SPARC T4: A DYNAMICALLY THREADED SERVER-ON-A-CHIP

THE SPARC T4 IS THE NEXT GENERATION OF ORACLE'S MULTICORE, MULTITHREADED 64-BIT SPARC SERVER PROCESSOR. IT DELIVERS SIGNIFICANT PERFORMANCE IMPROVEMENTS OVER ITS PREDECESSOR, THE SPARC T3 PROCESSOR. THE AUTHORS DESCRIBE SPARC T4'S KEY FEATURES AND DETAIL THE MICROARCHITECTURE OF THE DYNAMICALLY THREADED S3 PROCESSOR CORE, WHICH IS IMPLEMENTED ON SPARC T4.

Manish Shah
Robert Golla
Greg Grohoski
Paul Jordan
Jama Barreh
Jeff Brooks
Mark Greenberg
Gideon Levinsky
Mark Luttrell
Christopher Olson
Zeid Samoail
Matt Smittle
Tom Ziaja
Oracle

Starting with the Niagara family of processors, Oracle has been a pioneer in the field of highly threaded microprocessors. Sparc T1, the first-generation processor of the Niagara family, incorporated eight processor cores with a total of 32 threads on a single chip, thus achieving industry-leading throughput performance on a single chip.[1] Sparc T2, the next processor, delivered twice the performance of Sparc T1 with hardware support for 64 threads on a single chip.[2,3] In 2010, Oracle introduced the Sparc T3 microprocessor with 16 processor cores and 128 threads, which again doubled the throughput performance over Sparc T2.[4]

Sparc T4, Oracle's latest multithreaded processor, significantly increases single-threaded computational capabilities over Sparc T3, while doubling the per-thread throughput performance.

Overview of improvements

Sparc T4 is socket-compatible with Sparc T3 and uses T3's system-on-chip (SoC) components with, essentially, no changes. Sparc T4 enhances the overall cryptographic performance, maintains compatibility with Sparc Version 9 (V9) and CMT (chip multithreading technology) models, and extends reliability and serviceability (RAS) capabilities over T3. Sparc T4 implements eight Sparc S3 cores and has hardware support for eight threads on each core. The newly designed Sparc S3 core can run at aggressive clock frequencies in excess of 3.0 GHz and incorporates advanced branch prediction, out-of-order execution, instruction-based cryptography, and a three-level cache hierarchy consisting of private Level 1 (L1) and Level 2 (L2) caches within each Sparc core. Furthermore, T4 has a unified Level 3 (L3) cache shared among all eight cores. These architectural enhancements deliver a 5× improvement in single-thread integer performance over T3. The highly integrated Sparc T4, with 64 threads, delivers twice the per-thread throughput performance as Sparc T3 in the same power envelope.

Figure 1 shows a block diagram of the Sparc T4 processor. The eight Sparc cores share an eight-way banked, 4-Mbyte L3 cache. Each bank of the L3 cache is 16-way set associative, with a line size of 64 bytes. The L3 cache connects to two dual-channel, on-chip DDR3-1066 memory controllers, which communicate with external double data rate type three (DDR3) dual in-line memory modules (DIMMs) through four high-speed serial links. Each memory controller supports an aggregate read/write

Published by the IEEE Computer Society

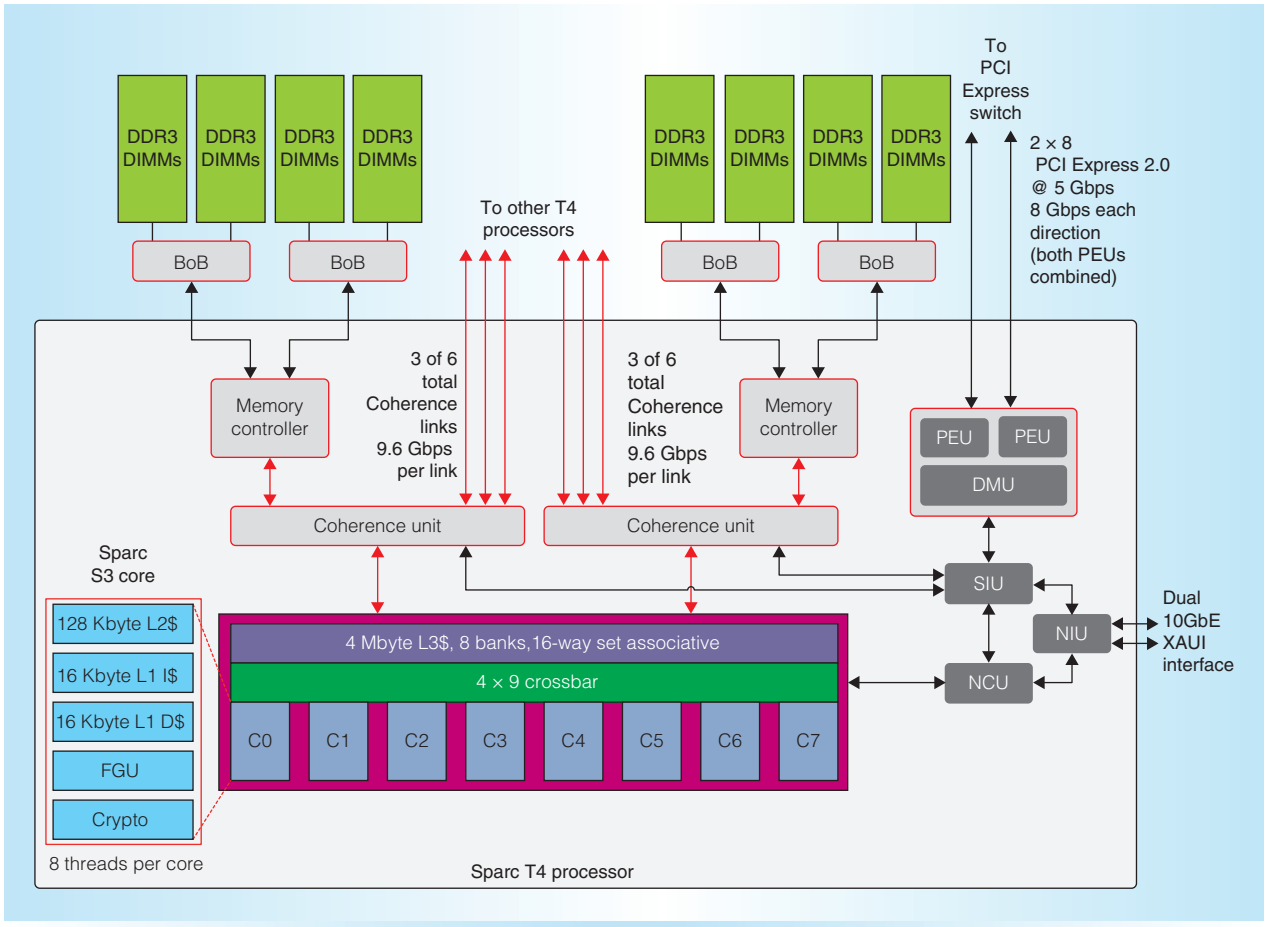

Figure 1. Sparc T4 block diagram. T4 consists of eight Sparc S3 cores, a 4-Mbyte L3 cache, two dual-channel memory controllers, and six coherency links to connect with four other T4 processors without requiring an external hub chip. (PEU: PCI Express unit; DMU: data management unit; NIU: network interface unit; NCU: noncacheable unit; SIU: system interface unit.)

bandwidth of 63 GBps. Sparc T4 supports two (x8) PCI Express Gen2 ports, which are capable of operating at 5 GBps bi-directionally. Sparc T4 also provides dual 10-Gbit Ethernet ports. Sparc T4's coherency links support up to four Sparc T4 processors in a system without requiring any additional glue logic.

Figure 2 shows a die photo of Sparc T4. Sparc T4 is manufactured in Taiwan Semiconductor Manufacturing Company's 12-metal, 1.0 V, high-performance, 40-nm process; it consists of about 855 million transistors. Sparc T4's die size is 403 mm^2. The eight Sparc S3 cores, the L3 cache, and the processor-cache crossbar (CCX) are outlined in the figure.

Sparc S3 processor core

The Sparc T4 processor implements eight high-performance Sparc S3 cores, each of which has full hardware support for executing eight hardware threads. The Sparc S3 core incorporates a 16-stage integer pipeline to support high-frequency operations in excess of 3.0 GHz, and has an out-of-order, dual-issue execution engine with a dynamically threaded pipeline. Figure 3 shows the Sparc S3 processor core's block diagram.

Cache hierarchy and MMU

Each Sparc S3 processor core's cache hierarchy consists of a 16-Kbyte, four-way set associative L1 instruction cache (L1I) and a 16-Kbyte, four-way set associative L1 data

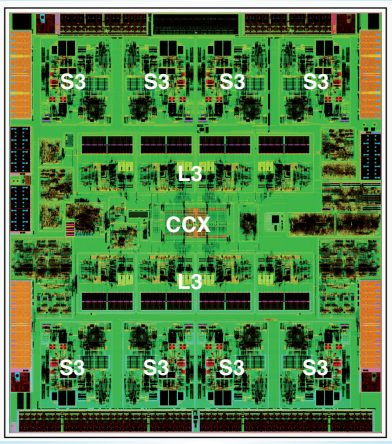

Figure 2. Sparc T4 chip micrograph. T4 consists of 855 million transistors and is manufactured in 40-nm process with a die size of 403 mm^2. (CCX: processor-cache crossbar.)

cache (L1D), backed by a unified, private, 128-Kbyte, eight-way set associative L2 cache. The L1I and L1D have a 32-byte line size. All eight threads on the processor core share the caches, resulting in efficient utilization of the silicon area. The instruction cache incorporates a sequential cache-miss prefetcher, which prefetches up to three sequential lines upon a cache miss. The data cache incorporates a sophisticated, stride-based prefetcher. The L2 cache has a 32-byte line size and incorporates a sequential next-line prefetcher.

The processor core's memory management unit (MMU) consists of a 64-entry, fully associative instruction translation lookaside buffer (ITLB) and a 128-entry, fully associative data TLB (DTLB). The instruction and data TLBs are shared by all eight threads on the processor core. The MMU supports 8-Kbyte, 64-Kbyte, 4-Mbyte, 256-Mbyte, and 2-Gbyte page sizes and implements a hardware table-walk engine for fast TLB reloading. The hardware table-walk engine is shared by all eight threads and supports up to 24 pending TLB misses.

Instruction fetching

The processor core's instruction fetch unit (IFU) fetches up to four instructions every cycle from the instruction cache and delivers them to the per-thread, 32-entry instruction buffers. The IFU selects a new thread to fetch every cycle, based on a least-recently-fetched algorithm. A thread can be in one of two states: Ready or Wait. A thread is placed in Wait if it encounters a stalling condition such as an instruction cache miss or instruction TLB miss. A thread in Wait doesn't take part in thread selection arbitration, which allows other threads to access the fetch resources. The IFU is vertically threaded. Each pipe stage of the IFU can process a different thread concurrently.

The IFU is also responsible for generating the next fetch address for every thread and uses advanced branch prediction techniques for determining the next fetch address. Each Sparc S3 core implements a two-level branch prediction scheme. Level 1 consists of a small, low-latency branch target cache (BTC), which is backed by a highly accurate perceptron-based direction predictor for conditional branches.[5] Our evaluations found that the perceptron-based direction predictor offered the best cost-performance tradeoff for both single-threaded and multithreaded workloads.

Our studies have shown that most relative branches have their targets within 8 Kbytes of the branch's address. For these branches, classified as near branches, the instruction is modified by having the partial target of the branch overlaid in the branch instruction's displacement field. This modification of the instruction is done when the instruction is written into the instruction cache. When the branch instruction is fetched from the instruction cache, its target is formed by concatenating the branch's address and the branch instruction's modified displacement field. This mechanism avoids the need for computing the target once the instruction is fetched from the instruction cache and eliminates a fundamental speed path. Because the target is available as soon as the instruction is read out of the instruction cache, the Sparc processor core doesn't need a large branch target buffer (BTB) to predict the targets.

The IFU implements a 512-entry branch far table (BFT) for predicting branches with targets beyond 8 Kbytes of the branch's address. The IFU also implements a 512-entry indirect branch table (IBT) for predicting

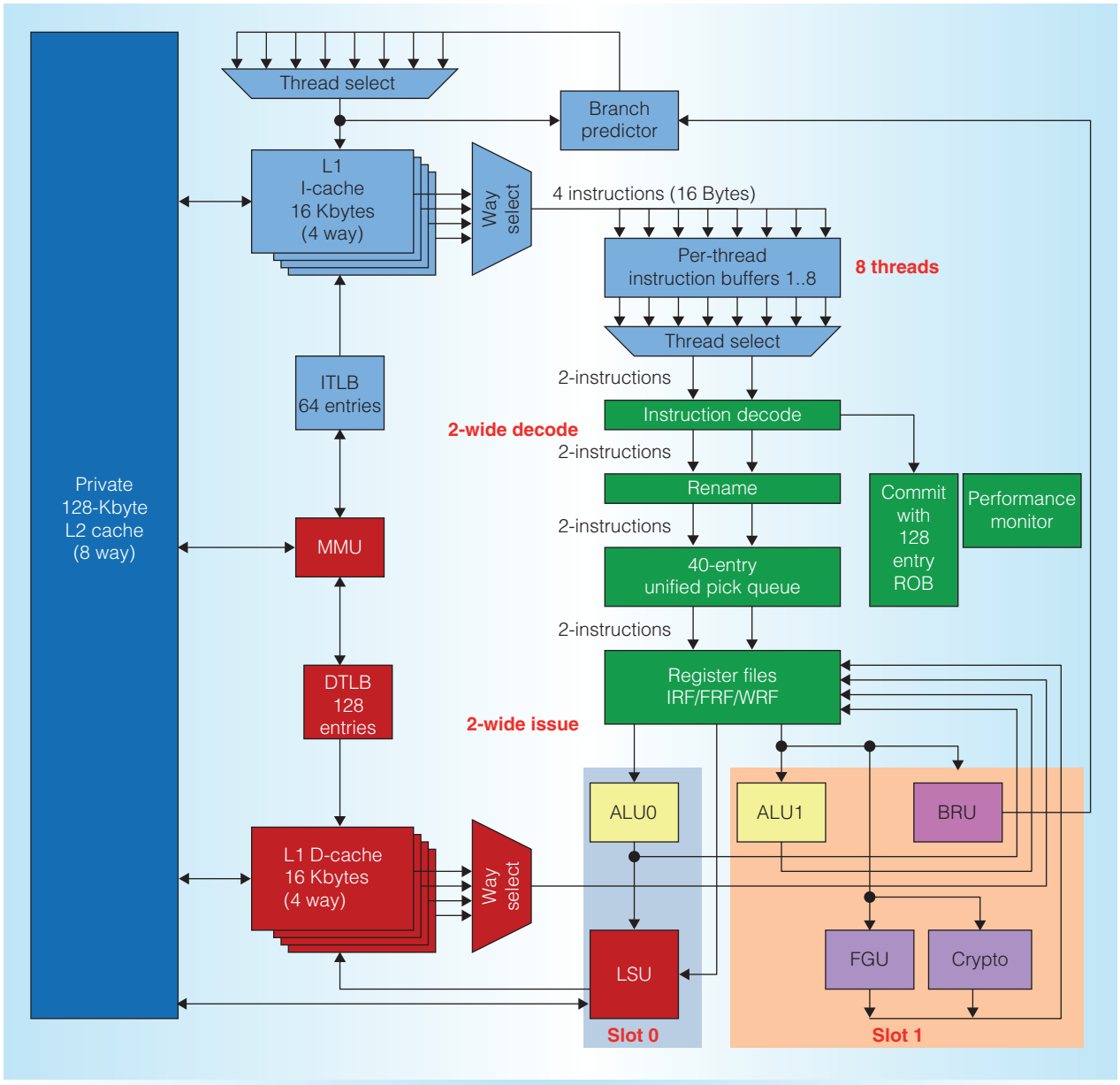

Figure 3. S3 processor core block diagram. Sparc T4 fetches four instructions per cycle while it decodes and issues two instructions per cycle to the executions units. (DTLB: data translation lookaside buffer; ITLB: instruction TLB.)

indirect branches. A per-thread, 16-entry return stack is used for predicting the return instructions' targets. All eight threads on the processor core dynamically share the BTC, BFT, IBT, and perceptron direction predictors.

Decode and rename

The decode unit (DU) selects a new thread to decode every cycle, based on a least-recently-decoded algorithm. Two instructions for the selected thread are read out of the 32-entry instruction buffers for that thread, and then are decoded and delivered to the rename unit (RU). For decode selection purposes, a thread can be in one of two states: Ready or Wait. A thread can be placed in Wait owing to a lack of valid instructions, a synchronization condition, or other instruction-related conditions.

A thread in Wait doesn't take part in thread selection arbitration, which lets other threads access the decoder resources.

The Sparc S3 processor core's decode unit has support for several new instructions, including a Pause instruction and a fused compare-branch instruction. The decode unit also supports user-level cryptographic instructions.

The rename unit renames the source and destination registers of two instructions each cycle and resolves destination-source dependencies between instructions within a thread. The Sparc S3 processor core's execution units are divided into two slots. Slot 0 contains an integer execution unit (ALU0) and a load/store unit (LSU), while slot 1 contains an integer execution unit (ALU1), a branch unit (BRU), a cryptographic unit (Crypto), and a floating-point and graphics unit (FGU). The Rename unit assigns issue slots to the instructions and creates an age-vector-based dependency for each instruction on the basis of its issue slot. The rename unit delivers two renamed instructions every cycle to the pick queue (PQ).

The decode and rename units are vertically threaded, with each pipe stage capable of independently processing a different thread.

The Sparc S3 processor core has a unified, 40-entry PQ that selects the oldest ready-to-execute instruction per slot each cycle. The PQ is thread agnostic and picks up to two instructions each cycle. An instruction is considered ready if all its sources are available.

Execution units

The integer execution unit (EXU) can execute up to two instructions per cycle. Single-cycle integer instructions are executed in either the ALU0 (slot0) or ALU1 (slot1). Load and store address operations are executed in ALU0. Branch instructions, floating-point instructions, multicycle integer instructions, and cryptographic instructions are executed in ALU1.

The load store unit (LSU) processes all memory reference instructions and properly orders all memory references. The LSU receives load and store instructions out of order from the PQ. Loads may be issued out of order with respect to other loads, and stores may be issued out of order with respect to other loads and stores. However, loads will not be issued ahead of previous stores whose addresses are unknown. In addition to the memory references required by the instruction set, LSU contains a stride-based hardware prefetcher, which prefetches data into the L1 data cache based upon detected access patterns.

Instructions are executed and completed out-of-order by the execution units. The Trap Logic Unit (TLU) commits instructions, maintains machine state, and handles exceptions and interrupts. Instructions are committed in order within a thread but independently between threads. Up to 128 instructions can be in flight within the processor core, in any combination across the active threads.

Figure 4 shows the Sparc S3 core pipeline.

Cryptographic unit

Oracle Sparc Architecture CMT processors have always provided hardware support for a range of cryptographic operations. UltraSparc T2 and Sparc T3 contained discrete, hyper-privileged, per-core coprocessors. Cryptographic performance for small packets on these coprocessors was limited by the software overhead associated with processing each packet. Sparc T4 dispenses with per-core coprocessors and, instead, provides access to integrated cryptographic accelerators, which are implemented as functional units within the processor core via nonprivileged instructions. These instructions accelerate bulk ciphers, secure hashes, and public-key algorithms. The cryptographic functionality on Sparc T4 is designed to achieve wire-speed encryption and decryption on Sparc T4's 10-Gbyte Ethernet ports.

Each of the eight Sparc S3 cores on Sparc T4 implements a cryptographic unit designed to accelerate generic encryption and authentication operations as well as important protocols, such as Secure Sockets Layer (SSL), Transport Layer Security (TLS), and Internet Protocol security (IPsec). The cryptographic unit on S3 supports new on-chip Encryption Instruction Accelerators and authentication engines with direct nonprivileged support for 16 industry-standard cryptographic algorithms. The following algorithms are supported:

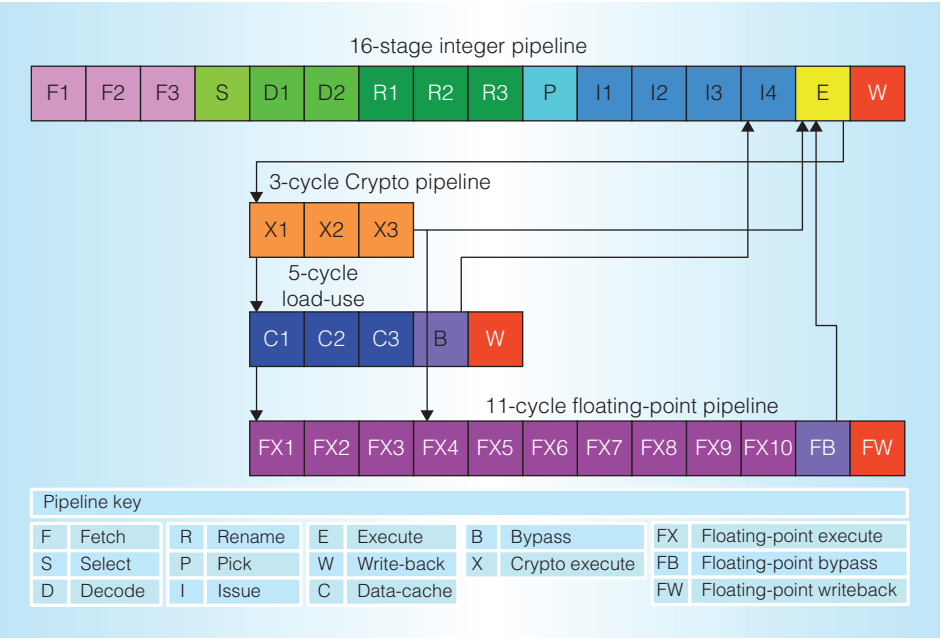

Figure 4. Sparc S3 core pipeline. S3 consists of a 16-stage integer pipe, a 20-stage load-store pipe, and a 27-stage floating-point graphic pipe.

Advanced Encryption Standard (AES), Camellia, Cyclic Redundancy Code (CRC) 32c, Data Encryption Standard (DES), 3DES, Diffie-Hellman (DH), Digital Signature Algorithm (DSA), elliptic curve cryptography, Kasumi, Message Digest 5 (MD5), Rivest Shamir Adleman (RSA), Secure Hash Algorithm (SHA)-1.

Sparc T4 implements symmetric ciphers such that a single instruction can perform a significant portion of a round. Secure hashes are implemented such that a single instruction performs a single block of the hash operation.

Multiple-precision modular multiplications are key components for encrypting and signing digital data in applications such as public-key cryptography. Traditionally, these algorithms have been implemented in software because they're expensive to build in hardware. Sparc T4 provides single-instruction support for multiple precision multiply (MPMUL), Montgomery multiplication (MONTMUL), and Montgomery squaring (MONTSQR) algorithms. For example, a single MPMUL instruction can execute a 2,048 × 2,048-bit multiplication in approximately 1,100 cycles.

Dynamic threading

Most Sparc S3 core resources are shared among all active threads. Examples of shared core resources include branch prediction structures, load buffer entries, store buffer entries, reorder buffer entries, PQ entries, load-miss buffer entries, and DTLB-miss buffer entries. A policy of static resource allocation statically divides a shared resource among all active threads; this isn't optimal for mixed workloads, which consist of one or more high-IPC (instructions per cycle) threads concurrently executing with one or more low-IPC threads. High-IPC threads execute more instructions per clock. A thread with good temporal and spatial locality often exhibits high-IPC behavior. Conversely, low-IPC threads often have poor temporal or spatial locality. With static allocation, low-IPC threads typically have more resources than needed, while high-IPC threads have fewer resources than needed. This imbalance limits overall core-throughput performance.

Dynamic resource allocation has better overall performance for mixed workloads than static allocation. Each thread is allocated resources based on need. Resource

constraints are relaxed versus static allocation, such that high-IPC threads can have more resources for execution and low-IPC threads less.

The Sparc S3 core is dynamically threaded. The shared resources are dynamically and seamlessly allocated between all active threads within the core. Scalability in software applications is often impacted by an application's fundamental serial content. Per Amdahl's law, the ultimate speedup in an application is limited by the fractional amount of time spent executing an application's serial component. Dynamic resource allocation reduces the time needed for the execution of an application's serial component. This directly improves overall software application scalability.

Dynamic resource allocation on a Sparc S3 core is done each cycle, based on the number of active threads currently executing. Because the core dynamically allocates resources among the active threads, there is no explicit single-thread mode or multithread mode for software to activate or deactivate. If software parks or halts all threads except one on a core, the core devotes all of its resources to the sole running thread. Thus, that thread will run as quickly as possible. Similarly, if software parks six out of eight threads, the two active threads share core execution resources. The hardware doesn't require synchronization because software adds or removes threads from a core.

As a further example of software visible benefits of dynamic threading, the Sparc S3 core implements a new user-level Pause instruction, which can be used to pause a thread for a fixed number of cycles. While a thread is paused, it essentially uses little or no resources within the core. Active threads on the core can use resources that are freed by a paused thread, thus improving overall core performance. After the fixed number of cycles has elapsed, the thread is restarted and begins dynamically allocating shared resources. The Pause instruction is useful when implementing various lock acquisition and lightweight sleep constructs.

Different thread execution characteristics

The extent to which threads compete for core resources depends on their execution characteristics, such as cache and TLB footprints, interinstruction dependencies in their own execution streams, and branch prediction effectiveness. For example, consider a process with a small cache footprint and low branch misprediction rate, such that—when running alone on a core—it achieves two instructions per cycle (which is the peak rate of instruction execution for Sparc S3 core): this a high-IPC process. If another process with similar characteristics is activated on a different thread on the same core, each of the threads will likely operate at approximately one instruction per cycle. In other words, the single-thread performance of each process has been cut in half. As a rule of thumb, activating N high-IPC threads will result in each thread executing at $1/N$ of its peak rate, assuming each thread can execute close to two instructions per cycle.

Now consider a process that is largely memory bound. Its native IPC will be small, perhaps 0.2. If this process runs on one thread on a core with another clone process running on a different thread, there's a good chance that both threads will suffer no noticeable performance loss, and the core throughput will improve to 0.4 IPC. If a low-IPC process runs on one thread with a high-IPC process running on another, it's unlikely that either thread's IPC will be greatly perturbed, though the high-IPC thread might suffer a slight performance degradation (as long as the low-IPC thread doesn't cause a substantial increase in cache- or TLB-miss rates for the high-IPC thread). Dynamic threading allows core throughput to approach the theoretical maximum for different sets of thread execution characteristics.

Thread hogs

One issue with dynamic resource allocation is the concept of a thread hog—a thread that fails to release its shared resources in a timely fashion. If left unmitigated, a thread hog can eventually take over most or all of the core's shared resources and severely limit its overall throughput performance. A simple example of a thread hog is a thread executing sequentially dependent integer divides. If such a thread were to run with

other nonthread-hog threads, it would eventually take over all of the PQ's entries.The Sparc S3 decode unit stalls once the PQ is fully allocated. Thus, all threads would have to wait at the decode pipe stage for the thread hog to deallocate a PQ entry before being allowed down the pipe. Because integer divides typically are long-latency operations, this would result in a significant loss of throughput performance. To prevent this from happening, the Sparc S3 core employs various techniques to mitigate thread hogs.

The Sparc S3 core uses low and high watermarks for many of the shared resources to prevent any given thread from becoming a thread hog. Low and high watermarks are established for each thread on the basis of the number of currently active threads on a core. Once a thread allocates enough entries of a shared resource to meet or exceed a high watermark, a stall is put into effect, disabling that thread from allocating any more of the respective shared resource. The thread remains stalled until it deallocates enough shared resources to reach a low watermark. Upon reaching the low watermark, the stall is removed and the thread is once again able to allocate shared resources as needed. The high and low watermarks used by the Sparc S3 core were determined through extensive simulation of commercial workloads. In general, different low and high watermarks exist for each shared resource, depending on a core's actual number of active threads.

On the Sparc S3 core, the PQ is one of the most critical shared resources with respect to thread hogging. The Sparc S3 core employs special techniques to manage the dynamic resource allocation of the PQ between threads. Threads in the PQ are expected to exhibit a certain rate of PQ deallocation. If threads don't deallocate PQ entries at or above a predefined rate of deallocation, then—by adjusting that thread's high and low PQ watermarks—the PQ resources available to them are reduced. If the thread continues to exhibit poor PQ deallocation rates over a long enough period of time, the Sparc S3 core considers the respective thread to be a *true thread hog*. A true thread hog is only allowed up to two instructions at a time—from the decode pipe stage through the pick pipe stage. If a thread hog begins to deallocate PQ entries at or above a predefined allocation rate, PQ resources are increased for that respective thread, which is no longer considered a true thread hog. By monitoring the rate of PQ resources' deallocation, the Sparc S3 core minimizes the effect of PQ thread hogs on the execution of other threads. The other threads are given more resources for execution, thus improving the core's overall throughput performance.

The use of low and high watermarks can't effectively manage thread hogs that occur due to certain long-latency events on the Sparc S3 core. For these long-latency events, Sparc S3 flushes the respective thread, thereby releasing all shared resources allocated to that thread. For example, when more than one thread is active, Sparc S3 flushes a thread on loads that miss the L3 cache. When the L3 load miss is detected, the respective thread is flushed and any shared resources being used by the thread being flushed are released to other threads for execution. The core also flushes the oldest load or store in the machine that doesn't commit for a predefined number of cycles (load with a RAW [read-after-write] hazard and a prior store miss is one example of this kind of event). Finally, the core flushes a thread that performs an I/O access. Once the I/O access reaches commit, the core flushes the thread and makes available any shared resources to other threads for execution. For these long-latency events, when an L3 miss or I/O access is completed, the core restarts the thread.

RAS features

Sparc T4 implements advanced RAS features aimed at identifying, containing, and recovering from soft errors. Major hardware structures are protected with error-correcting codes (ECC) or parity, as appropriate. Errors are detected and presented to the hypervisor, which logs the error. The hypervisor may take additional action to correct and clear an error. Table 1 summarizes some major Sparc S3 core components' RAS features. Recovery from detected soft errors is conducted via hardware or software instruction retry. Depending on the hardware structure, an error could affect either a single thread or

Table 1. Reliability and serviceability features of key hardware structures on the Sparc T4 processor core.

Hardware structure	Protection	Retry	Threads affected by single error
Instruction cache tag	Parity	Hardware	Multiple
Instruction cache data	Parity	Hardware	Multiple
Integer register file	Single error correct, double error detect (SEC/DED) error correcting code (ECC)	Software	Single
Floating-point register file	SEC/DED ECC	Software	Single
Working register file	Parity	Hardware	Multiple
Data cache tag	Parity	Hardware	Multiple
Data cache data	Parity	Hardware	Multiple
Store buffer data	SEC/DED ECC	Software	Multiple
Store buffer tag	Parity	Software	Multiple
Store queue data	Parity	Software	Single
Store queue tag	Parity	Software	Single
L2 cache data	SEC/DED ECC	Hardware	Multiple
L2 cache tag	Parity	Hardware	Multiple

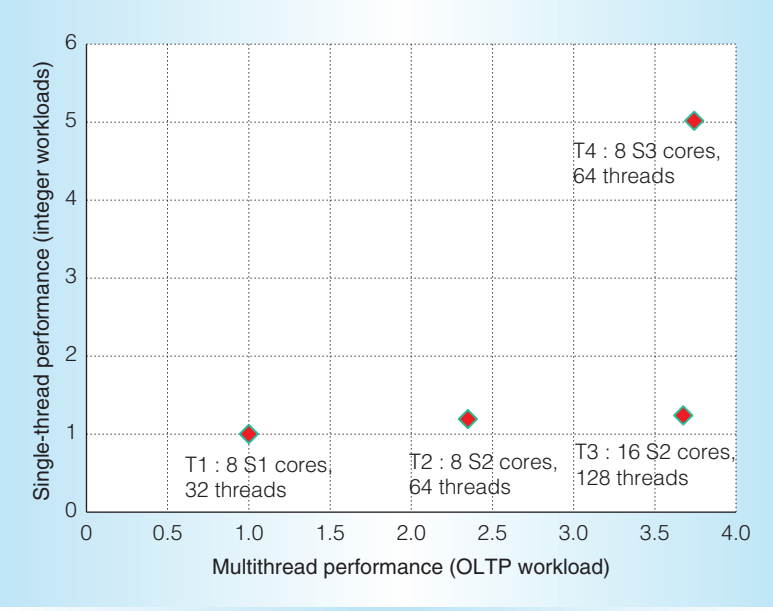

Figure 5. T4 relative performance. Sparc T4 delivers a 5× increase in single-thread integer performance and a 2× increase in per-thread throughput performance over previous Sparc T-Series systems.

multiple threads. Many other structures on Sparc T4, beyond those in Table 1, are ECC or parity protected.

If an error occurs, Sparc T4's CMT nature lets independent workloads continue operating unaffected until the error condition is logged and, if necessary, corrected and cleared. Its CMT nature also allows for threads on a Sparc S3 core to be deconfigured if the core encounters a hard fault that only affects one of its threads; the seven remaining threads can still be used. For example, a failing integer register file location affects only one thread. Similarly, a hard fault affecting only one core doesn't affect other cores' operation; the failing core can be deconfigured and the seven remaining cores can continue to be used. Thus, Sparc T4's highly threaded CMT nature enables graceful system degradation in the presence of hard errors or failing components, mitigating the urgency of a service action to replace a faulty processor.

Applications and results

Sparc T4's balanced single-threaded and multithreaded performance lets it perform well across a much broader set of workloads and enterprise applications than its predecessors—including web, database, and application servers; Java virtual machines; and enterprise applications, such as payroll, resource planning, and decision support.

Figure 5 shows Sparc T4's relative performance over Sparc T1 to T3 on single-thread

integer benchmarks and multithreaded online transaction processing (OLTP) benchmarks.

Sparc T4 has an estimated performance on SPECint2006* that is 5× that of Sparc T3. On SPECfp2006*, Sparc T4 has an estimated performance that is 7× that of Sparc T3.

Sparc T4 delivers improvements over Sparc T3 on various applications. Figure 6 depicts its performance improvement on the Oracle telco Billing and Revenue Management (BRM) benchmark, an Enterprise Java application benchmark, and Oracle's Siebel application.

Oracle has also published various industry-standard benchmarks, including Transaction Processing Performance Council (TPC-H). Oracle's Sparc T4-4 server, Oracle Solaris, and Oracle Database 11g Release 2 achieved a TPC-H benchmark performance result of 201,487 QphH@1,000 Gbytes with price/performance of $4.60/QphH@1,000 Gbytes. The Sparc T4-4 (four-processor) result is faster than eight-processor servers from other vendors.

For many users, the most important single-threaded applications are batch applications. Oracle's Sparc T4-4 server achieved world-record performance on the Unicode version of Oracle's PeopleSoft Enterprise Payroll batch (N.A) 9.1 extra-large-volume model benchmark. The Sparc T4-4 server Unicode result of 30.84 minutes on Payroll 9.1 is 2.8× faster than the IBM z10 EC 2097 Payroll 9.0 (Unicode version) result of 87.4 minutes. The IBM mainframe is rated at 6,512 million instructions per minute (MIPS).

Figure 7 shows Sparc T4's relative performance compared to other processors on a cryptography benchmark suite internally developed by Oracle. This test measures the maximum throughput of in-memory, on-chip encryption operations using Advanced Encryption Standard (AES) 256-bit key Cipher Block Chaining (CBC) mode encryption. Results are depicted in Mbytes per second.

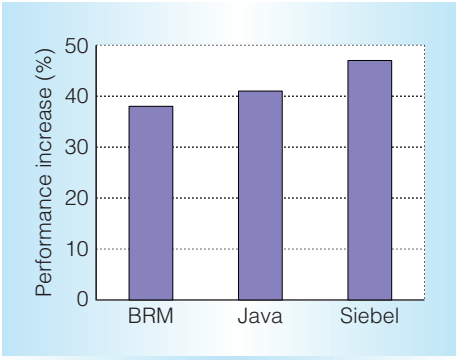

Figure 6. T4 relative throughput performance increase. On Billing and Revenue Management, Siebel Batch, and a Java application tier benchmarks, Sparc T4 has shown a 38 to 47 percent increase in throughput.

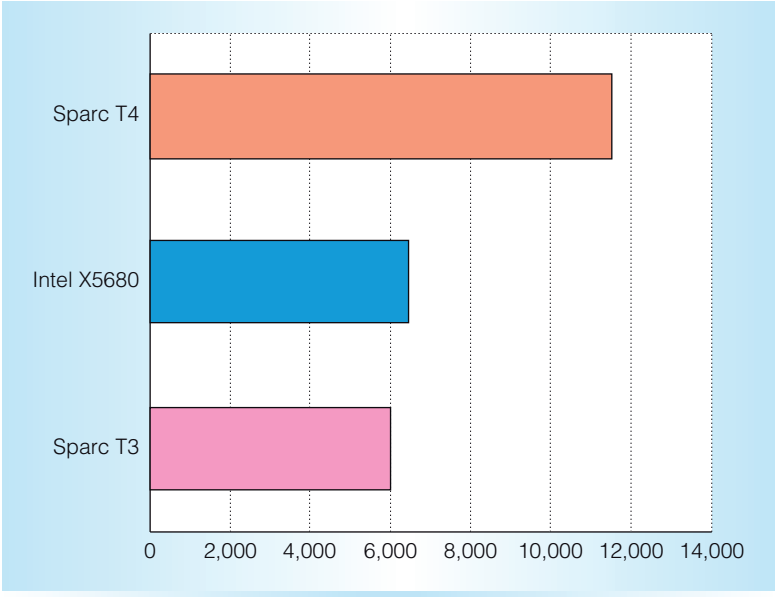

Figure 7. Sparc T4-2 comparison on cryptographic AES-CBC-256 cipher (Mbytes/s). T4 provides superior throughput performance on in-memory on-chip encryption operations.

Due to its flexible, dynamically threaded, out-of-order processor core, the Sparc T4 provides balanced single-threaded and multithreaded performance across a wide range of applications. Future work includes leveraging the T4 core across Oracle's Sparc processor roadmap, improving single-threaded and multithreaded processor performance, and exploring additional chip-level and system-level optimizations to improve the capabilities and performance of Oracle's software suite. MICRO

Acknowledgments

We acknowledge the contributions from the entire Sparc T4 CPU development

team at Oracle. *SPEC estimates are as of 18 Aug. 2011. Sparc T4-4 201,487 QphH@1,000 Gbytes, $4.60/QphH@1,000 Gbytes, available 30 Oct. 2011, four processors, 32 cores, 256 threads; IBM Power 780 QphH@1,000 Gbytes, 164,747.2 QphH@1,000 Gbytes, $6.85/QphH@1,000 Gbytes, available 31 Mar. 2011, eight processors, 32 cores, 128 threads.

References

1. P. Kongetira et al., ''Niagara: A 32-Way Multithreaded Sparc Processor,'' *IEEE Micro,* vol. 25, no. 2, 2005, pp. 21-29.
2. G. Grohoski, ''Niagara-2: A Highly Threaded Server-on-a-Chip,'' 18th Hot Chips Symp., 2006, http://www.openSparc.net/pubs/preszo/06/04-Sun-Golla.pdf.
3. M. Shah et al., ''UltraSparc T2: A Highly-Threaded, Power-Efficient, Sparc SoC,'' *Proc. IEEE Asian Solid-State Circuits Conf.,* IEEE CS Press, 2007, pp. 22-25.
4. S. Patel et al., ''Sun's Next-Generation Multithreaded Processor: Rainbow Falls,'' 21st Hot Chips Symp., 2009, http://www.openSparc.net/pubs/preszo/09/sunmicro_rainbowfalls_hotchips09.pdf.
5. D. Jimenez and C. Lin, ''Neural Methods for Dynamic Branch Prediction,'' *ACM Trans. Computer Systems,* vol. 20, no. 4, 2002, pp. 369-397.

Manish Shah is a senior principal engineer and member of the processor core development team at Oracle. His research interests include high performance computer architectures, branch prediction, and high-speed multithreaded processor designs. Shah has an MS in electrical engineering from Oklahoma State University.

Robert Golla is a senior hardware architect at Oracle, where he oversees the architecture of Oracle's processor cores. His research interests include high-performance computer architecture, microprocessor design, and throughput computing. Golla has an MS in electrical engineering from the University of Texas.

Greg Grohoski is a senior director at Oracle, where he oversees the development of Oracle's processor cores. His research interests include computer architecture and parallel computing. Grohoski has an MS in electrical engineering from the University of Illinois.

Paul Jordan is a senior principal hardware engineer and member of the core RTL development team at Oracle. His research interests include high-performance computer architectures and memory management. Jordan has a BS in electrical engineering from Rice University.

Jama Barreh is a principal engineer and member of the RTL development team at Oracle. His research interests include high-speed processor design and computer architecture. Barreh has an MS in electrical engineering from the Illinois Institute of Technology.

Jeff Brooks is a senior principal engineer and member of the processor core development team at Oracle. His research interests include microprocessor architecture and design and computer arithmetic. Brooks has a BS in electrical engineering from the University of Memphis.

Mark Greenberg is a principal engineer and member of the RTL development team at Oracle. His research interests include high-performance computer architectures, processor physical design, and simulation. Greenberg has a PhD in computer engineering from the University of Manchester, England.

Gideon Levinsky is a principal engineer and member of the RTL development team at Oracle. His research interests include microprocessor design, cache hierarchy, and computer architecture. Levinsky has an MS in electrical engineering from Stanford University.

Mark Luttrell is a senior principal engineer and member of the processor core design team at Oracle. His research interests include high-speed microprocessor design, cache hierarchy, and computer architecture. Luttrell has an MS in electrical engineering from Stanford University.

Christopher Olson is a principal engineer and member of the RTL development team at Oracle. His research interests include microprocessor architecture and design, computer arithmetic, and hardware cryptographic acceleration. Olson has a BS in electrical engineering from the University of Wisconsin.

Zeid Samoail is a senior hardware engineer and member of the RTL design team at Oracle. His research interests include high performance computer architecture. Samoail has a BS in computer engineering from the State University of New York at Stony Brook.

Matt Smittle is a principal engineer and member of the RTL development team at Oracle. His research interests include thread hog mitigation in multithreaded processor designs and power-aware designs. Smittle has an MS in electrical engineering from Southern Methodist University.

Tom Ziaja is a principal engineer and member of the RTL development team at Oracle. His research interests include Memory Test, Built-in Self Test, and DFT architecture. Ziaja has a PhD in computer engineering from the University of Texas.

Direct questions and comments about this article to Manish Shah, Oracle, 5300 Riata Park Ct., Austin, TX 78727; manish.s.shah@oracle.com.