore Processor: Acceleration Interfaces and Architecture," *Proc. Hot Chips 23rd Symp.*, 2011; doi:10.1109/HOTCHIPS.2011.7477491.
7. T. Nguyen and D. Wentzlaff, "MORC: A Manycore-Oriented Compressed Cache," *Proc. 48th Int'l Symp. Microarchitecture*, 2015, pp. 76–88.
8. P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded SPARC Processor," *IEEE Micro*, vol. 25, no. 2, 2005, pp. 21–29.
9. *OpenSPARC T1 Microarchitecture Specification*, Sun Microsystems, report no. 819-6650-11, Apr. 2008.
10. M. Caccamo et al., "MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-Core Platforms," *Proc. IEEE 19th Real-Time and Embedded Technology and Applications Symp.*, 2013, pp. 55–64.
11. L. Subramanian et al., "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems," *Proc. IEEE 19th Int'l Symp. High Performance Computer Architecture*, 2013, pp. 639–650.

Michael McKeown is a PhD candidate in the Department of Electrical Engineering at Princeton University. His research interests include parallel computer architecture, cloud computing, and energy efficiency. McKeown received an MA in electrical engineering from Princeton University. Contact him at mmckeown@princeton.edu.

Yaosheng Fu is a PhD candidate in the Department of Electrical Engineering at Princeton University. His research interests include computer architecture, memory systems, parallel computing, and distributed systems. Fu received an MA in electrical engineering from Princeton University. Contact him at yfu@princeton.edu.

Tri Nguyen is a PhD candidate in the Department of Electrical Engineering at Princeton University. His research focuses on manycore processors and memory bandwidth requirements. Nguyen received an MA in electrical engineering from Princeton University. Contact him at trin@princeton.edu.

Yanqi Zhou is a PhD candidate in the Department of Electrical Engineering at Princeton University. Her research focuses on configurable architecture and fine-grained resource provisioning for clouds. Zhou received a BS in electrical and computer engineering from University of Michigan and Shanghai Jiao Tong University. Contact her at yanqiz@princeton.edu.

Jonathan Balkind is a PhD candidate in the Department of Computer Science at Princeton University. His research interests include computer systems, programming languages, and computer architecture with the aim of improving the efficiency of modern multicore systems in mobile and datacenter environments. Balkind received an MSci in computing science from the University of Glasgow and an MA in computer science from Princeton University. Contact him at jbalkind@princeton.edu.

Alexey Lavrov is a PhD candidate in the Department of Electrical Engineering at Princeton University. His research interests include multitenant I/O devices, multicore processor design, and reconfigurable architectures. Lavrov received an MA in electrical engineering from Princeton University and an MA in applied physics and math from the Moscow Institute of Physics and Technology. Contact him at alavrov@princeton.edu.

Mohammad Shahrad is a PhD candidate in the Department of Electrical Engineering at Princeton University. His research focuses on combining computer systems, computer architecture, and economics to improve the efficiency of large-scale computing systems. Shahrad received an MA in electrical engineering from Princeton University. Contact him at mshahrad@princeton.edu.

Samuel Payne is a system software engineer at Nvidia. His work focuses on writing efficient software to abstract the technology in Tegra embedded processors for high-performance embedded and AI applications. Payne received a BS in electrical engineering from Princeton University, where he completed the work for this article. Contact him at spayne@nvidia.com.

David Wentzlaff is an assistant professor in the Electrical Engineering Department at Princeton University. His research interests include parallel computer architecture, architectures for cloud computing, and biodegradable computing systems. Wentzlaff received a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology. He has received the NSF CAREER award, the DARPA Young Faculty Award, the AFOSR Young Investigator Prize, and the Princeton E. Lawrence Keyes Faculty Advancement Award. Contact him at wentzlaf@princeton.edu.