A Single-Chip Multiprocessor

What kind of architecture will best support a billion transistors? A comparison of three architectures indicates that a multiprocessor on a chip will be easiest to implement while still offering excellent performance.

Lance Hammond
Basem A. Nayfeh
Kunle Olukotun
Stanford University

Integrated circuit processing technology offers increasing integration density, which fuels microprocessor performance growth. Within 10 years it will be possible to integrate a billion transistors on a reasonably sized silicon chip. At this integration level, it is necessary to find parallelism to effectively utilize the transistors. Currently, processor designs dynamically extract parallelism with these transistors by executing many instructions within a single, sequential program in parallel. To find independent instructions within a sequential sequence of instructions, or thread of control, today's processors increasingly make use of sophisticated architectural features. Examples are out-of-order instruction execution and speculative execution of instructions after branches predicted with dynamic hardware branch prediction techniques.

Future performance improvements will require processors to be enlarged to execute more instructions per clock cycle.[1] However, reliance on a single thread of control limits the parallelism available for many applications, and the cost of extracting parallelism from a single thread is becoming prohibitive. This cost manifests itself in numerous ways, including increased die area and longer design and verification times. In general, we see diminishing returns when trying to extract parallelism from a single thread. To continue this trend will trade only incremental performance increases for large increases in overall complexity.

Although this parallelization might be achieved dynamically in hardware, we advocate using a software approach instead, allowing the hardware to be simple and fast. Emerging parallel compilation technologies,[2] an increase in the use of inherently parallel applications (such as multimedia), and more widespread use of multitasking operating systems should make this feasible.

ALTERNATIVE APPROACHES

Researchers have proposed two alternative microarchitectures that exploit multiple threads of control: simultaneous multithreading (SMT)[3] and chip multiprocessors (CMP).[4]

SMT processors augment wide (issuing many instructions at once) superscalar processors with hardware that allows the processor to execute instructions from multiple threads of control concurrently when possible, dynamically selecting and executing instructions from many active threads simultaneously. This promotes much higher utilization of the processor's execution resources and provides latency tolerance in case a thread stalls due to cache misses or data dependencies. When multiple threads are not available, however, the SMT simply looks like a conventional wide-issue superscalar.

CMPs use relatively simple single-thread processor cores to exploit only moderate amounts of parallelism within any one thread, while executing multiple threads in parallel across multiple processor cores. If an application cannot be effectively decomposed into threads, CMPs will be underutilized.

From a purely architectural point of view, the SMT processor's flexibility makes it superior. However, the need to limit the effects of interconnect delays, which are becoming much slower than transistor gate delays, will also drive the billion-transistor chip design. Interconnect delays will force the microarchitecture to be partitioned into small, localized processing elements. For this reason, the CMP is much more promising because it is already partitioned into individual processing cores. Because these cores are relatively simple, they are amenable to speed optimization and can be designed relatively easily.

In this article, we explain why software and hardware trends will favor the CMP microarchitecture. We base our conclusion on the performance results from a comparison of simulated superscalar, SMT, and CMP microarchitectures.

EXPLOITING PARALLELISM

Parallelism exists at multiple levels in modern systems. Parallelism between individual, independent instructions in a single application is *instruction-level parallelism*. *Loop-level parallelism* results when the instruction-level parallelism comes from data-independent loop iterations. The finite number of instructions that can be examined at once by hardware looking for instruction-level parallelism to exploit is called the *instruction window size*. This window size is a crucial factor in billion-transistor processors, because they must be designed to exploit as much parallelism as possible.

Compilers, which have essentially infinite virtual

We expect thread and process parallelism to become widespread, for two reasons: the nature of the applications and the nature of the operating system.

instruction windows as they generate code, can help increase usable parallelism by reordering instructions. Instructions are reordered so that instructions that can be issued in parallel are close to each other in executable code, allowing the hardware's finite window to detect the resulting instruction-level parallelism. Some compilers can also divide a program into multiple threads of control, exposing *thread-level parallelism*. This form of parallelism simulates a single, large, hardware instruction window by allowing multiple, smaller instruction windows—one for each thread—to work together on one application.

A third form of very coarse parallelism, *process-level parallelism*, involves completely independent applications running in independent processes controlled by the operating system.

In the future, we expect thread and process parallelism to become widespread, for two reasons: the nature of the applications and the nature of the operating system.

First, tomorrow's multimedia user interfaces will make extensive use of 3D graphics, full-motion video, image recognition, voice generation, and voice recognition. These multimedia applications are computationally intensive, with significant loop-level parallelism. The emergence of sophisticated parallelizing compiler technology will enable the conversion of loop-level parallelism into thread-level parallelism.[2] Future applications that include, for example, computational chemistry, fluid dynamics, and other forms of physical modeling will also feature significant loop-level parallelism. These applications can also be made multithreaded with a compiler. Some applications, such as compilers themselves, are impossible to parallelize even with the most sophisticated compiler technology. However, hardware support that eliminates the difficult problem of detecting memory dependencies at compile time can make even these applications parallelizable.[5,6]

A second reason that parallelism will become more prevalent is that multiprocessor-aware operating systems and environments, such as Microsoft Windows NT and Unix, execute separate applications in parallel to increase throughput and provide a more responsive computing environment. Because these sophisticated, multimedia-laden user environments have enabled users to run more applications simultaneously, process-level parallelism is increasing as a result.

IMPLEMENTATION TECHNOLOGY CONCERNS

In a billion-transistor CMOS implementation technology, the most crucial design issue will be managing

Table 1. Characteristics of superscalar, simultaneous multithreading, and chip multiprocessor architectures.

Characteristic	Superscalar	Simultaneous multithreading	Chip multiprocessor
Number of CPUs	1	1	8
CPU issue width	12	12	2 per CPU
Number of threads	1	8	1 per CPU
Architecture registers (for integer and floating point)	32	32 per thread	32 per CPU
Physical registers (for integer and floating point)	32 + 256	256 + 256	32 + 32 per CPU
Instruction window size	256	256	32 per CPU
Branch predictor table size (entries)	32,768	32,768	8 × 4,096
Return stack size	64 entries	64 entries	8 × 8 entries
Instruction (I) and data (D) cache organization	1 × 8 banks	1 × 8 banks	1 bank
I and D cache sizes	128 Kbytes	128 Kbytes	16 Kbytes per CPU
I and D cache associativities	4-way	4-way	4-way
I and D cache line sizes (bytes)	32	32	32
I and D cache access times (cycles)	2	2	1
Secondary cache organization (Mbytes)	1 × 8 banks	1 × 8 banks	1 × 8 banks
Secondary cache size (bytes)	8	8	8
Secondary cache associativity	4-way	4-way	4-way
Secondary cache line size (bytes)	32	32	32
Secondary cache access time (cycles)	5	5	7
Secondary cache occupancy per access (cycles)	1	1	1
Memory organization (no. of banks)	4	4	4
Memory access time (cycles)	50	50	50
Memory occupancy per access (cycles)	13	13	13

interconnect delay. While transistor gate delay decreases linearly with decreases in minimum feature size, the wire delay stays nearly constant or increases as wires become finer. This is due to increasing wire resistance to load capacitance ratios. Furthermore, die sizes are expanding as designers strive for higher performance by using more gates than increased gate density alone provides. The result is that today the delay of the longest wires on a chip is improving two to four times more slowly than gate delay.

In addition, processor clock rates have been rising exponentially as circuit designers have increasingly optimized critical paths. A long wire in a circuit can now affect the processor's cycle time if it happens to be on a critical timing path, or require additional pipeline stages just to hide the delay of the wire drivers. For example, Digital Equipment Corp.'s Alpha 21264[7] includes a cycle in its pipeline just to drive data from the outputs of the primary data cache back to the processor core.

The result of these trends is that the layout of a billion-transistor chip will significantly affect the processor's architecture. Resources on a processor chip that must communicate with each other in a single cycle must be physically close together. Architecture designers will need to excel either at arranging the sections of their CPUs to avoid long wires or at designing pipelines that tolerate long wire latencies. This type of design should logically result in a CPU built of several small, high-speed logic blocks connected by longer, slower wires that are only infrequently needed or whose delay can be easily hidden by pipelining.

A concern second only to managing interconnect delay is managing design complexity. The abstraction level at which microprocessors are designed has risen, and better tools for logic synthesis and automatic layout tools are available. Nevertheless, the size of microprocessor design teams has grown proportionally to the transistor gate count. Another factor in design complexity is the quest for higher performance, which drives up clock frequency. This in turn requires even more designers, to maintain the time to market. Additionally, more engineers are needed to validate the increasingly complex designs. Our increased ability to create complex architectures is outstripping our ability to validate them. Thus, a radical change in design style or a breakthrough in validation technology will be required to economically complete a billion-transistor microprocessor.

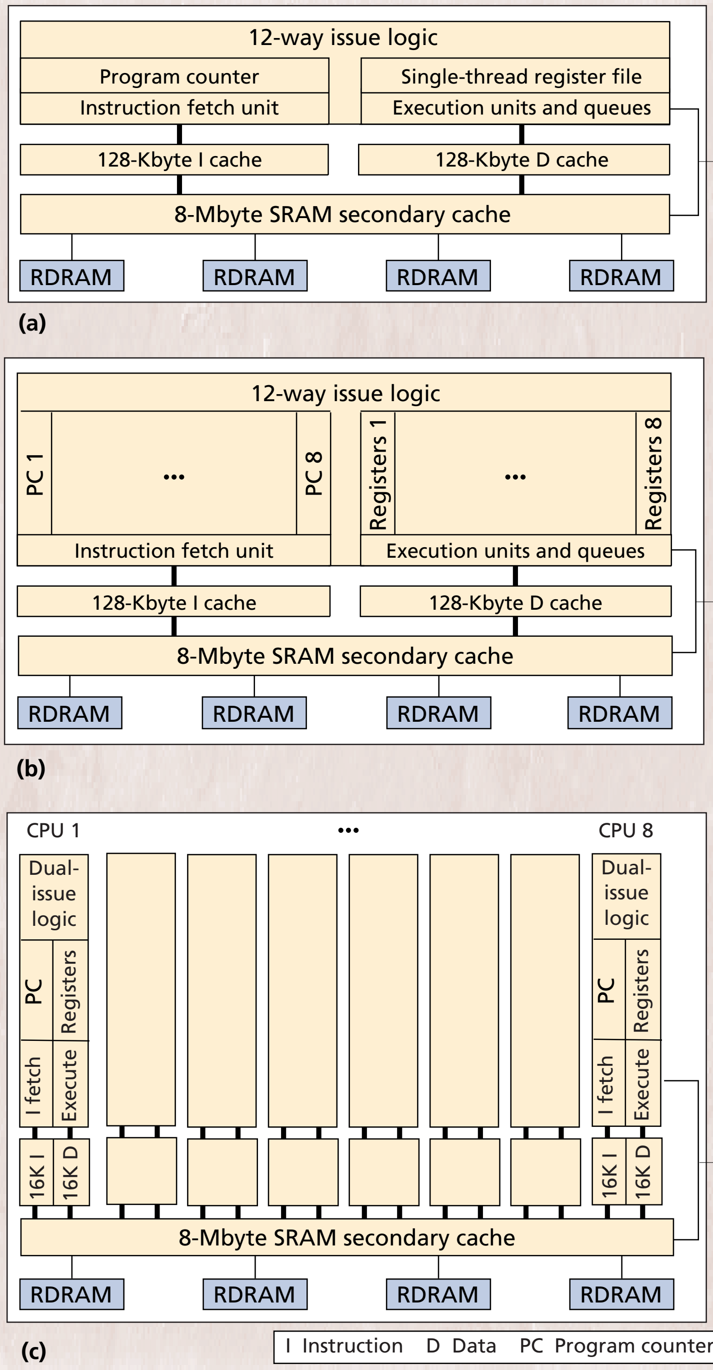

Figure 1. Comparing (a) superscalar, (b) simultaneous multithreading, and (c) chip multiprocessor architectures.

COMPARING ALTERNATIVE ARCHITECTURES

With these implementation concerns and design challenges in mind, we simulated three architectures—superscalar, SMT, and CMP—to see which one held the most potential at billion-transistor integration levels. We established a standard chip area and integration density based on a one-billion transistor DRAM. From this standard, we determined the parameters for the three architectures listed in Table 1. Figure 1 illustrates the architectures.

The superscalar processor, shown in Figure 1a, can dynamically issue up to 12 instructions per cycle. This is a straightforward extension of current commercially

With even more advanced IC technologies, the logic, wire, and design complexity advantages will increasingly favor a multiprocessor implementation.

available superscalar processors. The SMT processor, shown in Figure 1b, is identical to the superscalar except that it has eight separate program counters and executes instructions from up to eight different threads of control concurrently. The processor core dynamically allocates instruction fetch and execution resources among the different threads on a cycle-by-cycle basis to find as much thread-level and instruction-level parallelism as possible.

The CMP, shown in Figure 1c, is composed of eight small 2-issue superscalar processors. This processor depends on thread-level parallelism, since its ability to find instruction-level parallelism is limited by the small size of each processor.

Before we discuss the quantitative performance results we achieved from our simulations, we explain the architectures' major design considerations in a qualitative manner.

CPU cores

To keep the processors' execution units busy, our simulated superscalar and SMT processors feature advanced branch prediction, register renaming, out-of-order instruction issue, and nonblocking data caches.[8] As a result, the processors have numerous multiported rename buffers, issue queues, and register files.

The inherent complexity of these architectures results in three major hardware design problems that the CMP approach solves by keeping the hardware simple:

- Their area increases *quadratically* with the core's complexity. The number of registers in each structure must increase proportionally to the instruction window size. Additionally, the number of ports on each register must increase proportionally to the processor's issue width.

  The CMP approach minimizes this problem because it attempts to exploit higher levels of instruction-level parallelism using more processors instead of larger issue widths within a single processor. This results in an approximately *linear* area-to-issue width relationship, since the area of each additional processor is essentially constant, and it adds a constant number of issue slots. Using this relationship, the area of an 8 × 2-issue CMP (16 total issue slots) has an area similar to that of a single 12-issue processor.

- They can require longer cycle times. Long, high-capacitance I/O wires span the large buffers, queues, and register files. Extensive use of multiplexers and crossbars to interconnect these units adds more capacitance. Delays associated with these wires will probably dominate the delay along the CPU's critical path. The cycle time impact of these structures can be mitigated by careful design using deep pipelining, by breaking up the structures into small, fast clusters of closely related components connected by short wires, or both. But deeper pipelining increases branch misprediction penalties, and clustering tends to reduce the ability of the processor to find and exploit instruction-level parallelism.

  The CMP approach allows a fairly short cycle time to be targeted with relatively little design effort, since its hardware is naturally clustered—each of the small CPUs is already a very small fast cluster of components. Since the operating system allocates a single software thread of control to each processor, the partitioning of work among the "clusters" is natural and requires no hardware to dynamically allocate instructions to different component clusters. This heavy reliance on software to direct instructions to clusters limits the amount of instruction-level parallelism that can be dynamically exploited by the entire CMP, but it allows the structures within each CPU to be small and fast.

  Since these factors are difficult to quantify, the evaluated superscalar and SMT architectures represent how these systems would perform if it was possible to build an optimal implementation with a fairly shallow pipeline and no clustering, a combination that would result in an unacceptably low clock cycle time in reality. This probably gives the CMP a handicap in our simulations.
- The CPU cores are complicated and composed of many closely interconnected components. As a result, design and verification costs will increase since they must be designed and verified as single, large units.

  The CMP architecture uses a group of small, identical processors. This allows the design and verification costs for a single CPU core to be lower, and amortizes those costs over a larger number of processor cores. It may also be possible to utilize the same core design across a family of processor designs, simply by including more or fewer cores.

With even more advanced IC technologies, the logic, wire, and design complexity advantages will increasingly favor a multiprocessor implementation over a superscalar or SMT implementation.

Memory

A 12-issue superscalar or SMT processor can place large demands on the memory system. For example, to handle load and store instructions quickly enough, the processors would require a large primary data cache with four to six independent ports. The SMT processor requires more bandwidth from the primary cache than the superscalar processor, because its multiple independent threads will typically allow the core to issue more loads and stores in each cycle, some from each thread. To accommodate these accesses, the superscalar and SMT architectures have 128-Kbyte, multibanked primary caches with a two-cycle latency due to the size of the primary caches and the bank interconnection complexity.

The CMP architecture features sixteen 16-Kbyte caches. The eight cores are completely independent and tightly integrated with their individual pairs of caches—another form of clustering, which leads to a simple, high-frequency design for the primary cache system. The small cache size and tight connection to these caches allows single-cycle access. The rest of the memory system remains essentially unchanged, except that the secondary cache controller must add two extra cycles of secondary cache latency to handle requests from multiple processors. To make a shared memory multiprocessor, the data caches could be made write-through, or a MESI (modified, exclusive, shared, and invalid) cache-coherence protocol could be established between the primary data caches. Because the bandwidth to an on-chip cache can easily be made high enough to handle the write-through traffic, we chose that simpler coherence scheme for the CMP. In this way, designers can implement a small-scale multiprocessor with very low interprocessor communication latency.

To provide enough off-chip memory bandwidth for our high-performance processors, we made all simulations with main memory composed of multiple banks of Rambus DRAMs (RDRAMs), attached via multiple Rambus channels to each processor.

Compiler support

The main challenge for the compiler targeting the superscalar processor is finding enough instruction-level parallelism in applications to use a 12-issue processor effectively. Code reordering is fundamentally limited by true data dependencies and control dependencies within a thread of instructions. It is likely that most integer applications will be unable to use a 12-issue processor effectively, even with very aggressive branch prediction and advanced compiler support for exposing instruction-level parallelism. Limit studies with large instruction windows and perfect branch prediction have shown that a maximum of approximately 10–15 instructions per cycle are possible for general-purpose integer applications.[9] Branch mispredictions will reduce this number further in a real processor.

On the other hand, programmers must find thread-level parallelism in order to maximize CMP performance. The SMT also requires programmers to explicitly divide code into threads to get maximum performance, but, unlike the CMP, it can dynamically find more instruction-level parallelism if thread-level parallelism is limited. With current trends in parallelizing compilers, multithreaded operating systems, and the awareness of programmers about how to program parallel computers, however, these problems should prove less daunting in the future. Additionally, having all eight of the CPUs on a single chip allows designers to exploit thread-level parallelism even when threads communicate frequently. This has been a limiting factor on today's multichip multiprocessors, preventing some parallel programs from attaining speedups, but the low communication latencies inherent in a single-chip microarchitecture allow speedup to occur across a wide range of parallelism.[4]

Logic, wire, and design complexity advantages will increasingly favor multiprocessor over superscalar or SMT implementations.

Performance results

We evaluated the performance of each architecture using four representative application workloads running under a realistic simulation environment that included the operating system. For each architecture, we assumed that there is a functional unit of each type (integer, floating point, load/store) for each issue slot, which means that only data dependencies and issue width could prevent an instruction from issuing. The functional unit latencies used in the simulation are similar to those of the recent Silicon Graphics' MIPS R10000.[10] Our simulator properly modeled contention at the primary cache, the secondary cache, and main memory.

The four benchmark programs we ran represent a few large segments of computer usage.

- compress, from SPEC95, represents the field of general integer applications with little instruction-level parallelism and no thread-level parallelism—as a result, we just ran it as a single thread.
- mpeg-2 decode represents the increasing number of multimedia applications found in both desktop and server environments. Like most of these applications, it has significant inherent instruction-level and thread-level parallelism, but moderate memory requirements due to the algorithm's computationally intensive nature. We easily parallelized this application by hand in a manner orthogonal to the fine-grained parallelism exploited by multimedia extensions to instruction set architectures such as Intel's MMX.
- tomcatv, also from SPEC95, represents scientific floating-point applications with large amounts of loop-level parallelism and significant memory bandwidth requirements. The superscalar architecture exploited instruction-level parallelism within it, while the SMT and CMP both benefited from thread-level parallelism automatically found using a compiler.
- multiprogram is an integer multiprogramming workload consisting of several different system simulations, all of them computation-intensive. We ran them as separate processes.

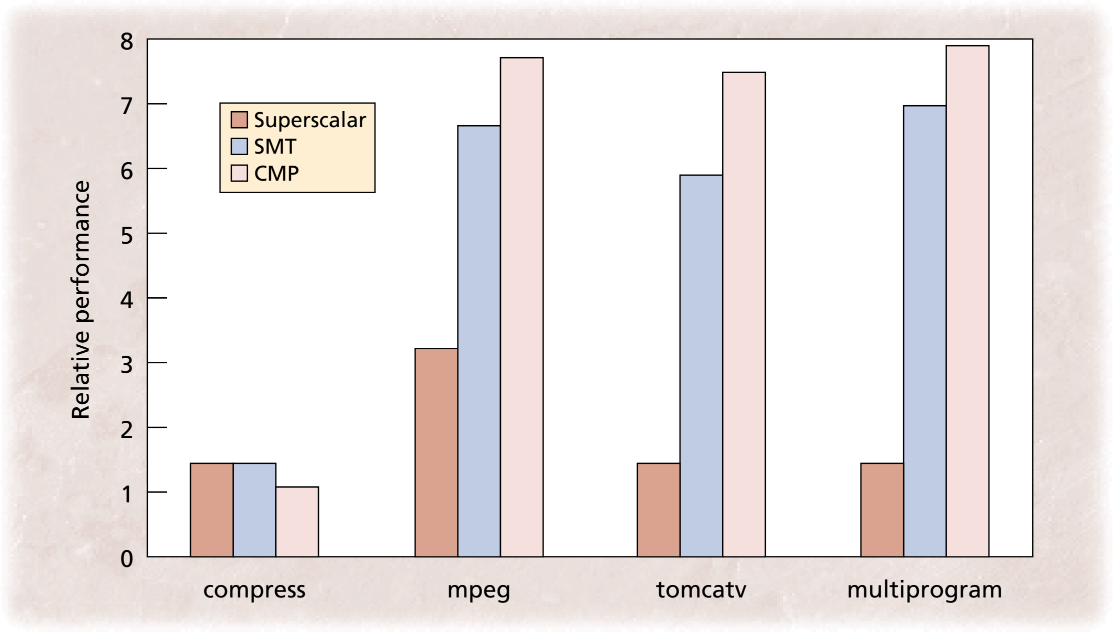

Figure 2. Relative performance of superscalar, simultaneous multithreading, and chip multiprocessor architectures compared to a baseline, 2-issue superscalar architecture.

Figure 2 shows the performance of the superscalar, SMT, and CMP architectures on the four benchmarks relative to a baseline architecture—a single 2-issue processor attached to the superscalar/SMT memory system.

The first two benchmarks show performance on applications with moderate memory behavior and no thread-level parallelism (compress) or large amounts of thread-level parallelism (mpeg). In compress, the wide-issue superscalar and SMT architectures achieved a 43 percent performance gain over the baseline processor, while the single active processor in the CMP was roughly equivalent to the baseline.

Although improved compiler technology may provide better instruction scheduling in the future, it is unlikely to significantly affect the performance of benchmarks such as compress, which is more limited by a lack of instruction-level parallelism due to true data dependencies and unpredictable branches.

The superscalar architecture performed quite well on mpeg, though it was still somewhat limited by the lack of instruction-level parallelism that could be exposed dynamically in the instruction window. The SMT and CMP architectures were able to exploit thread-level parallelism, as well, to operate more efficiently. The SMT utilized its core resources most efficiently, but the CMP achieved higher performance since it had a total of 16 issue slots instead of 12.

Tomcatv has both instruction-level and thread-level parallelism, but it also has extremely high memory bandwidth requirements. The ability to provide this memory bandwidth was a crucial factor in achieving good performance on this benchmark. The superscalar architecture achieved only a 40 percent performance gain over the baseline processor. With only a single thread of control, each cache miss tended to stall the pipeline and waste large numbers of issue slots. A more aggressive compiler could probably have scheduled instructions more efficiently, but multiple threads of control were ultimately more effective at hiding cache-miss latency because the execution of some threads overlapped with the stalls of others.

The CMP experienced a nearly eight-times performance improvement over the single 2-issue processor. The separate primary caches are beneficial because they could be accessed by all processors in parallel. In a separate test with eight processors sharing a single cache, bank contention between accesses from different processors degraded performance significantly. The average memory access time to the primary cache alone went up from 1.1 to 5.7 cycles, mostly due to extra queuing delays at the contended banks, and overall performance dropped 24 percent. In contrast, the shared secondary cache was not a bottleneck in the CMP because it received an order of magnitude fewer accesses.

SMT results showed similar trends. The speedups tracked the CMP results closely when we modeled similar degrees of data cache contention. The nominal performance was similar to that of the CMP's with a single primary cache, and performance improved 17 percent when we temporarily deactivated primary cache contention. The multiple threads of control in the SMT allowed it to exploit thread-level parallelism. Additionally, the dynamic resource allocation in the SMT allowed it to be competitive with the CMP, even though it had fewer total issue slots.

However, tomcatv's memory behavior highlighted a fundamental problem with the SMT architecture: the unified data cache architecture was a bandwidth limitation. Making a data cache with enough banks or ports to keep up with the memory requirements of eight threads requires a more sophisticated crossbar network that will add more latency to every cache access, and may not help if there is a particular bank that is heavily accessed. The CMP's independent data caches avoid this problem but are not possible in an SMT.

As with compress, the multiprogramming workload has limited amounts of instruction-level parallelism, so the speedup of the superscalar architecture was only a 35 percent increase over the baseline processor. Unlike compress, however, the multiprogramming workload had large amounts of process-level parallelism, which both the SMT and CMP exploited effectively. This resulted in a linear eight-times speedup for the CMP. The SMT achieved nearly a seven-times speedup over the 2-issue baseline processor, more than the increase in the number of issue slots would indicate possible, because it efficiently utilized processor resources by interleaving threads cycle by cycle.

The chip multiprocessor is a promising candidate for a billion-transistor architecture. On the basis of our simulations, a CMP offers superior performance using relatively simple hardware. On code that can be parallelized into multiple threads, the many small, high-performance CMP cores working together will perform comparably to—or better than—more complicated wide-issue superscalars or SMTs on a cycle-by-cycle basis. They are also much easier to design and optimize. Additionally, several small, high-performance cores with tightly integrated data caches should allow higher clock rates and/or shorter pipelines in the CMP. SMTs can almost always use execution resources more efficiently than CMPs, but more execution units can be included in a CMP of similar area, since less die area need be devoted to the wide-issue logic. As a result, they perform and utilize area comparably on multithreaded code.

The primary disadvantage of the CMP is that it is slower than the more complex architectures when presented with code that cannot be multithreaded, because only one processor can be targeted to the task. However, a single 2-issue processor on the CMP is usually only moderately slower than superscalar or SMT architectures, since applications with little thread-level parallelism often also lack instruction-level parallelism that can be exploited by superscalar processors. Also, the remaining processors are free to increase system throughput by running independent processes.

We have shown that CMPs are already a strong billion-transistor architecture, even with today's applications and compilers. As application software increases the use of thread- and process-level parallelism, CMPs will become a better solution in the future. Emerging operating environments and applications, such as multimedia, inherently will possess larger amounts of parallelism, and future research into parallel compilers will extend multiprocessors' reach into currently hard-to-parallelize applications. ❖

Acknowledgment

This work was supported by US Defense Advanced Research Projects Agency contract DABT63-95-C-0089.

References

1. Y. Patt, "First Let's Get the Uniprocessor Right," *Microprocessor Report*, Aug. 5, 1996, pp. 23-24.
2. M. Hall et al., "Maximizing Multiprocessor Performance with the SUIF Compiler," *Computer*, Dec. 1996, pp. 84-88.
3. D. Tullsen, S. Eggers, and H. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," *Proc. 22nd Ann. Int'l Symp. Computer Architecture*, ACM Press, New York, 1995, pp. 392-403.
4. K. Olukotun et al., "The Case for a Single Chip Multiprocessor," *Proc. 7th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, ACM Press, New York, 1996, pp. 2-11.
5. G. Sohi, S. Breach, and T. Vijaykumar, "Multiscalar Processors," *Proc. 22nd Ann. Int'l Symp. Computer Architecture*, ACM Press, New York, 1995, pp. 414-425.
6. J. Oplinger et al., *Software and Hardware for Exploiting Speculative Parallelism in Multiprocessors*, Tech. Report CSL-TR-97-715, Computer Systems Laboratory, Stanford Univ., Stanford, Calif., 1997.
7. L. Gwennap, "Digital 21264 Sets New Standard," *Microprocessor Report*, Oct. 28, 1996, pp. 11-16.
8. J.L. Hennessy and D.A. Patterson, *Computer Architecture A Quantitative Approach, 2nd Edition*, Morgan Kaufman, San Francisco, 1996.
9. D.W. Wall, *Limits of Instruction-Level Parallelism*, WRL Research Report 93/6, Digital Western Research Laboratory, Palo Alto, Calif., 1993.
10. K. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, Apr. 1996, pp. 28-40.

Basem A. Nayfeh *is a PhD candidate in electrical engineering at Stanford University. His research interests are parallel computer architecture and software, architectural support for compilers and operating systems, and dynamic compilation. Nayfeh received a BS in electrical engineering from the University of Cincinnati, and an MS in electrical engineering from Stanford University.*

Kunle Olukotun *is an assistant professor of electrical engineering at Stanford University, where he leads the Hydra project (http://www-hydra.stanford.edu). He is interested in the design, analysis and verification of computer systems using ideas from computer and computer-aided design. Olukotun received a BS from Calvin College and an MS and a PhD in computer engineering from the University of Michigan. He is a member of the IEEE and the ACM.*

Contact the authors in care of Kunle Olukotun at Stanford University, Rm. 302, Gates Computer Sci. 3A, Stanford, CA 94305-9030; {lance, bnayfeh, kunle}@ ogun.stanford.edu; http://www-hydra.stanford.edu.