Cahners
MICROPROCESSOR REPORT
THE INSIDERS' GUIDE TO MICROPROCESSOR HARDWARE
VOLUME 13, NUMBER 16
DECEMBER 6, 1999

Compaq Chooses SMT for Alpha

Simultaneous Multithreading Exploits Instruction- and Thread-Level Parallelism

MICROPROCESSOR FORUM 1999

by Keith Diefendorff

As it climbs rapidly past the 100-million-transistor-per-chip mark, the microprocessor industry is struggling with the question of how to get proportionally more performance out of these new transistors. Speaking at the recent Microprocessor Forum, Joel Emer, a Principal Member of the Technical Staff in Compaq's Alpha Development Group, described his company's approach: simultaneous multithreading, or SMT.

Emer's interest in SMT was inspired by the work of Dean Tullsen, who described the technique in 1995 while at the University of Washington. Since that time, Emer has been studying SMT along with other researchers at Washington. Once convinced of its value, he began evangelizing SMT within Compaq. His efforts have apparently paid off, as Compaq has officially adopted SMT for the Alpha 21464 (see MPR 11/15/99, p. 13), code-named EV8, which is due to appear in systems in 2003. That Compaq is talking about this processor three full years in advance indicates great confidence in SMT technology as well as a strong desire to establish that Alpha has a future.

SMT processors are similar to conventional superscalar out-of-order processors, but they have additional hardware resources that allow them to interleave the execution of multiple instruction streams, or threads, onto the execution units, as Figure 1 shows. By more fully utilizing the execution units in this way, SMT processors achieve higher sustained throughput and improved tolerance of memory latency.

The Debate: ILP or TLP

Even with 100 million of them on a chip, transistors are not free—yet. Hence, the question persists of how to deploy them in a way that maximizes performance. One alternative is to use them just to build larger on-chip memories, as Intel has done with the new Pentium III (see MPR 10/25/99, p. 1). This approach is effective, but only up to a point, beyond which little is gained from adding more cache. At that point, performance becomes limited by the speed of the processor core.

Given a full complement of on-chip memory, increasing the clock frequency will increase the performance of the core. One way to increase frequency is to deepen the pipeline. But with pipelines already reaching upwards of 12–14 stages, mounting inefficiencies may close this avenue, limiting future frequency improvements to those that can be attained from semiconductor-circuit speedup. Unfortunately this speedup, roughly 20% per year, is well below that required to attain the historical 60% per year performance increase. To prevent bursting this bubble, the only real alternative left is to exploit more and more parallelism.

Indeed, the pursuit of parallelism occupies the energy of many processor architects today. There are basically two theories: one is that instruction-level parallelism (ILP) is abundant and remains a viable resource waiting to be tapped; the other is that ILP is already tapped out, and it's time to move on to the richer vein of thread-level parallelism (TLP).

TLP proponents point to the rather depressing history of ILP progress. Over the past 10 years, processors have grown from simple single-issue machines with fewer than

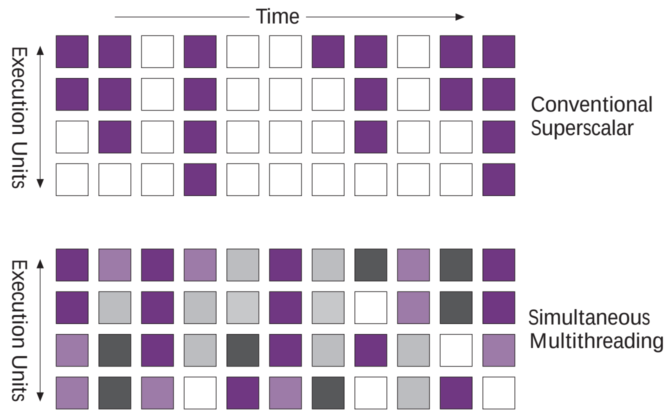


Figure 1. By interleaving instructions from multiple threads (various colors) onto a conventional superscalar pipeline, SMT processors leave fewer empty execution slots (white). This resulting high execution-unit utilization gives SMTs superior throughput for a given amount of hardware resources.

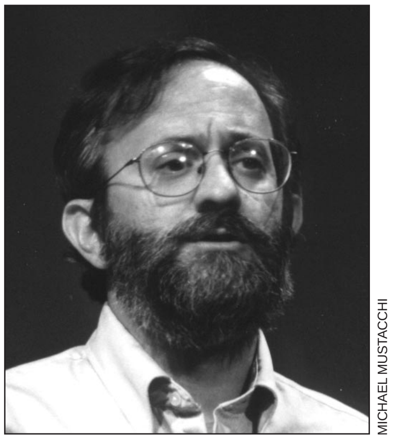
1 million core transistors to four-wide out-of-order behemoths with 10-million-transistor cores. At the same time, however, sustained ILP has done little better than double. Professor John Hennessy of Stanford in his keynote speech at the Forum showed data indicating that while the theoretical ILP of an assortment of SPEC95 benchmarks ranges from about 18 to 150 IPC, practical four-wide out-of-order superscalar processors rarely achieve even 2 IPC. ILP pessimists further assert that progress will be more dismal in the future, as diminishing returns will more severely curtail ILP gains.

No Lack of Ideas to Use Transistors

ILP proponents counter that with just a few more tens of millions of transistors, mixed with a little compiler magic, they can unleash this wealth of ILP. According to this group, radical models of execution that could not be considered in past are becoming feasible. HP and Intel with Itanium (see MPR 10/6/99, p. 1) are depending on static instruction scheduling by a compiler, predication, and very large register files to achieve a step-function increase in ILP.

Hal's Sparc64 V (see MPR 11/15/99, p. 1) is using trace processing and superspeculation to achieve high ILP. Optimists like Yale Patt at the University of Texas and John Shen at Carnegie-Mellon believe that such advanced superscalar techniques will allow ILP to scale with transistor count, ultimately enabling 16- or 32-wide processors with sustained ILP of 10 IPC or more on general-purpose applications.

Even if they succeed in extracting this degree of parallelism, such processors will all have one thing in common: physically large monolithic cores. Many will also have staggeringly complex control mechanisms. Although transistors will be plentiful enough to implement such machines, physics will surely intervene to enforce its immutable rule that large things are slow things. Furthermore, design and verification will become ever more difficult and time consuming.



Compaq researcher Joel Emer describes simultaneous multithreading at Microprocessor Forum.

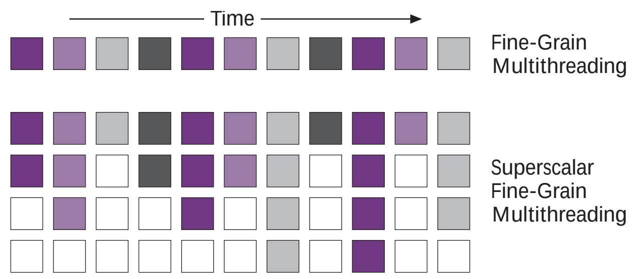

Time
Fine-Grain Multithreading
Superscalar Fine-Grain Multithreading

Figure 2. The precursor of SMT, fine-grain multithreading, fills pipeline bubbles by alternating the issue of instructions from several threads. Extending this idea to a superscalar pipeline would fill temporal pipeline bubbles, but, unlike SMT, would not fill execution units in space.

These realities, along with less confidence in ILP, have motivated IBM with Power4 (see MPR 10/6/99, p. 11) and Sun with MAJC (see MPR 10/25/99, p. 18) to shift their attention from ILP to explicit thread-level parallelism. These companies are using their transistors to build chip multiprocessors (CMPs). They believe it is wiser to keep processor cores small and fast, by limiting their issue widths, while relying on the parallelism between independent program threads to achieve higher performance.

A major drawback of both high-ILP processors and CMPs is that they suffer from poor transistor utilization when the workload doesn't match the processor. High-ILP processors speculate poorly or leave function units idle when faced with programs having inherently low ILP. Similarly, CMPs must leave entire processors idle when enough threads aren't available.

Enter SMT

Simultaneous multithreaded processors are a cross between wide-issue superscalar processors and fine-grain-multithread processors (see MPR 7/14/97, p. 13). Fine-grain multithreading (FMT) was first implemented by Seymour Cray in the peripheral-processing unit of the CDC 6600 (circa 1964), then again in the late 1970s in Denelcor's HEP, and more recently in Tera's MTA. FMTs maintain state information for several active threads, and on each cycle they issue one instruction from a different thread. The advantage of this technique is that it fills pipeline bubbles created by dependencies on long latency operations (e.g., memory accesses) with instructions from known-independent threads. This is far easier and more effective than trying to fill bubbles by ferreting out and reordering independent operations from a single thread.

If FMT were straightforwardly extended to superscalar issue, as Figure 2 shows, it would address the problem of low temporal utilization of execution units (pipeline bubbles), but the problem of low spatial utilization (empty execution slots) would remain, due to intrathread dependencies. Simultaneous multithreading, however, allows instructions to be selected for issue from any ready thread, as Figure 1 shows. In this way, SMT processors can fill unused execution slots with useful work.

The real beauty of SMT is that as threads execute, the machine can dynamically reallocate execution resources on the basis of the mix of parallelism in the workload. A single thread with a high degree of ILP can utilize the full resources of the machine for maximum speed; alternatively, resources can be distributed among several threads to achieve high throughput, even in the face of low ILP. Indeed, any combination of workload types can execute concurrently, with performance limited only by the total available resources.

SMT's ability to exploit parallelism in a wide variety of workloads produces consistently high execution-unit utilization, a fact that enables designers to consider wider superscalar designs than could be justified on ILP alone. Although Emer counts himself in the camp of ILP-optimists and says that EV8 would have been eight-wide even without SMT, he is less sanguine about ILP beyond that which is exploitable by an eight-wide processor. With SMT to take up the excess, however, even wider machines might be effective.

Not That Different From Wide Superscalar

Conceptually, SMTs are similar to wide-issue dynamically scheduled processors, as Figure 3 shows. In fact, no new control mechanisms are needed to issue instructions from multiple threads. The traditional register-renaming scheme, for example, avoids false dependencies (register name conflicts) between threads in the same way it does within a single thread: by mapping architectural registers from the active thread onto the processor's pool of physical registers.

This is not to say that no additional hardware is required for SMT. Thread identifiers, for example, must be appended to each instruction so thread-specific operations, such as branch prediction and virtual address translation, can be performed as instructions flow through the pipeline. Also, some processor resources must be duplicated so that state information (registers, program counter, etc.) can be maintained separately for each active thread context.

Other hardware, such as that required for recovering from branch mispredictions, handling program exceptions, maintaining precise interrupts, returning from subroutines, and retiring instructions in order, must either be replicated for each thread or shared, which requires more complex bookkeeping logic.

While the aforementioned additions are required to achieve proper function, even more hardware is probably needed to carry the heavier load of multiple threads. Instruction queues must be deeper, and more registers must be available in the renaming pool. Caches, translation-lookaside buffers (TLBs), and branch-history tables (BHTs), should also be larger, be more associative, and have more ports. And because the SMT's execution units are shared among several simultaneous threads, their number and symmetry may have to be increased to prevent contention.

While these additional hardware resources do not themselves add much complexity beyond that found in a conventional superscalar processor, they do add size. To prevent this size increase from impacting cycle time, steps must be taken that do indeed increase complexity. The caches, for example, may have to be partitioned into multiple smaller banks; the register files and execution units may also have to be partitioned, as they are in the 21264 (see MPR 10/28/96, p. 11); and the pipeline may have to be lengthened, putting pressure on the branch predictor, rename registers, and reorder buffer.

Even though SMTs require incremental hardware to support each thread, an SMT capable of running four simultaneous threads, for example, would be nowhere near four times larger than a single-thread superscalar of the same issue width. Two things account for this economy: first, SMT threads exploit hardware that would otherwise be sitting idle; second, the statistical variations in multiple threads running asynchronously prevent excessive contention for some hardware. Thus, a good deal of hardware—the execution units and caches, for example—can be effectively shared, avoiding hardware increases for each thread. Indeed, Compaq says that EV8's resources are sized for a single-thread and that additional SMT threads are treated as opportunistic. Future processors, however, may indeed have beefed-up resources to reduce conflicts.

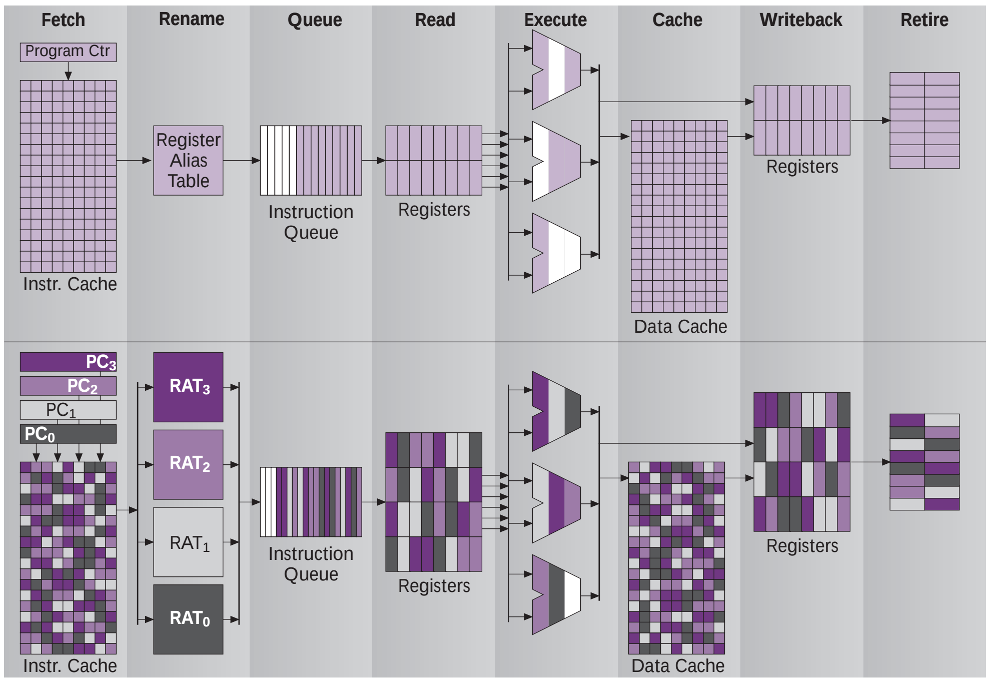


Figure 3. The SMT pipeline (bottom) is similar to a conventional out-of-order superscalar pipeline (top) with the addition of a few thread-specific resources at the front of the pipeline, thread-sensitive BHT and TLBs, and some extra registers.

Instruction Fetch Limits Throughput

Because the SMT has more independent instructions at its disposal (from separate threads), it can issue instructions at a far higher rate than a single-thread processor. This higher issue rate puts severe pressure on the instruction fetcher. In fact, instruction fetch is potentially the most severe bottleneck in an SMT processor. Therefore, it is necessary to minimize branch mispredictions, to minimize fetching speculative instructions when nonspeculative ones are available, and to have an intelligent mechanism for selecting threads from which to fetch.

Emer described one possible scheme in a paper he co-authored for the Sept/Oct '97 issue of *IEEE Micro*. In the eight-wide SMT hypothesized in that paper, on every cycle the instruction fetch unit fetched eight instructions from each of two threads that were not currently processing an instruction-cache miss. Instructions were selected for dispatch from the first thread until either a branch or an end-of-cache line was encountered, at which time instructions were selected from the second thread.

The two threads were selected using an Icount feedback technique. This technique prioritizes fetch from threads that currently have the fewest instructions in the front end of the pipeline. The theory behind Icount is that it gives the highest fetch priority to the fastest-moving threads and maximizes interthread parallelism by maintaining an even distribution of instructions from different threads in the instruction queues. Icount also prevents thread starvation, since threads with the fewest instructions in the pipeline are the first to get new fetch cycles.

Icount scheduling has the fortuitous characteristic of very low hardware cost; all that's required is a simple up/down counter for each thread and some comparators to select the two threads with the smallest counts. In the *Micro* paper, the researchers found Icount to be more effective than alternative schemes that sought to fetch from threads in ways that minimized branch mispredictions or load delays.

But Does It Work?

Apparently so. Although no one has yet built an SMT, simulations show it to be promising. On the hypothetical eight-wide machine in Emer's *Micro* paper—which had six ALUs (four of which can load or store) and four FPUs—Emer reported a speedup of slightly more than 2× for four threads over one. The speedup held for both multiprogrammed single-thread applications and for single multithreaded programs. To simulate the worst case for multiprogramming (most potential interthread contention), the same application was executed for all four threads. Applications were selected from the SPEC95 and Splash2 benchmark suites.

Performance gains flattened abruptly for more than four threads; eight threads showed no appreciable benefit over four. Presumably, four threads were able to saturate the execution resources of the hypothetical machine, limiting further gains. This result was probably influential in Compaq's decision to limit the eight-wide EV8 to four active threads.

To support more than four simultaneous threads, EV8's fetch, dispatch, and issue widths would probably have to be increased along with the number of execution units. Since EV8 is slated for a 0.125-micron process and a 250-million-transistor budget, we doubt it was concern over transistor count that limited the width. Instead, it was probably the complexity and cycle-time implications of going beyond eight-wide superscalar. It could also have been the enormous demand SMT puts on memory: just to support four threads, EV8 will have a direct multichannel interface to RDRAM main memory, and, although Compaq has not stated this, it will probably have more than 3M of on-chip L2 cache.

At the Forum, Emer presented additional simulated-benchmark results, further illustrating the speedup achievable by an SMT processor. As Figure 4 shows, with a multiprogramming workload of mixed integer and floating-point benchmarks, four-way SMT had nearly 125% higher throughput on four threads than on one. On multithreaded programs, four-way SMT achieved better throughput by an average of

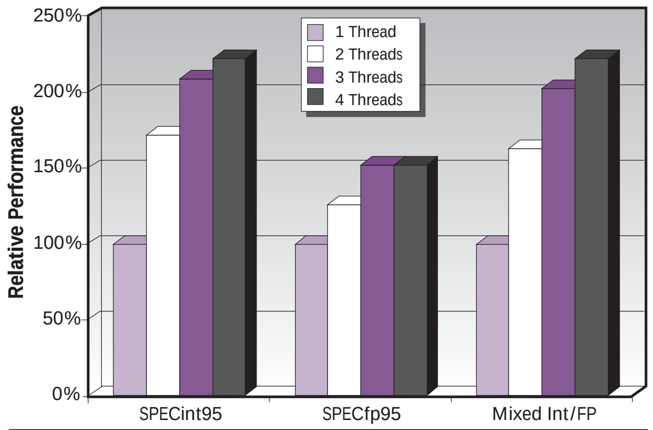


Figure 4. On a multiprogramming workload comprising a mixture of SPECint95 and SPECfp95 benchmarks, Compaq claims the SMT it simulated achieves a 125% higher throughput with four simultaneous threads than with just one thread. (Source: Compaq)

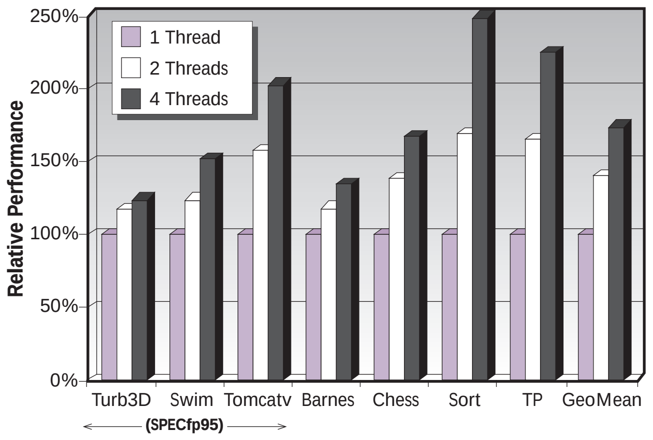


Figure 5. On applications that have been written to be multithreaded, Compaq found that its SMT design averaged nearly 75% better throughput when running four threads. (Source: Compaq)

75%, as Figure 5 shows. The SPECfp95 benchmarks in this suite were automatically decomposed into threads, and Emer says a manual decomposition may produce better results.

These results are impressive, considering the modest amount of hardware required to support three additional threads. Compaq claims that for EV8 the additional silicon area for its four-thread SMT core above the base eight-wide superscalar core is less than 10%. In comparison, doubling the silicon area of a single-thread processor typically boosts performance by less than 50%—and that percentage is trending down. We know of no other EPIC or advanced superscalar approach that could double the performance of an eight-wide superscalar Alpha processor for less than double the silicon. Thus, the approximately 2× speedup Emer reported would seem to make SMT a real bargain.

It is important to remember, however, that Emer's benchmarks measure speedup when thread-level parallelism is present. In real systems, however, sometimes TLP will not be present. Thus, in practice, the speedup from SMT will, on average, be less than Emer's benchmark results show.

It is also important to note that there is a big difference in complexity and area between a four-wide and an eight-wide superscalar. Therefore, these results cannot be used to compare EV8 with the alternative of, for example, two four-wide superscalar processors on a CMP (chip multiprocessor). In his *Micro* paper, however, Emer reported that a CMP with two cores, each having roughly half the resources of the hypothetical eight-wide SMT, showed similar speedups for two threads, but it fell well short of SMT's four-thread performance. The SMT is also likely to have better single-thread performance than the CMP when ILP is present.

The Pesky Matter of Software

As is frequently the case with techniques to speed up processors, SMT is not without software issues. Although SMT executes single-thread programs with no difficulty, problems creep in when you try to use its multithreading capability.

For multiprogramming workloads (workloads comprising multiple individual programs running simultaneously), the problems are tractable; the software implications are minor and restricted to the operating system. For this case, the OS simply needs to prioritize threads and to keep the most important thread contexts resident on the processor. Multiprogramming speedup, however, is important today only in server environments that are currently served by symmetric multiprocessors (SMPs).

To fully justify SMT, however, it is necessary to also take advantage of single-program multithreading. To enable this, programs must be decomposed into multiple independent threads that the SMT can execute in parallel. This requires two things: the presence of thread-level parallelism in the program and the ability to find and expose it.

Unfortunately, techniques for automatically decomposing programs into parallel threads are in their infancy. Guri Sohi at the University of Wisconsin is pursuing multiscalar techniques in which a single thread is decomposed into mini-"tasks" according to the program flow graph; multiple task sequencers then use aggressive control and data-value speculation to execute these tasks in parallel. Former graduate student Scott Breach has shown that enhanced SMT hardware can be used to run these minitasks in parallel.

But how effective compilers will be in automatically creating parallel threads from a single program remains to be seen. Today, the burden of parallelizing programs remains a largely manual process. To make matters worse, debugging multithreaded programs is notoriously difficult—a fact that deters many programmers. Although multithreading is becoming a more accepted style of programming, especially with Java, today most programs are still single threaded, and most programmers are still poorly trained to code for explicit parallelism. This obstacle could prevent SMTs from realizing their full potential for several years. Perhaps by 2003, when EV8 systems are due to appear, things will have changed.

The architectural abstraction that Compaq has adopted for programming EV8 is that of a CPU with four thread-processing units (TPUs), as Figure 6 shows. This abstraction creates a programming model of SMT as a sort of virtual CMP. In fact, the SMT is functionally similar to CMP in many ways. For example, both share data among threads without going off chip, both exploit thread-level parallelism, and both can switch thread contexts in about the same amount of time.

One difference between the two, one that Compaq's abstraction makes clear, is that SMT threads share data at the L1 without the overhead of the cache-coherency actions required by CMPs with separate L1s. This feature gives SMTs the potential for slightly finer-grain threading and tighter coupling between threads. On the other hand, because they share data at the L2—and do not share L1s, BHTs, TLBs, execution units, or anything else—CMPs provide a higher degree of thread isolation; that is, the performance of one thread is less dependent on the characteristics of other threads than it would be on an SMT. This isolation may be an advantage in some situations, such as in critical real-time applications.

To make the TPU model work, one problem Compaq had to eliminate was the problem of spin loops. Whenever

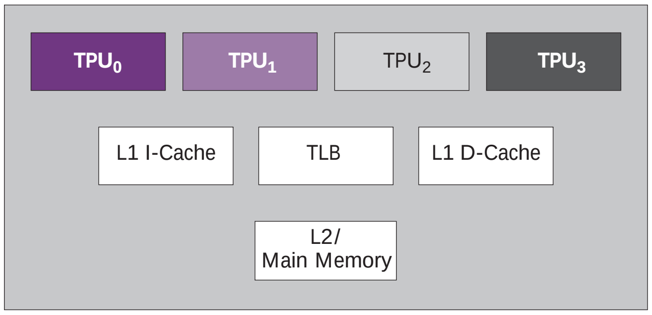


Figure 6. The programming abstraction Compaq will use treats EV8 as four virtual processors, called thread-processing units (TPUs), which all share a common set of caches and TLBs.

multiple threads cooperate, mechanisms are needed to synchronize threads, communicate between threads, lock shared resources, and protect critical software sections. These functions are normally accomplished in software by low-level semaphore operations that involve putting the processor into spin loops while polling for semaphore changes. Spin loops in an SMT, however, are a disaster because they consume one of the TPUs while performing no real work.

To circumvent this problem, Compaq devised a method for putting a thread to sleep and waking it when a given memory location changes. Instructions are not fetched or issued from a sleeping thread, allowing other active threads to utilize more of the processor's resources. The scheme was inexpensive to implement, as it relies on the existing load-with-lock/store-conditional semaphore mechanisms already in the Alpha architecture and the cache-coherency mechanisms that already exist to detect cache-line modifications.

Won't Affect Cycle Time, Right?

According to Emer, SMT need not lengthen cycle time. Emer believes that the cycle time of a CPU should be set according to the highest speed that the ALU can evaluate and forward results to subsequent instructions. The pipeline length should then be established by dividing execution into stages no longer than the ALU cycle time. But SMTs need more registers and thus longer operand-read times than a superscalar. To prevent these factors from impacting the cycle time, it is very likely that at least one additional pipeline stage will be required, which would add to the branch-mispredict penalty. Other SMT-specific resources, such as more instruction-completion writeback ports, could impose additional stages.

As a result, it is likely that a single-thread application will not perform as well on an eight-wide SMT as it would on a superscalar of similar design. This loss of single-thread performance, if indeed it is only one pipeline stage, probably amounts to only a few percent. If SMT turns out to have other resources or control complexities that add more pipeline stages or increase cycle times, the net benefit of SMT will be less clear. But Emer sees no reason to expect any cycle-time penalties or any more than one or two extra pipeline stages.

Another potential performance limitation is resource contention among threads. Even in a 0.125-micron process, execution units will not be completely symmetric, and not every structural hazard will be eliminated. Even worse, contention for the caches, the BTB, and the TLB could increase miss rates or, in the worst case, cause severe thrashing. Assuming these resources are sufficiently associative, thrashing should be avoidable, but cache miss rates will definitely go up, due to increased conflicts. SMT's greater ability to tolerate memory latency should compensate to some degree—but to what extent remains to be seen. Compaq says it has seen cases of positive interference, such as prefetching system code, but these cases are probably the exception rather than the rule.

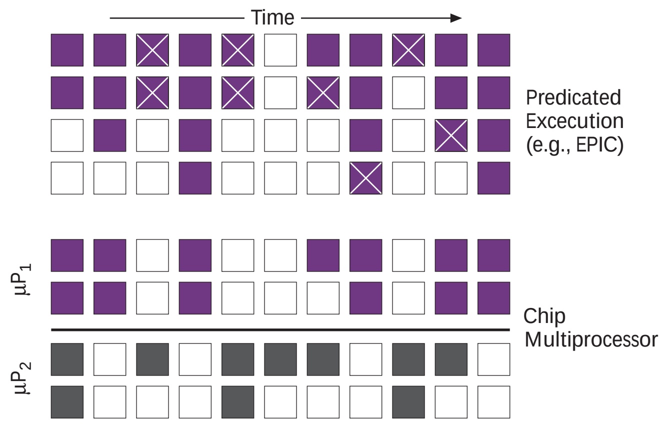


Figure 7. Predicated-execution, such as in Intel and HP's EPIC, tries to fill unused execution slots with speculative operations but often ends up throwing away results (×). Chip multiprocessors, like SMTs, exploit TLP, but they have lower single-thread performance than SMTs for a given amount of resources.

Alternatives Abound

With transistor budgets soon to exceed 100 million transistors per chip, a host of architects with ideas on how to spend those transistors has emerged. The most popular ideas being espoused for general-purpose microprocessors, aside from SMT, include advanced superscalar processors (e.g., trace processing, superspeculation, and multiscalar), EPIC (explicitly parallel architectures), and CMP (chip multiprocessors).

These ideas are not necessarily mutually exclusive and could conceivably be used in combination. In the near term, however, sheer size and complexity will preclude most combinations. Longer term, with say a billion-transistor budget, nearly any combination could, in theory, be built. But many hybrids will not bear fruit, regardless of the transistor budget. SMT is likely to be incompatible with some of the advanced superscalar techniques. These techniques, for example, frequently depend on speculation. But SMT and speculation both vie for the same resources, and both stress the fetch unit to achieve their goal.

SMT is probably even less compatible with EPIC than it is with advanced superscalars. Although Intel has alluded to the possibility of eventually adding multithreading to future IA-64 implementations, it is not clear that move will be feasible. SMT depends, by its very nature, on the dynamic-scheduling hardware that is present in superscalars but is completely lacking in EPIC. Adding these mechanisms on top of EPIC would risk massive complexity, and it would defeat one of its central tenets. Furthermore, the result may be disappointing. EPIC, using predicated execution, attempts to fill idle function units with speculative operations from the current program thread. To the extent it succeeds in this objective, EPIC would reduce the effectiveness of SMT by usurping the very execution units on which SMT thrives, as Figure 7 shows.

Since the techniques are not always synergistic, SMTs will likely end up facing advanced superscalar and EPIC processors in the market. Against these techniques, SMT will have the powerful advantage of being able to evoke either thread-level or instruction-level parallelism at will. This advantage will materialize only when enough total parallelism is available, but this flexibility will allow the SMT to perform well in many situations where these other techniques would fail miserably.

SMT will have the disadvantage, however, that in single-thread environments, programs must be explicitly written to expose thread-parallel parallelism. If programs do not migrate to multithreaded construction, then SMT's additional resources will go for naught, and its single-thread performance is likely to be inferior to one of the other techniques using equivalent resources.

In server environments, which are usually heavily multiprogrammed, this disadvantage will not come into play. But even in a multiprogrammed or multithreaded environment, SMT will be of little benefit if individual programs have high ILP. In such a case, the execution units will be kept busy by the high-ILP thread, leaving few execution slots for other threads.

SMT, CMP Square Off

SMT's most serious long-term challenge will probably come from CMPs, which have some compelling advantages of their own. A CMP core, because it is typically smaller and less ILP-aggressive than an SMT core, is likely to achieve a higher frequency and/or have a shorter, more efficient pipeline. If ILP turns out to be limited or to be hard to exploit with wide-issue machines—and there is precious little hard evidence to the contrary—then CMPs, which can also play the thread-level-parallelism card, might perform as well as an SMT.

If performance is similar, then CMP construction wins. Building one small, simple core and replicating it along with a shared L2 is a far simpler and more expeditious task than designing a large, complex, monolithic core. In addition, CMPs introduce the potential for using partially-good die. This possibility can reduce manufacturing scrap, thereby reducing the average manufacturing cost of a CMP die.

Because an SMT shares more resources among threads, it will probably have a physically smaller die than an equivalent performance CMP. But this advantage may be less than it seems. For one thing, given upcoming transistor budgets, sharing resources may not save enough silicon to be worth the control complexity needed to do so. Second, the high execution-unit utilization of SMTs could create longer queue delays and longer latencies that would require additional hardware in the SMT to ameliorate. Third, utilization is naturally higher and therefore less of a problem on narrow-issue CMP cores. SMTs, in a sense, create an artificially low utilization situation by starting out with an excessively wide-issue engine.

In the future, CMP and SMT techniques might create an interesting marriage. If low utilization is a problem even for modest four-wide superscalar CMP cores—which, with a throughput of less than 2 IPC, would seem to be the case—then a simple four-wide/two-thread SMT core might eliminate the problem. Arraying this core in CMP fashion might provide a simple path for scaling beyond the four active threads that are the limit of an EV8-class eight-wide SMT. Architects of IBM's Power4 CMP (see MPR 10/6/99, p. 11) have already expressed a possible interest in multithreading for the future.

Putting multiple EPIC or advanced-superscalar processors on a chip will be another way to exploit ILP and TLP; the question is whether there is enough ILP to justify using these more complex cores. Although this option may not be realistic in the near term—say over the next three to four years, while transistor budgets are limited to a measly 100 million to 250 million transistors per chip—in the long term it could pose a powerful alternative to SMT.

In the meantime, the one incontrovertible advantage of SMT—and the characteristic that makes it attractive over all other known forms of advanced superscalar, EPIC, CMP, or combinations thereof—is its unique ability to shift resources on the fly between ILP and TLP at a very fine grain. The ultimate value of this advantage, however, will depend heavily on software evolution.

To go beyond servers, either something like multimedia must drive up the use of multiprogramming in PC environments, or a much broader range of applications must move to multithreaded construction. This move could happen quickly if compiler techniques evolve to automatically create parallel threads, or if Java—which already has multithreaded API classes and background tasks—takes hold. If either event happens over the next three years, we may see more vendors adopting the clever technique of SMT.

For multiprogrammed server environments, however, SMT is readily applicable. And Compaq says the programs used in many of Alpha's key application areas, such as data warehousing, graphics rendering, and government supercomputing, are already multithreaded. Assuming that Compaq remains committed to Alpha, and doesn't let annoying details such as IC process and system design stand in its way, SMT should provide a solid basis for the company to retain Alpha's long-standing performance title over all comers. M

For More Information

"Simultaneous Multithreading: Maximizing On-Chip Parallelism," Tullsen, Eggers, and Levy, *ISCA95*.

"Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreaded Processor," Tullsen, Eggers, Emer, Levy, Lo, and Stamm, *ISCA96*.

"Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading," Lo, Eggers, Emer, Levy, Stamm, and Tullsen, *ACM Transactions on Computer Systems*, August 1997.

"Simultaneous Multithreading: A Platform for Next-Generation Processors," Eggers, Emer, Levy, Lo, Stamm, and Tullsen, *IEEE Micro*, October, 1997.