RICHARD MCDOUGALL AND
JAMES LAUDON

multi-core microprocessors are here

Richard McDougall is a Distinguished Engineer at Sun Microsystems, specializing in operating systems technology and systems performance.

richard.mcdougall@sun.com

James Laudon is a Distinguished Engineer and a Niagara processor line architect at Sun Microsystems. His specialties are hardware multi-threading, multi-processing, and performance modeling.

james.laudon@sun.com

THE ADVENT OF SYMMETRIC MULTI-Processing (SMP) added a new degree of scalability to computer systems. Rather than deriving additional performance from an incrementally faster microprocessor, an SMP system leverages multiple processors to obtain large gains in total system performance. Parallelism in software allows multiple jobs to execute concurrently on the system, increasing system throughput accordingly. Given sufficient software parallelism, these systems have proved to scale to several hundred processors.

More recently, a similar phenomenon is occurring at the chip level. Rather than pursue diminishing returns by increasing individual processor performance, manufacturers are producing chips with multiple processor cores on a single die. For example, the AMD Opteron and UltraSPARC IV now provide two entire processor cores per die, providing almost double the performance of a single-core chip. The Sun UltraSPARC T1 (Niagara) processor packs eight cores onto a single die and can provide up to eight times the performance of the dual-core UltraSPARC processors.

There are three main types of multi-core processors:

- Simple multi-core processors have two complete processors placed on the same die or package (e.g., the dual-core AMD Opteron processor).
- Shared-cache multi-core processors consist of two complete cores, sharing some levels of cache, typically Level 2 (L2) or Level 3 (L3) (e.g., the Sun UltraSPARC IV+ and Intel Woodcrest processors, which share caches between two cores).
- Multi-threaded multi-core processors have multiple cores, with multiple threads within each core (e.g., the Sun UltraSPARC T1).

As processor frequency increases, the amount of time spent waiting for memory stalls increases. This means that placing multiple cores on a die can increase performance, but ultimately multi-threading in the CPU is critical to overcoming memory latency stalls. Implementations that use multi-cores *plus* hardware threading have recently proven to give superior performance at much lower power consumption.

These new multi-core processors are bringing what was once a large multiprocessor system

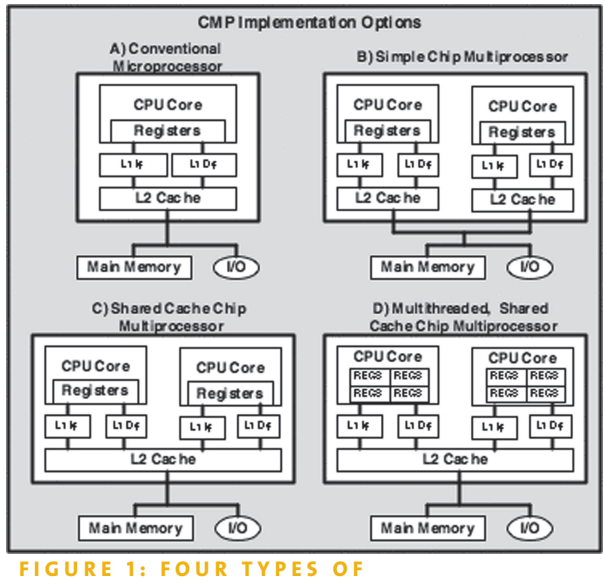

FIGURE 1: FOUR TYPES OF CHIP-LEVEL MULTIPROCESSING

down to the chip level, providing a significant level of throughput in a small package, with extremely low power consumption. In the case of the Sun UltraSPARC T1 processor with eight cores and four threads per core, power consumption of the chip is less than 70 watts. We have effectively reduced the equivalent throughput of a refrigerator-sized server (such as the Sun E6000 30-way, circa 2000) into a 1U single-processor machine, using less than 180 watts.

In this article, we'll contrast the different types of multi-core approaches and look at the performance advantages and tradeoffs. We will also discuss the potential implications for systems and application software.

Multi-Core Processors

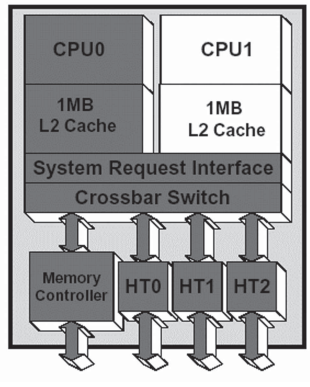

FIGURE 2: THE AMD OPTERON USES SIMPLE CHIP MULTI-PROCESSING, WITH THE TWO CORES SHARING ACCESS TO MEMORY AND TO OTHER BUSES (HYPERTRANSPORT)

The latest dual-core AMD Opteron is an example of a multi-core design. The chip has two complete processor cores, sharing a bus to memory. As shown in the left of Figure 2, it is almost identical to its single-core predecessor; the second core is a complete duplication of the first, including its pipeline and caches. From a performance perspective, the chip behaves much like a dual-processor SMP machine, albeit with some potential contention for memory bandwidth through the shared path to memory. From a software perspective, the chip appears almost indistinguishable from a dual-processor system. Software threads are scheduled onto the processor cores by the operating system—at least two threads are required to keep both cores busy.

Multi-Threading

Processor designers have found that since most microprocessors spend a significant amount of time idly waiting for memory, software parallelism can be leveraged to hide memory latency. Since memory stalls typically take on the order of 100 processor cycles, a processor pipeline is idle for a significant amount of time. Table 1 shows the amount of time spent waiting for memory in some typical applications, on 2 GHz processors. For example, we can see that for a workload such as a Web server, there are sufficient memory stalls such that the average number of machine cycles is 1.5—2.5 per instruction, resulting in the pipeline waiting for memory up to 50% of the time.

Waiting Application	*Typical Number of Cycles per Instruction*	*Percent of Time for Memory Stalls*
Transaction database	3–6	>75%
Web server	1.5–2.5	~50%
Decision support database	1–1.5	~10–50%

TABLE 1: EXAMPLES OF THE PERCENTAGE OF TIME SPENT WAITING FOR MEMORY ACCESS TO COMPLETE (STALLS OR WAIT STATES)

In Figure 3, we can see that less than 50% of the processor's pipeline is actually being used to process instructions; the remainder is spent waiting for memory. By providing additional sets of registers per processor pipeline, multiple software jobs can be multiplexed onto the pipeline, a technique known as simultaneous multi-threading (SMT). Threads are switched on to the pipeline when another blocks or waits on memory, thus allowing the pipeline to be utilized potentially to its maximum. Figure 4 shows an example with four threads per core. In each core, when a memo-

ry stall occurs, the pipeline switches to another thread, making good use of the pipeline while the previous memory stall is fulfilled. The tradeoff is latency for bandwidth; with enough threads, we can completely hide memory latency, provided there is enough memory bandwidth for the added requests. Successful SMT systems typically allow for very high memory bandwidth from DRAM, as part of their balanced architecture.

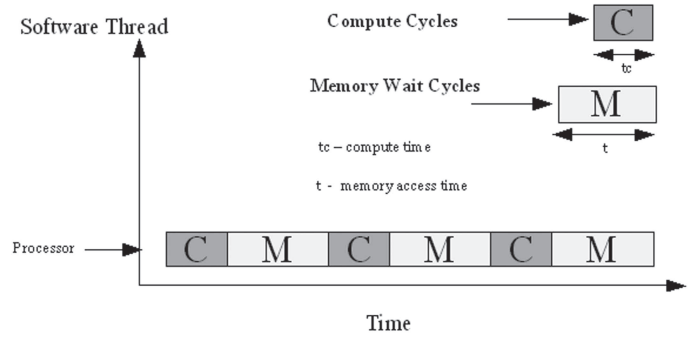

FIGURE 3: IN SINGLE-THREADED PROCESSOR DESIGNS, MORE TIME IS SPENT WAITING FOR MEMORY THAN ACTUALLY PROCESSING DATA

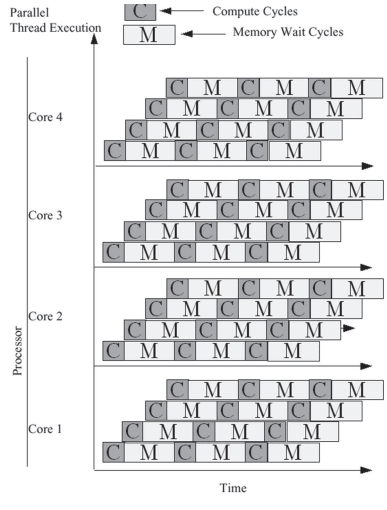

FIGURE 4: SIMULTANEOUS MUTLI-THREADING DESIGNS ALLOW THE PROCESSOR TO CONTINUE WITHOUT WAITING FOR MEMORY BY QUICKLY SWITCHING BETWEEN THREADS

SMT has a high return on performance in relation to additional transistor count. For example, a 50% performance gain may be realized by adding just 10% more transistors with an SMT approach, in contrast to making the pipeline more complex, which typically affords a 10% performance gain for a 100% increase in transistors. Also, implementing multi-core alone doesn't yield optimal performance—the best design is typically a balance of multi-core and SMT.

Contrasting the Different Types of Threading

There are three main ways to multi-thread a processor: *coarse-grain*, *vertical*, and *simultaneous* [1].

With coarse-grain threading, a single thread occupies the full resources of the processor until a long-latency event such as a primary cache miss is

encountered, as shown in Figure 3. At that point, the pipeline is flushed and another thread starts executing, using the full pipeline resources. When that new thread hits a long-latency event, it will yield the processor to either another thread (if more than two are implemented in hardware) or the first thread (assuming its long-latency event has been satisfied). Coarse-grain threading has the advantage that it is less of an integral part of the processor pipeline than either vertical or simultaneous multi-threading and can more easily be added to existing pipelines. However, coarse-grain threading has a big disadvantage: the high cost of switching between threads. When a long-latency event such as a cache miss is encountered, all the instructions in the pipeline behind the cache miss must be flushed from the pipeline and execution of the new thread starts filling the pipeline. Given the pipeline depth of modern processors—as many as 16 instructions in Intel-styled processors—this means a thread switch cost in the tens of processor cycles. This high switch cost means that coarse-grain threading cannot be used to hide the effects of short pipeline stalls owing to dependencies between instructions and even means that the thread-switching latency will occupy much of the latency of a primary cache miss/secondary cache hit. As a result, coarse-grain multi-threading has been primarily used when existing, single-threaded processor designs are extended to include multi-threading.

The two remaining techniques for threading, vertical threading (VT) and SMT, switch threads on a much finer granularity (and not surprisingly are referred to as fine-grained multi-threading). On a processor capable of multiple instruction issue, an SMT processor can issue instructions from multiple threads during the same cycle, whereas a VT processor limits itself to issuing instructions from only one thread each cycle (see Figure 4). On a single-issue processor there is no difference between VT and SMT, as only one instruction can be issued per cycle, but since there is no issue of instructions from different threads in the same cycle, single-issue, fine-grained multi-threaded processors are labeled VT.

Both SMT and VT solve the thread switch latency problem by making the thread switch decision part of the pipeline. The threading decision is folded in with the instruction issue logic. Since the issue logic is simply trying to fill the pipeline with instructions from all of the hardware threads, there is no penalty associated with switching between threads. However, a little extra complexity gets added to the issue logic, as it now needs to pick instructions from multiple ready threads. This additional issue logic complexity is fairly small (certainly much smaller than all the other issue-related complexity that is present in a modern superscalar processor) and well worth it in terms of performance. The advantages of SMT and VT are that very short pipeline latencies (all the way down to a single cycle) can be tolerated by executing instructions from other threads between the instructions with the pipeline dependency.

The ability to switch threads at no cost is the key to enabling the impressive performance of the new processors.

Chip-Level Multi-Threading

A new generation of processors that use Chip-Level Multi-Threading (CMT) combine multi-core with SMT, thereby providing a large core count and the ability to extract the maximum performance from each core. The UltraSPARC T1 is an example of a CMT processor design.

ULTRASPARC T1

Most people are familiar with the hyperthreaded Intel processors, which employ SMT. They support two threads in hardware and show modest gains on some parallel workloads. Given that SMT is the most aggressive of the three threading schemes, one would expect SMT to deliver the highest performance, but in general the performance gains seen from hyperthreading are small (and sometimes hyperthreading actually leads to performance losses). However, the gains seen from hyperthreading are not limited by the SMT but more by the memory system, and unfortunately the Intel hyperthreading implementation delivers a misleading message about the performance to be gained from fine-grained multi-threading.

The UltraSPARC T1, in contrast, was built from the ground up as a multi-threaded chip multiprocessor, and each of the eight pipelines employs vertical threading of four hardware threads. The eight pipelines in the UltraSPARC T1 are short (six stages), and one might be tempted to employ the slightly simpler coarse-grain threading. However, even on the UltraSPARC T1, the gains from vertical threading over coarse-grained multi-threading ended up being substantial. In fact, the very earliest proposals for what became the UltraSPARC T1 employed coarse-grain threading. Rather quickly, the modest additional complexity of vertical threading was traded off against its performance gains and the switch to vertical threading was made. The performance and performance/watt numbers from the UltraSPARC T1 show that it's been worth it!

The UltraSPARC T1 processor uses eight cores on a single die. Each core has four threads sharing a pipeline, an L1 instruction cache, a data cache, and a memory management unit (MMU).

The UltraSPARC T1 architecture has the following characteristics:

- Eight cores, or individual execution pipelines, per chip.
- Four hardware threads (strands) or active thread contexts that share a pipeline in each core. Each cycle of a different hardware strand is scheduled on the pipeline in round-robin order.
- A total of 32 threads per UltraSPARC T1 processor.
- A strand that stalls for any reason is switched out and its slot on the pipeline is given to the next strand automatically. The stalled strand is inserted back in the queue when the stall is complete.
- Cores that are connected by a high-speed, low-latency crossbar in silicon. An UltraSPARC T1 processor can be considered SMP on a chip.
- Hardware strands that are presented by the operating system as a processor. For example, Solaris and Linux see each thread as a separate processor.
- Cores that have an instruction cache, a data cache, an instruction translation-lookaside buffer (iTLB), and a data TLB (dTLB) shared by the four strands.
- Strands defined by a set of unique registers and logic to control state.
- A 12-way associative unified L2 on-chip cache. Each hardware strand shares the entire L2 cache. Historically, the level of associativity we typically see is around 4 for a non-CMT core, but with 32 strands sharing the L2, larger associativity is critical.
- Low-latency Double Data Rate 2 (DDR2) memory to reduce stalls. Four on-chip memory controllers provide high memory bandwidth (with a theoretical maximum of 25 gigabytes per second).
- An operating system scheduler that schedules LWPs on UltraSPARC T1 hardware strands. It is the task of the hardware to schedule strands in the core.

- A modular arithmetic unit (MAU) for each core that supports modular multiplication and exponentiation to help accelerate Secure Sockets Layer (SSL) processing.

The layout of a system implemented with the UltraSPARC T1 processor is shown in Figure 5 (which, for clarity, does not show the four memory controllers between the L2 cache and the four banks of SDRAM).

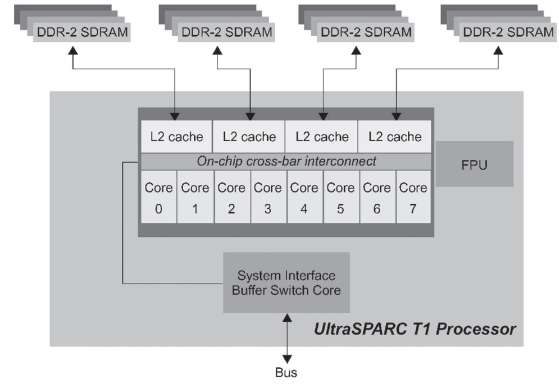

FIGURE 5: ULTRASPARC T1 CHIP LAYOUT

The Power Advantages of CMT

More complex pipelines use a significantly larger amount of power for little gain in performance. For example, when the clock rate of the Intel Celeron went from 1.2 GHz to the Pentium 4 at 2.2 GHz, power increased from 29 to 55 watts, but there was only a 20% performance improvement (measured using the Business Winstone Benchmark 2001).

Keeping the pipeline simple significantly reduces power consumption. In aggressive CMT architectures, such as the UltraSPARC T1, power per core is as low as 8 watts. This is achieved by keeping the pipeline simple, for example using a single-issue pipeline and eliminating many of the nonessential pipeline features, such as memory prefetching.

As an example, we can look at the throughput and power consumption of a typical Web workload, represented by the SPECweb2005 benchmark (see Table 2). By measuring the ratio of performance against watts consumed, we can contrast the performance/power efficiency of the system.

	UltraSPARC T1	*2x Dual Core @ 2.8 GHz*
Space (rack units)	1	2
Watts (system)	188	450
Performance	10,466	4850
Performance per watt	55.7	10.8

TABLE 2: ULTRASPARC T1 PERFORMANCE CHARACTERISTICS*

* From sun.com: "Sun Fire T1000 (8 cores, 1 chip) compared to Dell PowerEdge 2850 (4 cores, 2 chips). Results from www.spec.org as of May 30th 2006. Dell power measurements taken from the Dell Power Calculator, 03/06/06, posted: http://www1.us.dell.com/content/topics/topic.aspx/global/products/pedge/topics/en/config_calculator?c=us&cs=555&l=en&s=biz. System configured with 2 x Dual Core 2.8GHz processors, 16GB RAM, 2 x USCSI disks. Sun Fire T1000 server power consumption taken from measurements made during the benchmark run."

In this example, the CMT design provides roughly twice the throughput of the nonthreaded system, at half the power [2].

The Software View of CMT

A very simplistic view of a CMT system is that its software performance is much like that of an SMP system with the number of processors equal to

the number of strands in the chip, each with slightly reduced processing capability. Software threads are scheduled by the operating system onto individual hardware threads, and strands are scheduled onto the pipeline by a hardware scheduler. The number of software threads required to keep the core busy varies from one to many, depending on the ratio of memory stalls to compute events.

SINGLE-THREAD PERFORMANCE

Since each hardware thread is sharing the resources of a single processor core, each thread has some fraction of the core's overall performance. Thus, an eight-core chip with thirty-two hardware threads running at 1 GHz may be somewhat crudely approximated as an SMP system with thirty-two 250 MHz processors. Applications which are single-threaded will see lower performance than that of a processor with a more complex pipeline. The reduction in performance for single-threaded applications will depend on whether the application is more memory- or compute-bound, with compute-bound applications showing the largest difference in performance.

CMT LOVES "THROUGHPUT WORKLOADS"

To achieve per-thread performance with a significant increase of throughput and a reduction in power requires concurrency in the software. Applications that are server-oriented typically have bountiful amounts of concurrency. Typically these types of throughput applications are driven by a large number of connections or users, meaning there is enough natural concurrency to exploit SMP or CMT systems.

For a throughput-oriented workload with many concurrent requests (such as a Web server), the marginal increase in response time is virtually negligible, but the increase in system throughput is an order of magnitude over a non-CMT processor of the same clock speed.

A number of classes of applications benefit directly from the ability to scale throughput with CMT processors:

- Multi-threaded native applications: Multi-threaded applications are characterized by having a small number of highly threaded processes. Examples of threaded applications include Lotus Domino or Siebel CRM.
- Multi-process applications: Multi-process applications are characterized by the presence of many single-threaded processes. Examples of multi-process applications include the Oracle database, SAP, and PeopleSoft.
- Java applications: Java applications embrace threading in a fundamental way. Not only does the Java language greatly facilitate multi-threaded applications, but the Java Virtual Machine is a multi-threaded process that provides scheduling and memory management for Java applications. Java applications that can benefit directly from CMT resources include application servers such as Sun's Java Application Server, BEA's Weblogic, IBM's Websphere, and the open-source Tomcat application server. All applications that use a Java 2 Platform, Enterprise Edition (J2EE platform) application server can immediately benefit from CMT technology.
- Multi-instance applications: Even if an individual application does not scale to take advantage of a large number of threads, it is still possible to gain from CMT architecture by running multiple instances of the application in parallel. If multiple application instances require some

degree of isolation, virtualization technology (for the hardware of the operating system) can be used to provide each of them with its own separate and secure environment.

We've spent a great deal of time evaluating server application performance of CMT architectures; my blog [3] contains a good starting summary of the results we've had.

THOUGHTS ABOUT SOFTWARE SCALING

On a multi-threaded microprocessor, each hardware thread appears to the operating system as an individual processor. The ability of system and application software to exploit multiple processors or threads simultaneously is becoming more important than ever. As CMT hardware progresses, software is required to scale accordingly to fully exploit the parallelism of the chip.

Thus, bringing this degree of parallelism down to the chip level represents a significant change to the way we think about scaling. Since the cost of a CMT system is close to that of recent low-end uniprocessor systems, it's inevitable that even the cheapest desktops and servers will be highly threaded. Techniques used to scale application and system software on large enterprise-level SMP systems will now frequently be leveraged to provide scalability even for single-chip systems. We need to consider the effects of the change in the degree of scaling at the low end on the way we design applications, on which operating system we choose, and on the techniques we use to deploy applications.

Conclusion

In today's data centers, power and space are valuable resources. The advantages brought about by CMT are inevitable for optimizing these resources. The aggressiveness of CMT varies with different system designs; we expect to see four-core systems from AMD in the near future, UltraSPARC follow-ons are expected to increase the thread count, and Intel is discussing some radical multi-core designs. The interesting debate will be about the number of cores to have and to what degree each approach will utilize vertical threading within each core to hide memory latency. It's going to be a fun time in this space. Stay tuned!

ACKNOWLEDGMENTS

Thanks are owed to Denis Sheahan, Performance Specialist in the UltraSPARC T1 group for the UltraSPARC T1 specifications, and the PAE performance group at Sun Microsystems for providing the performance characterization data of workloads on CMT.

REFERENCES

[1] Jim Laudon's blog: http://blogs.sun.com/jlaudon.

[2] T1000 Server Benchmarks: http://www.sun.com/servers/coolthreads/t1000/benchmarks.jsp.

[3] Richard's Blog: http://blogs.sun.com/rmc.