Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

NIAGARA 2 OPENS THE FLOODGATES

Niagara 2 Design Is Closest Thing Yet to a True Server on a Chip

By Harlan McGhan {11/6/06-01}

At the recent **Fall Microprocessor Forum**, Sun Microsystems presented its new Niagara 2 microprocessor design, the successor to Niagara 1. Because Niagara 1 processors ship under the product name UltraSPARC T1, Niagara 2 will presumably go to market next year as the UltraSPARC T2. Sun's presentation at MPF was delivered by Robert Golla, Niagara 2 principal architect.

Sun's Niagara-series processors are, unquestionably, the odd duck among today's server processors. Rival server-processor designs differ from each other in degree. How much advantage does the true simultaneous multithreading (SMT) capability of dual-core, dual-threaded POWER6 processors provide over the coarse-grained or vertical multithreading (VMT) capability of dual-core, dual-threaded SPARC64 VI processors? Is the compiler-scheduled, static in-order wide-superscalar issue of VLIW-style Itanium processors better than the hardware-driven, dynamic out-of-order wide-superscalar issue of RISC processors? What benefits does HyperTransport (HT) technology confer on Opteron processors over the traditional front-side bus used by Xeon processors?

These are not the sort of questions to ask about Niagara processors. Niagara differs from other high-end server processors not merely in degree but also in kind. In a way, they are most readily understood by those who have spent the past two decades oblivious to the forces that have caused processor cores to evolve from simple, low-frequency, in-order scalar designs to complex, high-frequency, out-of-order superscalar designs—or by those who have taken the lessons of this progression most deeply to heart. Niagara processors are predicated on the conviction that just about everything in contemporary server-processor design is wrong, and that the best way forward is to revert to a far simpler design era. (See *MPR 11/17/03-03*, "Will Microprocessors Become Simpler?")

The Origin of Chip Multithreading

Niagara is by no means the first radical processor design ever to reach market. To the contrary, the progression of this latest proposed design revolution traces a familiar arc. It began as an academic research project. Once the feasibility of the notion was established, it was implemented by a startup specifically created by its academic founder to give commercial expression to the new idea. The startup was later acquired by an established system company that adopted the new idea for its own and played a key role in developing and promoting the technology in the marketplace.

During the 1980s, this path was trod by Professor John Hennessy's 1981–83 MIPS reduced instruction set computing (RISC) project at Stanford University, the 1984 startup Hennessy founded (MIPS Technologies), and Silicon Graphics (later SGI)—once, the principal supporter and defender of high-end MIPS RISC architecture processors. It also was traced by Josh Fisher's early 1980s Trace Scheduling project at Yale University, which led to the notion of very long instruction word (VLIW) hardware; the 1984 startup that Fisher founded (Multiflow); and Hewlett-Packard—which later teamed with Intel to bring the descendant VLIW-style Itanium processor family to market.

In the case of Niagara, the progression begins with Professor Kunle Olukotun's late-1990s Hydra chip multithreading

(CMT) project at Stanford, the 2000 startup Olukotun founded (Afara Websystems), and Sun Microsystems—which acquired Afara in July 2002 and publicized the commercial derivative of the Hydra design under the Niagara code-name. For an account of multithreading strategies and their associated terminology, see the sidebar "Multithreading Strategies in Server Processors."

In a sense, "Niagara 2" is a generational misnomer. In fact, Niagara 2 closely resembles the original vision of a commercial implementation of Hydra at Afara Websystems. Niagara 1 was a cut-back version of this design, initiated at Sun shortly after it acquired Afara, in order to fit the design onto a manufacturable die using TI's 90nm process technology and to get the product to market as rapidly as possible. The full realization of the initial Afara plan was postponed to the Niagara 2 generation. The additional development time afforded to a second-generation design, and the more ample transistor budgets of 65nm process technology, makes the Niagara 2 implementation commercially feasible.

Because of this relationship between the first two generations of the Niagara design—where Niagara 1 is, in effect, Niagara Jr.—Niagara 1 furnishes a convenient starting point for understanding Niagara 2.

Niagara 1 Core Overview

At the core level, Niagara 1 is a design not seen in commercial high-end desktop or server processors since the late 1980s. It is an integer-only core with an attached (shared) FPU. It revives the basic five-stage pipeline used by early RISC implementations: instruction fetch, decode, execution, memory access (for load/store instructions), and write-back. However, it does insert one extra stage—thread select—between instruction fetch and decode. Figure 1 shows the Niagara 1 core pipeline.

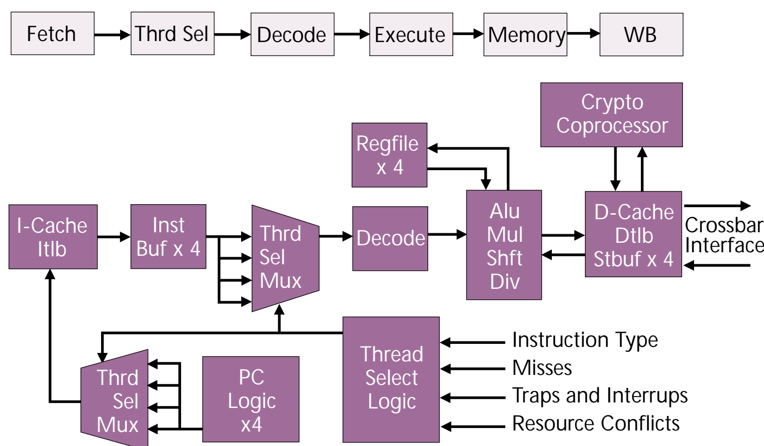

Figure 1. Niagara 1 core pipeline. This unit handles basic integer ALU operations plus shifts and integer multiply and divide. The only features that distinguish it from a basic 1980s scalar RISC pipeline are the thread-select stage and the cryptographic coprocessor. The cryptographic coprocessor shares the crossbar interface with the L1 D-cache but does not modify its contents, streaming data directly out of and back into the L2 cache.

The Niagara 1 core fetches one or two instructions from a thread out of the instruction cache during a clock cycle (depending on whether the fetch address is odd or even). The first instruction fetched proceeds down the pipeline, flowing through to the decode stage in the next clock cycle. When two instructions are fetched from an even boundary, the second "odd" instruction is held in a one-line instruction buffer (next-instruction flop) at the beginning of the select stage for later issue. Thus, although the Niagara 1 core sometimes fetches multiple instructions, it always issues just one instruction down the pipeline, like any simple scalar design.

Although the Niagara 1 core executes just one instruction per clock cycle, it manages four separate execution threads simultaneously. As Figure 1 indicates, each thread has its own program counter (PC), one-line instruction buffer (for holding "odd" instructions), and register file. The four threads share the instruction cache, the data cache, and the various execution resources in the core.

Niagara 1 processors implement a fine-grained multithreading scheme. The thread-select logic implements a least recently used (LRU) algorithm for picking from among the available threads. When all four threads are available, it simply plays round-robin on a clock-by-clock basis, fetching an instruction for thread 0 from the I-cache on clock 0 and issuing it to the decoder; fetching and issuing an instruction for thread 1 on clock 1; for thread 2 on clock 2; for thread 3 on clock 3; before starting over again with thread 0 on clock 4. If a thread stalls for any reason, the thread-select logic simply drops it from the round-robin until it is ready to resume execution, fetching and issuing instructions from the next executable thread in its place.

Effectively, the individual threads dynamically speed up and slow down as they run. At a processor frequency of 1.2GHz, the four executable threads each run at a speed of 300MHz. If one thread drops out, the remaining three threads run at 400MHz; or, if two threads drop out, at 600MHz. If all but one of the threads stall, the last remaining executable thread runs at the full 1.2GHz processor speed—until one or more of the stalled threads joins back in, slowing the individually executing threads down again in proportion to the number running at the same time.

For the most part, Niagara simply ignores the problems that other contemporary processors go to great lengths to minimize. Multithreading allows the core to stay busy without the complications of out-of-order execution. Rather than attempting to predict branches (with imperfect success), or predicate them (at great expense), Niagara simply drops a thread issuing a branch out of the rotation until its branch condition is resolved. And although Niagara is willing to speculate that loads hit the L1 cache, it

also lowers the priority of speculatively issued instructions. The result is that speculations issue only as a last resort, when the core has run out of more-certain instructions to decode. Ideally, before the core gets around to issuing a load-dependent instruction speculatively, the load will either have returned its data or missed the cache—either way, entirely removing the element of speculation from any dependent instructions.

The other feature (besides thread select) that distinguishes the Niagara 1 core from scalar 1980s RISC designs is an asynchronously operating cryptographic coprocessor. This supports public key RSA and (almost identical) DSA encryption/decryption for up to 2,048-bit keys. The unit shares the integer multiplier for modular arithmetic operations. Only one thread can use the crypto unit at a time. The thread sets up the operation by storing a value to a control register, after which it returns to normal processing. Triggered by the control register, the crypto coprocessor then initiates a streaming load/store directly into/out of the L2 data cache (bypassing L1). The streaming crypto operation completes either through polling or because of an interrupt.

Multithreading Improves Throughput

The advantage of Niagara 1's multithreading over a simple RISC pipeline may not be immediately apparent. Although Niagara's fine-grained multithreading handles four threads at one-quarter the processor clock speed, exactly the same amount of work would be accomplished by processing one thread at the processor's full clock speed. It's natural to wonder why Sun makes the substantial investment needed to implement fine-grained multithreading on Niagara processors.

The answer is that, even on a simple scalar core, few threads can sustain an execution rate of four instructions every four clock cycles for very long. Threads are constantly slowed by long-latency operations, including branches, loads, and inherently multicycle operations like multiply or divide. They stall altogether when loads miss the L1 cache or (far worse) the L2 cache, or when necessary resources are unavailable, or when a trap interrupts normal processing, e.g., due to an error or the need to handle a real-time event.

In Niagara, as threads slow or stall for any reason, they simply drop out of the rotation until ready to resume execution. The remaining threads speed up to fill the vacated decode slots. The LRU-selection algorithm automatically gives a stalled thread higher priority when it is ready to restart.

Even though Niagara 1 executes only one instruction from one thread per clock cycle, when the inevitable stalls are considered, it runs four threads concurrently just as fast as they could run in a simple, scalar pipeline. As a rule, Niagara threads do not starve for execution resources. To the contrary, by oversubscribing its core execution resources by a factor of four, Niagara comes much closer to (but still does not quite reach) the ideal RISC goal of executing one instruction every clock cycle. Figure 2 shows an analysis of core usage based on the SPECjbb benchmark program.

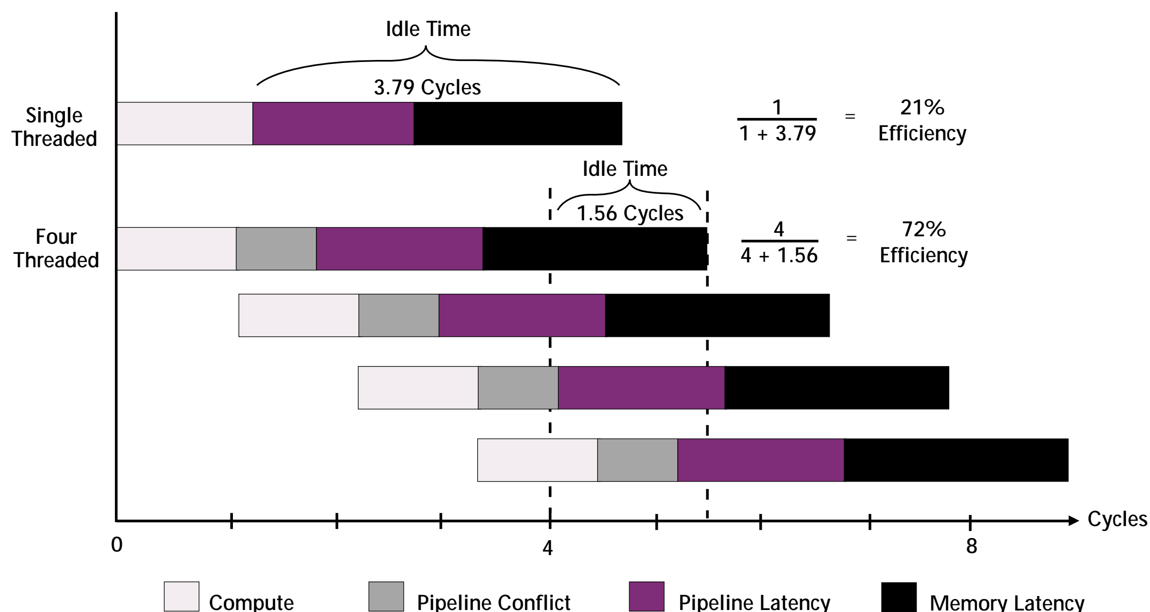

Figure 2. Here is a measured example of Niagara 1's execution efficiency, drawn from an analysis of the SPECjbb benchmark program that Sun provided at the 2006 International Solid-State Circuits Conference (ISSCC). Although the four-way multithreaded Niagara 1 core is not perfectly efficient, it is far more efficient than any single-threaded core. The idleness of nearly 80% shown for a single-threaded core on this benchmark is shocking in its own right, but it is only the tip of the iceberg of inefficiency where superscalar designs are concerned. For a superscalar processor, most busy cycles leave some execution slots unfilled, while every idle cycle leaves all its multiple execution slots unfilled. Source: A. S. Leon et al., "A Power-Efficient High Throughput 32-Thread SPARC Processor," ISSCC06, Paper 5.1

When running this program, a single-threaded version of the Niagara core would sit idle for 3.79 cycles for every one cycle spent executing code. That's an efficiency rating of just under 21%. The core spends most of its idle time (shown in black) waiting for memory. The remainder (shown in purple) is caused by pipeline bubbles—by branch penalties, load-use penalties, and so on. With a program of this type, a single-threaded processor core actually does nothing for nearly 80% of the time it is (supposedly) running.

Niagara's four-way multithreading introduces a small amount of additional overhead into the execution process (shown in gray)—and clearly does nothing to reduce either memory or pipeline latency. However, by switching to a different thread every clock cycle, Niagara manages to recover not only all the added overhead but most of the idle cycles, too. Efficiency rises dramatically in this four-thread design, with the core gainfully employed more than 70% of the time. In other words, rather than wasting four out of every five clock cycles, the Niagara four-way, fine-grained multithreading strategy effects a dramatic turnaround, putting nearly three out of every four clock cycles to productive use.

It is no exaggeration to say that Niagara processors seek to restore, first and foremost, execution efficiency to processor designs. This necessarily involves turning away from thread speed as a metric of excellence, since thread speed is invariably improved by extravagance. Adding extra issue slots to a superscalar design will improve a processor's speed—even if most software uses the extra slots only rarely. Adding a few hundred million extra transistors to a cache will improve a processor's speed—even if the hit rate of most programs increases only slightly. Adding another gigahertz to the clock frequency will improve a processor's speed—even if the extra cycles are largely left idle.

As long as CPU architects evaluate a design change only by the incremental improvement in performance resulting from it—and as long as they hide or ignore the costs of that improvement (in design time, transistors, power consumption, and heat)—the speed advantage will always go to the designs that consume the most. The winning designs will risk the widest instruction issue, the highest clock frequencies, the largest transistor counts, and the greatest power consumption.

Niagara processors are not about raw speed. Sun's goal is to make all the investments in the Niagara pipeline—from designing it on the drawing board to cooling it in server farms—pay off, by the simple expedient of keeping it productively employed for more clock cycles. This is not a goal that benefits from extravagance of any kind, including superscalar instruction issue, excessive cache sizes, and high clock frequencies.

This kind of execution efficiency depends on high throughput performance. Niagara cores churn out (close to) one completed instruction for every clock cycle by executing multiple threads in tight sequence, one instruction at a time. We must judge Niagara not by the amount of performance it delivers per thread, but rather by the amount of throughput it delivers per watt of expended energy or per unit of rack space.

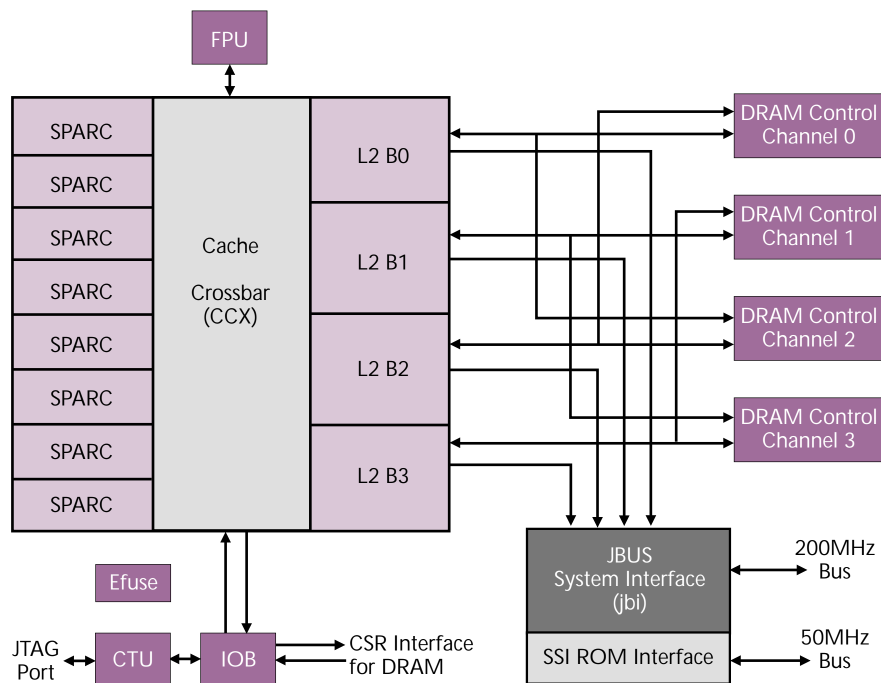

Figure 3. Niagara 1 block diagram. The small (11mm²) four-way threaded core is replicated eight times and integrated with a four-banked 3MB L2 cache, one FPU, four DDR2 memory controllers, a Sun JBus system interface (originally developed for the UltraSPARC IIIi), an SSI interface, and some built-in test, debug, and bring-up logic. A high-speed crossbar connects the cores to the FPU and L2 cache.

Niagara 1 Processor Design

Zooming back from the core level, Niagara processors emphasize high integration. Figure 3 is a block diagram of the Niagara 1 processor.

Just as Niagara's performance goal challenges traditional speed metrics, the overall chip design also rethinks some standard assumptions. Each core supports four threads with 16KB of I-cache and 8KB of D-cache. The whole chip supports 32 threads of execution with just 3MB of L2 cache. In applications where most or all threads share data (e.g., transactional processing workloads), a 3MB cache is sufficiently large that enlarging it does not substantially increase the hit rate. In other types of applications, even assuming some threads are sharing data, with each thread getting by on

just 4KB of L1 I-cache, 2KB of L1 D-cache, and less than 100KB of L2 cache, hit rates will be relatively poor.

But the function of on-chip caches in Niagara is not to maximize hit rates and thereby minimize memory latencies. Memory latencies are less critical to Niagara performance than for other processors, because Niagara is not striving for high thread speed. As long as the core can keep busy executing some threads, it does not greatly matter if other threads are suspended waiting on a memory access more or less often. In a sense, Niagara depends on memory latencies, exploiting them almost as a precious natural resource that opens the core for use by other threads.

The heart of Niagara processing performance really lies in its top-to-bottom memory bandwidth, starting with the four integrated DDR2 memory controllers. Operating at 400MHz (2 × 200MHz), these four 144-bit channels (16 bytes plus 16 parity bits) have the collective ability to transfer 25.6GB/s to and from the processor. To support its 32 executing threads, Niagara wants to exercise these links as heavily as possible. The on-chip caches are merely oversize buffers whose chief function is to keep the memory links from saturating, thereby capping the chip's effectiveness.

The on-chip bandwidth supplied by the crossbar that connects the eight cores to the four L2 banks (and the attached FPU) is even more phenomenal, supporting 134GB/s of traffic. The L2 cache is fully shared: any core can access any bank with about the same latency. It is this whole memory subsystem, with its associated bandwidths, that drives the throughput performance of Niagara.

The Sun-specific 3.2GB/s JBus interface is usually connected to an external PCI-X/PCIe bridge chip to provide high-speed I/O access to the processor. Cramming all this functionality on a die in TI's nine-layer copper-metal 90nm technology requires 279 million transistors and occupies 378mm^2 of silicon. At its maximum frequency of 1.2GHz, Niagara 1 dissipates around 70W, or just a bit more than 2W per thread.

Limitations of the Niagara 1 Design

The single shared FPU, attached to the cores through the crossbar switch, is an obvious weakness and potential bottleneck in the initial Niagara design. First, simply reaching the FPU with an instruction is a long-latency operation (about 30 clock cycles). Second, the FPU's capabilities are limited. For example, it supports just a subset of the Visual Instruction Set (VIS) extensions built into every UltraSPARC processor since the UltraSPARC I generation. (See *MPR 12/5/94-04*, "UltraSPARC Adds Multimedia Instructions.") Third, if 32 threads all start banging on the lone FPU at once, it very quickly becomes a bottleneck, choking the performance of the whole chip.

Niagara 1 can handle an occasional floating-point instruction, but if the software contains more than a percentage point or two of floating-point operations, then Niagara 1 is a poor choice. In fact, it should be entirely struck off the list of candidate processors.

Two other limitations of Niagara 1 are that software should be heavily threaded and none of the threads should be speed-critical. Niagara 1 gambles on finding four threads per core and 32 threads per processor. Moreover, the software must be able to tolerate the scalar thread-execution speed of one-quarter Niagara's relatively low clock frequency. And the order in which threads complete must be irrelevant, because the processor can't guarantee the order of their completion.

Niagara 1 is extremely efficient when running heavily threaded code at relatively low thread speeds, but its efficiency drops sharply as threads become scarce. Other processors generally ignore more than two or four threads at one time, even when more threads are available. Limited to a single thread, they happily cope with their abundant speed features, including superscalar instruction issue, out-of-order execution, branch prediction, hardware prefetching, speculative loads, and so on.

Niagara has none of these mechanisms. Its only defense against enforced idleness is multithreading. Take that away and it is defenseless. Largely for this reason, Niagara processors are optimized to run unmodified SPARC V9 code under Solaris—perhaps the largest, extensively threaded code base in existence.

The Niagara 2 Core Design

Almost everything said in general about Niagara processors applies to the Niagara 2 design. Niagara 2 differs from Niagara 1 primarily by hewing more closely to Afara's original vision of a true server on a chip. Niagara 2 also remedies the most glaring (addressable) deficiency of Niagara 1—its poor floating-point performance.

Niagara 2 still has eight cores, but the basic core design is upgraded in three major, and several minor, respects. The most significant change is the addition of a second four-threaded execution pipeline. Each Niagara 2 core executes eight threads at a time, not just four. Since the number of cores is the same, this means each 8-core × 8-thread Niagara 2 chip executes 64 threads, not 32 threads like the 8-core × 4-thread Niagara 1.

Because the second set of four threads per core executes in its own new pipeline, there is no additional contention for execution resources among the eight threads within a core. Admittedly, the additional four threads double the stress on the shared load-store unit and the crypto coprocessor. They also double the burden placed on the L1 caches, since these remain the same size. However, since Niagara processors depend on high memory bandwidth to achieve high throughput performance, this added pressure largely translates into pressure on the whole memory subsystem—which has been upgraded to support the doubling of threads.

The second important change is the addition of a floating-point/graphics unit (FGU) to each core, making eight FGUs per chip (each shared by eight threads). Even serially dependent floating-point operations from all eight

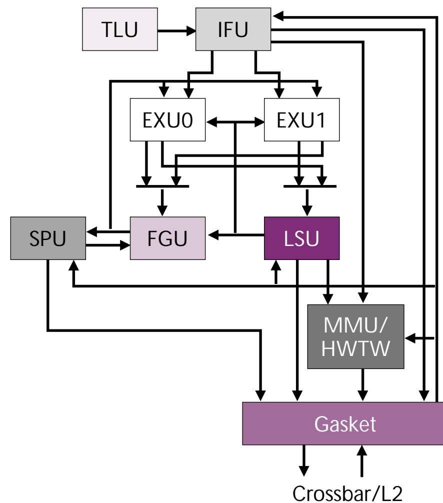

Figure 4. In Niagara 2 cores, the most visible changes are the dual execution pipelines (EXU0 and EXU1), each supporting four threads, and the new FGU. Other units were present in the Niagara 1 core. TLU is the trap logic unit (for interrupt and trap handling); IFU is the instruction-fetch unit (which includes the 16KB I-cache); SPU is the stream processing or crypto unit (which now shares resources with the FGU); and LSU is the load/store unit (which includes the 8KB D-cache). The memory management unit (MMU) includes hardware table walk (HWTW) for page sizes ranging from 8KB to 256MB. The gasket is the interface to the crossbar, which handles all traffic in and out of the core.

threads can run interleaved on the FGU, so Niagara 2 offers far more floating-point throughput than Niagara 1.

Moreover, each FGU is more functional than the single shared FPU of Niagara 1. As its name suggests, it provides full support for the current UltraSPARC 2.0 VIS extensions. While the eight new FGUs don't turn Niagara 2 into a floating-point monster, they do remove the severe limitations of Niagara 1 processors when running floating-point code of any consequence. Because Niagara 2 doesn't emphasize thread speed much more than Niagara 1 does, in suitable applications, Niagara 2 systems should run floating-point code well enough for most practical purposes.

The third important change is a significant upgrade of the in-core, asynchronous cryptographic coprocessor. In Niagara 1, the crypto unit handled basic RSA/DSA public-key ciphers. The upgraded crypto unit in Niagara 2 handles just about any cipher that might be of interest, including RC4, AES, DES, and 3DES. It also handles MD-5 and SHA-1/256 hashes. Moreover, running at full core speed, it is designed to keep up with the two new, integrated 10Gb Ethernet ports, allowing encryption/decryption of packets to keep pace with the wire-speed flow of traffic off the network into memory, and out of memory into the network. The DMA engine in the crypto unit shares the core's crossbar port. Figure 4 shows the new Niagara 2 core design.

Note that in contrast to high-performance, speed-oriented core designs that typically fan out into multiple parallel integer, floating-point, and load/store execution pipelines, Niagara 2 cores funnel all instructions through their two integer pipes. The two integer execution units pass load/store instructions to the shared LSU and pass floating-point and VIS instructions to the shared FGU. The asynchronous integrated SPU (crypto unit) operates in essentially the same way as before (set up by load/store instructions), except that it now shares resources with the integrated FGU rather than with the integer multiplier.

Among the other changes, the most significant is expansion of the next-instruction flops (used to hold the "odd" instruction on double fetches from even addresses) into full eight-entry instruction buffers, one for each thread. In Niagara 1, the (first) fetched instruction always flows down the pipeline. In Niagara 2, the new eight-entry instruction buffers decouple the fetch and pick stages of the pipeline.

More minor changes include a deeper integer pipeline. The pipeline needs an extra stage (labeled "pick") to divide the execution stream into two four-thread groups of instructions. The data TLB (translation lookaside buffer) is doubled from 64 fully associative entries to 128 fully associative entries. The I-cache is bumped up from four-way set associative to eight-way. Figure 5 shows the lengthened 8-stage integer pipeline and the new 12-stage floating-point pipeline.

The Niagara 2 Processor Design

Although zooming back from the core design to look at the overall processor does not show a greatly changed picture, it does reveal a greatly upgraded chip design. The eight cores now connect through the crossbar switch to eight banks of 16-way set-associative L2 cache, totaling 4MB (rather than four banks of 12-way set-associative L2 cache, totaling 3MB).

8 Stage Integer Pipeline

Fetch	Cache	Pick	Decode	Execute	Mem	Bypass	W

12 Stage Integer Pipeline

Fetch	Cache	Pick	Decode	Execute	Fx1	Fx2	Fx3	Fx4	Fx5	FB	FW

Figure 5. Niagara 2 execution pipelines. The new Pick stage supports the eight threads each core concurrently executes. The bypass stage was in Niagara 1 but was folded into the write-back stage and not explicitly labeled in earlier pipeline diagrams. The integer pipe has a three-cycle load-use penalty. The latency for dependent FP operations is six cycles. The 12-stage floating-point pipeline is for add, subtract, and multiply operations; divide and square-root operations need additional execution stages.

Because Niagara 2 doubles the number of threads per chip but increases the L2 cache by only 33%, it puts even more strain on memory bandwidth in workloads without high data sharing. Consequently, Niagara 2 must achieve higher average transfer rates than Niagara 1. To help cope with this challenge, the four DDR2 channels in the previous generation have been replaced with four dual FBDIMM DRAM channels. These burn a few watts more than DDR2 channels but require fewer pins and offer significantly better memory performance.

Of course, with an FGU integrated into every core, the FPU attached to the crossbar is gone in Niagara 2. Instead, the I/O port in the crossbar connects to a new system interface unit (SIU). This, in turn, supports an integrated x8 PCI Express port (formerly connected externally to the integrated JBus controller). There are also two new 10/1Gb Ethernet ports, enabling network packets to flow directly through the processor without needing external Ethernet controllers. The integrated crypto units enable secure network processing.

With a few exceptions in terms of interesting detail—including transistor count, power dissipation, confirmed memory bandwidth, and reported benchmark performance—Sun has now well-disclosed the Niagara 2 design. Table 1 compares the two generations of Niagara, and Figure 6 compares the two dies.

UltraSPARC Generation	Niagara 2 (US T2)	Niagara 1 (US T1)
Process Technology	TI 65nm	TI 90nm, 9LM
Date Introduced	3Q07*	4Q05
Frequency (MHz)	1,400*	1,200
Transistors (millions)	N/A	279
Die Area (mm^2)	342	378
Power Dissipation (watts)	N/A	~70
Cores/Die	8	8
Threads/Core	8	4
Integer Execution Units	16	8
Floating-point Execution Units	8	1
Load/Store Units	8	8
Cryptography Units	8 large	8 small
Instructions/Thread	1	1
Instructions/Clock	16	8
Peak mips	22,400	9,600
L1 I-cache	16KB, 8-way, 32B line	16KB, 4-way, 32B line
I-TLB	64 entry, full assoc	64 entry, full assoc
L1 D-cache	8KB, 4-way, 16B line	8KB, 4-way, 16B line
D-TLB	128 entry, full assoc	64 entry, full assoc
Instruction buffers	4 "odd" instruction flops	8 8-entry instruction buffers
L2 Cache	4MB, 16-way, 64B line	3MB, 12-way, 64B line
L2 Cache Banks	8	4
Crossbar Bandwidth (GB/s)	180 read + 90 write	134.4
Memory Controllers	4 x FB-DRAM	4 x DDR2
Memory Speed	FBD-667*	2 x 200MHz
Memory Bus Width (bits)	4 x 144**	4 x 144**
Memory Bandwidth (GB/s)	42.7*	25.6
Memory Size (GB)	128	128
Pipeline Stages	8 int, 12 fp	6 int
Integrated I/O	PCI Express X8@2.5 GB/s	JBus@3.2 GB/s
Integrated Networking	2 x 10/1 Gb/s Ethernet	none
Total Pins	1,831	1,933

Table 1. Comparison of the first two generations of Niagara processor design. The most important changes are doubling the threads per core and per chip; the small frequency bump; the great improvement in floating-point performance; the upgraded crypto units; the new higher-bandwidth FB DRAM channels; and the integrated PCI Express and Ethernet interfaces. **MPR* estimate. **16 bytes + 16 parity bits

Conclusion: A Significant Upgrade

Niagara 2 is the fully embodied vision of a throughput-oriented, highly efficient server-on-a-chip that the Afara team brought to Sun in July 2002. By leveraging Sun's design and financial resources, as well as Sun's long-standing semiconductor partnership with TI and its 65nm technology, this vision now exists in working silicon. It is expected to reach systems during 3Q07. Although Sun has not yet released full information about Niagara 2, we have enough detail to evaluate the design.

The throughput performance of Niagara 2 at its (presumed) target frequency of 1.4GHz should be more than twice the performance of the former design. Expect much more improvement with selected tasks that take advantage of the new floating-point capabilities, the new wire-speed cryptography, and the new integrated networking interfaces. The integrated PCI Express interface will improve I/O performance. Sun estimates that both raw throughput and throughput per watt will more than double; that single-thread integer performance will increase by 40%; and that floating-point throughput will improve by more than an order of magnitude.

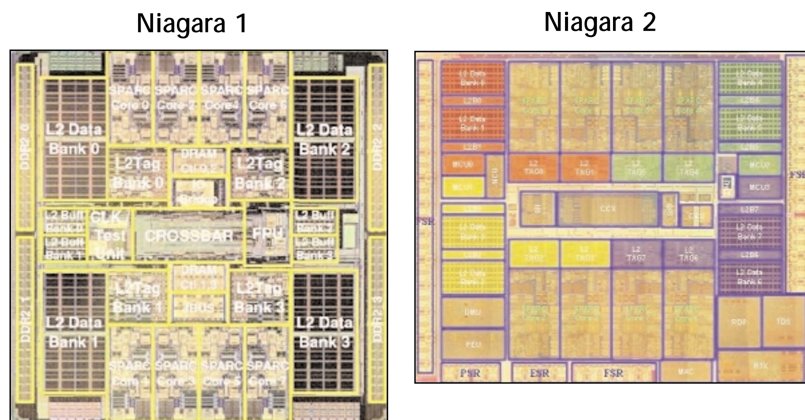

Figure 6. The Niagara 1 and Niagara 2 dies. These images are roughly to scale—the 65nm Niagara 2 chip is about 10% smaller than the 90nm Niagara 1 chip.

Multithreading Strategies in Server Processors

Server processors use a variety of multithreading strategies. Perhaps the simplest approach is the *chip multicore* strategy in Sun Microsystems' UltraSPARC IV processors. An UltraSPARC IV is essentially two single-threaded UltraSPARC III processor cores integrated on a single die. Apart from the usual cache-coherency issues in all multiprocessor systems, the only coordination needed between the two separate threads of execution is arbitration for shared resources.

Another strategy is core multithreading. The ideas behind chip multicore and core multithread designs are very different. Chip multicores are essentially multiprocessors on a chip. UltraSPARC IV processors provide the processing power of two equivalent UltraSPARC III processors—with no more than the frictional losses typical of all multiprocessor systems and an added small overhead cost for resource arbitration—because they effectively are two complete UltraSPARC III processors on a single die.

By contrast, core multithread designs are not full multiprocessors on a chip. For a variety of reasons—including limited exploitable instruction-level parallelism in many threads, large numbers of long-latency operations, and frequent thread stalls—wide superscalar designs tend to badly under-use available core execution resources. The goal of core multithreading is to take advantage of these otherwise idle resources to execute additional threads concurrently. This not only improves the overall usage of core resources but also gains incremental throughput performance at little incremental cost.

Of course, processors have long practiced timesharing to improve usage, switching among various tasks either on demand or at intervals. The drawback to timesharing is a relatively expensive context switch every time the executing thread changes. Hence, the more often a processor shares its time, the more cycles it spends handling overhead, and the fewer cycles it has left to advance the various time-sliced threads.

Core multithreading differs from traditional timesharing by holding the context of multiple threads "hot" in hardware. This involves duplicating the program counter, status and control registers, and integer and floating-point register sets. Other core facilities also may be augmented to reduce conflicts between the different "hot" threads contending for shared resources.

While the theory behind core multithreading is unimpeachable, implementation requires treading a fine line. In the worst case, core multithreading will deliver little or no benefit. In some programs, it may actually be detrimental: thread speed will slow, and there will be little or no compensating improvement in throughput. In the best case, there will be no noticeable degradation in thread speed, with up to a 40% (or better) improvement in throughput.

Contemporary sever processors practice three types of core multithreading. Coarse-grained multithreading involves switching away from an executing thread only when it stalls for a significant number of clock cycles (typically a cache miss that requires an access to main memory). In these schemes, a few cycles may be required to switch threads. The SPARC64 VI core implements coarse-grained multithreading (which Fujitsu calls vertical multithreading or VMT).

Fine-grained multithreading involves running multiple "hot" threads in round-robin fashion, switching between runnable threads every clock cycle. To be effective, fine-grained multithreading requires no-overhead or "zero-cycle" thread switching. Niagara processors implement fine-grained multithreading.

Both coarse-grained and fine-grained multithreading require concurrent threads to execute alternatively, with either few or no cycles required to switch between threads. Simultaneous multithreading (SMT), by contrast, enables a wide superscalar core to issue instructions from two (or more) threads in the same clock cycle. The POWER5 and POWER6 cores implement SMT. (See *MPR 10/30/06-02*, "POWER: The Sixth Generation.")

Where the first number represents the number of cores on die, and the second number represents the number of threads handled concurrently by each core, the chip multicore UltraSPARC IV processor can be characterized as a 2 × 1 design (with a total of two concurrent threads per processor chip). Current dual-core Opteron processors are another example of simple 2 × 1 multicore design. Next-generation quad-core Opterons will be 4 × 1 designs, or four-way multiprocessors on a chip.

Many of today's server processors combine chip multicore and core multithread strategies in the same design. Thus, SPARC64 VI and POWER6 both implement two dual-threaded cores per processor chip, or a 2 × 2 (four-thread) design. Evaluating these nominally similar designs, though, require an understanding of their different core multithreading strategies. A 2 × 2-VMT design is not the same as a 2 × 2-SMT design. And both these 2 × 2 four-thread designs differ significantly in potential from a 4 × 1 four-thread design.

Implementing multiple multithreaded cores on a single die allows the number of threads handled concurrently by a processor chip to escalate rapidly. The current leader among server processors, in terms of sheer thread count, is Sun's 8 × 8 (64-thread) Niagara 2. Because the Niagara core itself is a simple scalar design (double-wide, in the case of Niagara 2), Niagara processors are also the run-away leaders among contemporary server processors in core efficiency. They are perhaps the first processors ever to achieve an average instructions-per-cycle (IPC) efficiency remotely close to their peak IPC. To distinguish the sort of radical multiplication of threads possible with many (eight or more) heavily threaded cores integrated into a single processor chip, Sun uses the term chip multithreading or CMT.

Considering all the enhancements in Niagara 2, Sun's performance estimates are not surprising. What's interesting is the expected 40% improvement in single-thread performance—despite doubling the threads per core and the associated increase in pressure on shared memory resources. Assuming the frequency stays below 1.5GHz (an increase of less than 25%) and no great improvements in compiler efficiency, a 40% gain would indicate a significant boost in thread performance from the various microarchitectural and memory hierarchy enhancements.

In the year since Sun introduced the first Niagara 1 systems in December 2005, nobody else has introduced anything even remotely like Niagara. Sun reports strong sales of that product line—and not just to customers already in the installed SPARC/Solaris base. Assuming these early reports are indicative, Niagara 2 should not merely keep the momentum going, but accelerate it.

If Niagara processors are truly expanding the SPARC/Solaris universe, that's very good news for Sun and SPARC/Solaris customers in particular, and positive news in general for all vendors seeking to provide meaningful alternatives to ever more powerful 64-bit x86 processors from Intel and AMD. Not even Sun has been able to resist the price/performance benefits offered by 64-bit x86 systems, augmented by their advantages in software availability and vendor support.

Niagara 2 processors, like SPARC64 VI and POWER6 and Itanium 2 9000 processors, aim to show that even current dual and coming quad-core 64-bit x86 processors do not adequately address all computing needs. But other alternative high-end server processors are trying to attain levels of targeted performance, scalability, and reliability that commodity processors cannot economically reach. What sets Niagara apart is Sun's goal of expanding the computing spectrum with a new approach to performance efficiency. ◇

Price & Availability

Niagara 2 processors are expected to ship in systems in 2H07 (probably Q3) under the UltraSPARC T2 name. Sun has not yet indicated whether this design, like the UltraSPARC T1, will be placed in the OpenSPARC library available at *www.opensparc.net*.

Subscription Information

To subscribe to *Microprocessor Report*, contact our customer service department in Scottsdale, Arizona by phone, 480.483.4441; fax, 480.483.0400; email, *epotter@reedbusiness.com;* or Web, *www.MDRonline.com.*

One year	U.S. & Canada*	Elsewhere
Web access only	$895	$895
Hardcopy only	$995	$1,095
Both Hardcopy and Web access	$1,095	$1,195

Two years	U.S. & Canada*	Elsewhere
Web access only	$1,495	$1,495
Hardcopy only	$1,695	$1,895
Both Hardcopy and Web access	$1,895	$1,995

**Sales tax applies in the following states: AL, AZ, CO, DC, GA, HI, ID, IN, IA, KS, KY, LA, MD, MO, NV, NM, RI, SC, SD, TN, UT, VT, WA, and WV. GST or HST tax applies in Canada.*