Computer Technology and Architecture: An Evolving Interaction

John L. Hennessy, Stanford University
Norman P. Jouppi, Digital Equipment Corporation

With an eye toward the 1990s, the authors tell how the rapid rate of performance growth of the 1980s can be sustained and discuss challenges for more exotic implementation technologies.

The interaction between computer architecture and integrated circuit technology is complex and bidirectional. The characteristics of various implementation technologies affect decisions architects make by influencing performance, cost, and other system attributes. Developments in computer architecture also impact the viability of different technologies. The implementation requirements of various architectures can emphasize different technology characteristics, such as density, power, and speed.

To understand the interaction between computer architecture and IC technology and the attractiveness of a technology for use in computers, we need a metric to evaluate different computer designs. In designing a computer today, the most important considerations are usually performance and cost. Secondary metrics, which may vary in importance, include fault tolerance, power, and environmental factors such as size, cooling, and noise.

In this article, we discuss an assessment based primarily on performance and cost. Many of the secondary factors are indirectly measured by cost, while others are more difficult to quantify. Furthermore, we focus primarily on CPU performance, both because it is easier to measure and because the impact of technology is most easily seen in the CPU.

To evaluate the suitability of an IC technology, you must consider the technology in the context of a complete CPU. Considering only device-level performance characteristics without accounting for other possible deficiencies only yields misleading results.

We measure CPU performance by the execution time on some suitable work load. The execution time of a program can be expressed as the product of three terms:

- the number of instructions required,
- the average number of clock cycles per instruction, and
- the time per clock cycle.

Since the goal is to maximize performance, a computer designer wants to decrease each of these terms. Unfortunately, these factors are interrelated. The number of

0018-9162/91/0900-0018$01.00 © 1991 IEEE

instructions executed in a program depends primarily on the instruction-set architecture. The other two factors depend on the organization of the machine and the choice of implementation technology, as well as the instruction-set architecture. We will give examples of these interdependencies later.

The importance of this performance equation and the trade-offs among its factors have driven the development of new architectures for the past 10 years. The field of computer architecture has become quantitatively oriented, with comparisons driven by performance and cost. Thus, computer architecture is becoming more engineering and less art.

This shift has not led to a dramatic increase in the number of revolutionary new ideas. Indeed, many of the dominant ideas in computer architecture in the 1980s were old ideas. For example, simplified load/store instruction sets go back to the early supercomputers at Control Data Corp. and Cray Research, while the emphasis on pipelining goes back to machines like Stretch.

However, computer architecture remains challenging because the changing implementation technologies constantly alter the trade-offs, leading to reevaluation of older ideas or the adaptation of existing ideas to a new set of technology assumptions. Dramatic changes in technology, such as significant increases in integration level and decreasing memory cost, have been crucial to the development of many new architectures.

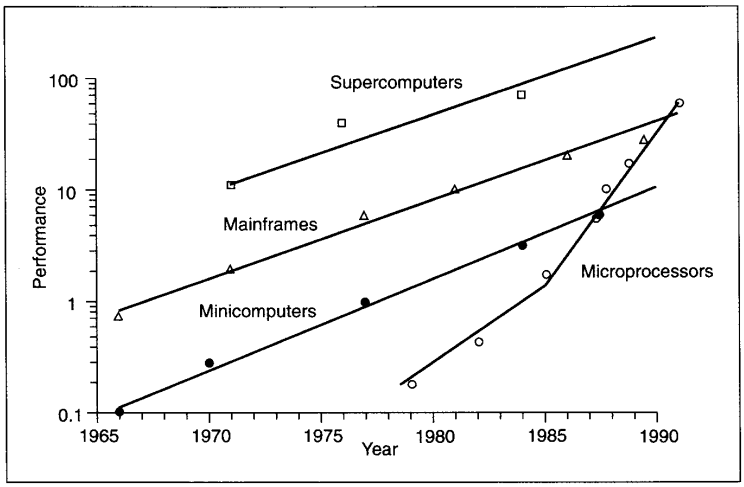


Figure 1. Trends in microprocessor and mainframe CPU performance growth.

Trends in technology and architecture

Recently, rapid IC technology improvements together with computer architecture innovations have led to a rate of CPU performance growth unmatched since the 1950s. Figure 1 shows the rate of improvement in CPU performance as measured by a general-purpose benchmark suite, such as SPEC.

Microprocessor-based machines have been improving in performance at a rate of between 1.5 and 2 times per year during the past six to seven years. Improvement rates for mainframes or minicomputers are about 25 percent per year. (This plot concentrates on general-purpose CPU performance. By ignoring I/O performance and not focusing on any application class, such as scientific, it might not show the most advantageous aspects of larger machines, specifically mainframes and supercomputers.)

Two major factors have driven this high growth rate:

(1) The dramatic increase in the number of transistors available on a chip.

(2) Architectural advances, including the use of RISC ideas, pipelining, and caches.

Because of the ability of microprocessors to rapidly take advantage of improvements in VLSI technology, computers based on microprocessor technology have increased in performance and decreased in cost most rapidly.

The increasing importance of short design times has been another result of this rapid rate of technology improvement and accompanying performance growth. When machine performance is growing rapidly from year to year, new

Glossary

BiCMOS	Bipolar complementary metal-oxide semiconductor
CCD	Charge-coupled device
CISC	Complex instruction-set computing
CMOS	Complementary metal-oxide semiconductor
DRAM	Dynamic random-access memory
ECL	Emitter-coupled logic
FET	Field-effect transistor
GaAs FET	Gallium arsenide field-effect transistor
MESFET	Metal semiconductor field-effect transistor
MOSFET	Metal-oxide semiconductor field-effect transistor
MIMD	Multiple instruction, multiple data
MIPS	Million instructions per second
NMOS	N-Channel metal-oxide semiconductor
PMOS	P-Channel metal-oxide semiconductor
RISC	Reduced instruction-set computing
SIMD	Single instruction, multiple data
SPEC	Systems Performance Evaluation Cooperative
SRAM	Static random-access memory
VLIW	Very long instruction word
VLSI	Very large scale integration

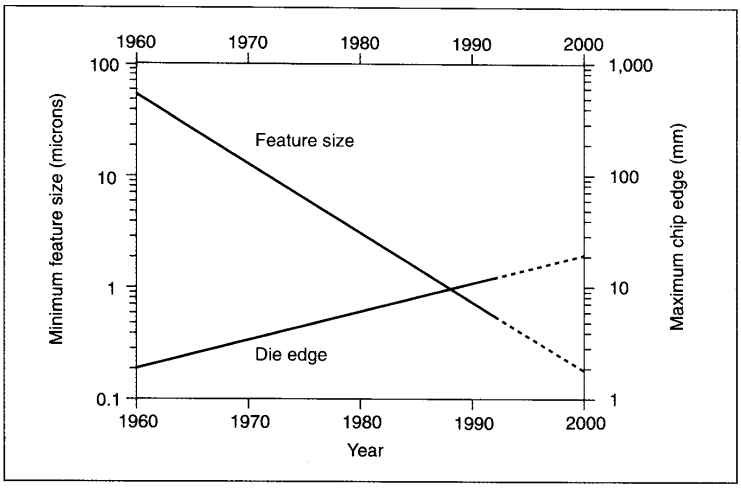


Figure 2. Trends in minimum feature size and maximum chip edge.

computers with long design times must offer substantially larger performance improvements to avoid being bypassed by machines with smaller performance improvements but shorter design times.

Additionally, as the design time of a machine increases, it becomes increasingly difficult to track the implementation technology and ensure that the machine will come to market with the best available technology. These factors mean that the long four- or five-year design cycles common in the mainframe and minicomputer business are becoming a major handicap in competition with machines with shorter design cycles.

Technology trends. By far, the most important technology trend for computers is the relentless decrease in the minimum feature size patternable by optical lithography. Minimum feature sizes have decreased from 50 microns in 1960 to 0.8 μ in 1990 (see Figure 2). Since the number of components per chip is an $O(n^2)$ function of feature size, the number of components per chip has dramatically increased.

Equally important is the fact that circuit characteristics such as switching time and power dissipation also improve with decreasing feature size. In particular, the circuit speed is roughly proportional to the feature size.[1] If we multiply the number of devices by their speed to get an overall measure of computing capability, lithography is responsible for an $O(n^3)$ improvement in computing capability per chip.

The primary challenge facing the computer designer is to translate this huge increase in computing capability into a correspondingly large increase in computing performance. Since the number of devices is increasing as $O(n^2)$ while device performance is only increasing as $O(n)$, effective utilization of ever larger numbers of components is critical.

In parallel with a reduction in feature size, the maximum chip edge dimension (or more precisely, the square root of the maximum chip area) has increased from 2 millimeters in 1960 to 13 mm in 1990. This also has an $O(n^2)$ effect on the number of components per chip. Unlike the effect of scaling minimum feature size on device performance, however, increasing the chip size leaves the basic device characteristics unchanged. Thus, this effect is less important than the continuing reduction in minimum feature size.

In contrast, most other new technologies provide only constant one-time factors. For example, a switch from a silicon CMOS process to a GaAs MESFET process would increase the switching speed of the devices by a fixed amount. Similarly, multichip modules offer a fixed improvement over the potential of single-chip packages.

A significant problem exists when evaluating the benefits of fixed one-time technological improvements in the presence of other $O(n^3)$ trends. If the evaluator does not ensure that the time frame of both technologies is the same, the effects of changes in integration will swamp the effects of the technology change being considered. For example, if an evaluation compares last year's silicon CMOS against this year's GaAs MESFETs, the benefits of GaAs over silicon will be overestimated.

Similarly, if a multichip module improves system performance by 30 percent, but a machine using multichip modules requires six months longer to get into production than a machine using conventional packaging, the $O(n^3)$ effects of integration during the six months may outweigh the benefits of the multichip module.

Memory size. The easiest way to use an $O(n^2)$ increasing number of transistors is in RAM. The first version of Whirlwind with magnetic core storage in 1953 had 4 kilobytes of storage, while the Cray-2 of 1989 has up to 4 gigabytes. Figure 3 shows the increasing storage capacity per RAM chip.[2]

Based on preliminary research, it appears likely that DRAMs will continue to increase in density to at least three orders of magnitude larger than the 256-kilobit DRAMs used in some models of the Cray-2. Over the years, increasing memory size has had a number of effects on computer design, not the least of which is a corresponding increase in the number of bits required to address memory.

Another result of the phenomenal increase in memory capacity has been the appearance of various levels of cache memory. This came about for two reasons. First, if the main memory consists of many RAM chips, the space required to package them will result in a significant amount of time, dictated by signal transmission time, for accessing them. This makes the use of a small, fast cache (or a set of hierarchical caches) attractive to reduce communication delay. Secondly, the large numbers of transistors available on the same chip as a microprocessor now make on-chip as well as off-chip caches feasible.

The ability to place the entire CPU (including its first-level instruction and data caches) on the same chip has a profound effect on microprocessor performance and system cost. Assuming

the caches are of reasonable size, a processor will not need to communicate off-chip to execute most instructions.

Similarly, parallel and very wide buses can be used on-chip without regard to pinout limitations. This largely decouples the CPU cycle time and bandwidth of on-chip processing from off-chip considerations. Hence, the processor can run at a very fast cycle time relative to the frequency of signals at the board level, and the processor can use relatively inexpensive packaging.

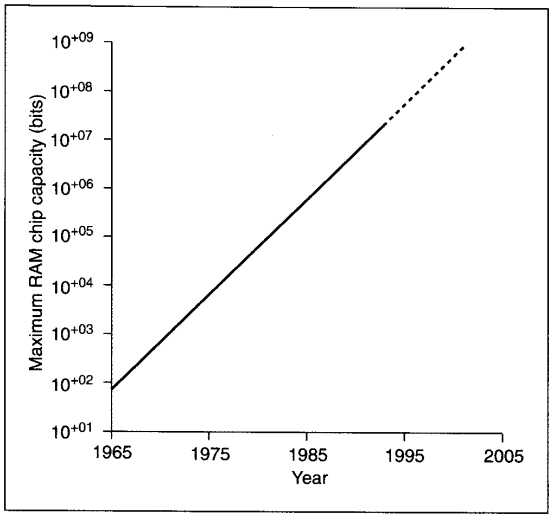


Figure 3. Trends in semiconductor RAM density.

Design complexity and design time. Although increasing memory size is the easiest way to use an $O(n^2)$ increasing number of transistors, increasing the memory in a processor by itself falls short of the $O(n^3)$ increase in computational capacity from lithography. Since the device performance is increasing only as $O(n)$, an approach relying on device performance for speed and increasing memory density for reducing cost would have CPU performance increasing only as $O(n)$. (Application performance could still increase much faster by taking advantage of larger main memories — for example, main memory databases.)

As the recent performance trends in Figure 1 suggest, with lithography the performance increase is much faster than $O(n)$. This added performance has been achieved by using the increasing number of devices as well as their increasing speed.

There are several ways to increase machine performance with relatively little complexity. For example, complete, independent pipelined functional units are often easier to design than single functional units that perform many different operations.

Similarly, increasing the performance of functional units by going from an iterative 2-bits-per-cycle multiplier design to a fully pipelined multiplier array is not much more difficult, but the fully parallel pipelined multiplier uses a much larger transistor count to achieve higher performance.

For simple structures such as multiplier arrays, peak performance can increase roughly with the number of transistors. This approaches the potential $O(n^3)$ increase in computing capability provided by advances in fabrication.

Unfortunately, general-purpose CPU performance cannot be arbitrarily increased by techniques such as increasing the number of bits retired per cycle in a multiplier. Eventually, as each functional unit becomes completely parallel, the only way to further increase performance beyond that provided by increases in component speed consists of more complicated forms of machine parallelism, such as multiple-instruction issue. However, as increasingly complex parallel machine structures are attempted, the critical issue of design time can outweigh the benefits attained from increasing performance through higher CPU complexity.

Efficient design scaling. Since scaling the minimum feature size provides an $O(n^3)$ increase in computational capability per chip, design styles that can easily take advantage of technology scaling have a significant advantage. In a microprocessor, the design can often be moved to a smaller feature size simply by shrinking the design features and performing a small number of minor design modifications (such as redesigning the padframe, since its size is based on bonding capabilities and not feature sizes).

In contrast, advances in base technology are much harder to exploit in a multichip design. Much of the work in designing a multichip machine involves partitioning the design based largely on the number of pins available per chip. A decrease in feature size by a factor of 2 might have very little effect on the performance of a multichip machine, unless the partitioning is changed, essentially requiring the redesign of the whole processor.

Also, since 50 percent of multichip-machine cycle time is typically spent in chip-to-chip communication, a factor of 2 increase in device speed with the same partitioning may only increase overall machine performance by (0.5/2 + 0.5)/1 = 0.75/1 = 33 percent. This magnitude of per-processor performance increase is typical for "mid-life kickers" in multichip machines. Any further increases in multichip-machine performance would likely require a complete redesign and/or more advanced multichip packaging technology.

In contrast, microprocessor designs can scale to much larger increases in performance. For example, initial MIPS R2000 chips operated at 8 megahertz. Recently, R3000 chips based on the same design core that operate at 40 MHz have become available, giving a 400 percent increase in performance over the original design. This ability of microprocessor-based designs to scale effectively in performance with improving base technology will become even more pronounced as cache hierarchies migrate on chip.

Trends in architecture. The design of a new computer is strongly affected by both how the machine is to implemented and how it is to be used. The ongoing developments in IC technology provide the strongest influence in the implementation. Thus, the strong growth in the level of integration has a significant impact on the choice of implementation techniques.

Two of the most important hardware techniques used to improve performance during the past decade have been pipelining and caches. Both techniques rely on using more devices to achieve higher performance. Of course, the techniques

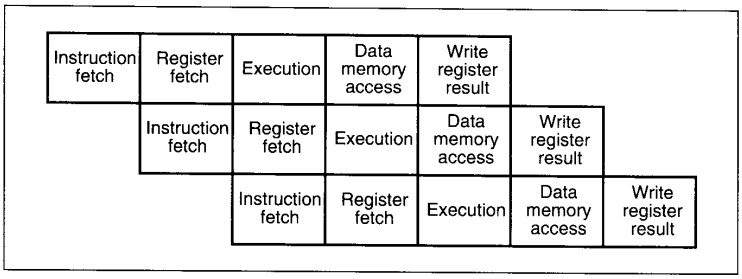


Figure 4. A simple machine pipeline.

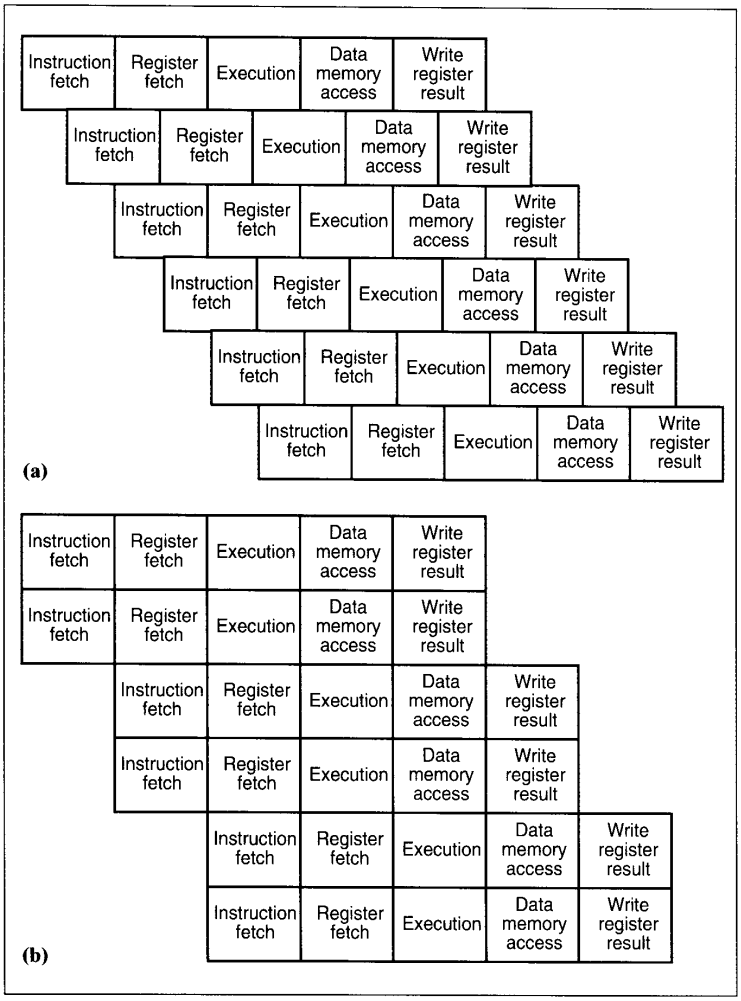


Figure 5. Superpipelined (a) and superscalar (b) machine pipelines.

have been used in mainframes and supercomputers for some time. However, as the integration levels have increased, it has been possible to use these techniques aggressively in microprocessor-based machines. In fact, a number of microprocessor-based machines have used pipelining and cache techniques that are more advanced than those in use in the biggest machines.

Pipelining. At the beginning of the 1980s, pipelining was used almost exclusively by mainframes and supercomputers. By the end of the decade, pipelining was used by *every* new machine.

Pipelining improves the throughput of a machine without changing the basic cycle time, as Figure 4 shows. Thus, pipelining is an especially useful technique when the gate count available is growing faster than gate speed. Of course, improvements in the gate speed can be independently incorporated into changes in the cycle time.

In the early 1980s, most machines had little or no pipelining and the average instruction took 5 to 10 cycles (that is, the cycles per instruction was 5 to 10). By the end of the decade, several RISC machines achieved CPI values close to 1 for integer code. For the basic pipeline shown in Figure 4, a machine with a CPI of 1 essentially represents an ideal value.

Pipelining increases performance by exploiting instruction-level parallelism. Instruction-level parallelism is available when instructions in a sequence are independent and thus can be executed in parallel by overlapping. In the ideal case, a pipelined machine can complete an instruction every cycle. One of the requirements to achieve this is an amount of instruction-level parallelism that is sufficient to keep the pipeline full of independent instructions.

To improve performance beyond the level achievable with the ideal pipeline above, we can either make the pipeline deeper, called superpipelining, or issue more than one instruction per clock into the pipeline, called superscalar. Figure 5 shows the two approaches. Because either approach can exploit more instruction-level parallelism, either can achieve higher performance, assuming enough parallelism is available in the application.

A superpipelined machine and a superscalar machine of the same degree achieve roughly the same performance: They are essentially duals.[3] However, secondary design choices in the implementation of either technique can have substantial performance impact, and each approach has different technology trade-offs. We discuss the trade-offs for each technique in the section entitled "Improving uniprocessor performance."

In addition, because instruction-level parallelism is often the limiting factor to the speedup achieved by these tech-

niques, there is a trade-off between the depth of a pipeline and the instruction issue rate. That is, if the latency of the functional units is made deeper, the instruction issue rate will drop.

This is not surprising; the parallelism used in the deeper pipeline is the same parallelism used in issuing multiple instructions. However, this trade-off has an interesting outcome. With a limited silicon budget, an architect must often choose between the bandwidth of a functional unit and its latency. Often, designers focus on the bandwidth, trying to design a unit that can issue a new operation on every clock. Alternatively, designing a unit with lower latency may be possible, at the cost of lowering the issue bandwidth. Emphasizing the reduction in latency, rather than the peak bandwidth, might lead to a faster machine.

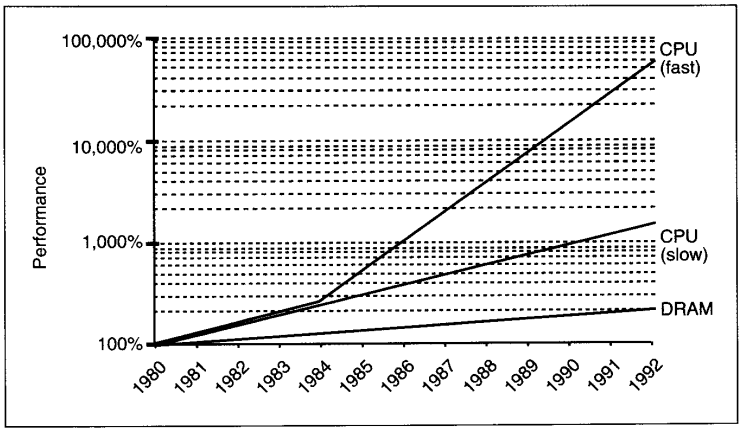


Figure 6. Trends in DRAM and processor cycle times.[4]

Memory systems. As the integration level has increased, memory has become progressively less expensive. This has had several important effects on computer design, including the decreased importance of conserving memory. In fact, designers now look for ways to trade-off memory for speed and reduced design complexity. RISC architectures, for example, use more memory than CISC architectures, but reduce the design complexity and offer higher speed. This trade-off was not attractive until the advent of large, relatively cheap DRAMs.

The improvements in IC technology affected not only DRAMs, but also SRAMs, making the cost of caches much lower. Caches are one of the most important ideas in computer architecture because they can substantially improve performance by the use of memory.

The growing gap between DRAM cycle times and processor cycle times, as Figure 6 shows, is a key motivation for caches.[4] If we are to run processors at the speeds they are capable of, we must have higher speed memories to provide data.

Caches were first used in the late 1960s and early 1970s, both in large machines and minicomputers. In the past few years, virtually every microprocessor has included support for a cache.

Although large caches can certainly improve performance, the cache design problem is one of technology-driven compromises. These compromises exist because each of the three main cache design parameters — total cache size, associativity, and block size — has optimal values that depend on the details of a design.

To understand these trade-offs, it is useful to examine the equation for average memory access time (assuming hit and miss detection time are the same):

Average memory access time
= hit time + miss rate * miss penalty

where the miss penalty is the time to handle a cache miss, including an access to memory. Because cache access time is nearly always the critical path in a processor design, the cache hit time often determines the processor cycle time, making the hit time critical.

Each of the three primary architectural parameters for a cache affects two factors in the above equation. Increasing the total cache size, increasing the associativity, or increasing the block size (to a limit) will decrease the miss rate. However, increasing the total cache size or the associativity often requires that the access time increase.

As a number of designers have realized, the increase in hit time may cost more than the improvement in miss rate.[5,6] Increasing the block size also increases the miss penalty, and at some point (when the increase in miss penalty exceeds the savings from an increased hit rate) will cause the average memory access time to increase.

Because there is an optimal design point for each of these three cache parameters (as determined by the average memory access time), there is an unavoidable miss rate, which cannot be zero.[5] As the gap between processor cycle time and DRAM access time increases, the impact of the remaining misses increases. That is, as processors get faster they will lose more and more of their performance to the memory system.

One technique to alleviate this problem is to introduce another level of caching between the primary cache and main memory. This idea of multilevel caches has recently been used both in high-end machines, where the allotted hit time limited the size of the primary cache, and in machines where the primary caches are on the processor chip and are limited in size by the amount of silicon available. Because the access time is less critical, these secondary caches can also be built using slower high-density RAMs, achieving sizes in the range of hundreds of kilobytes to megabytes of total cache.

Multiprocessors. The role of multiprocessors grew dramatically in the past decade as designers attempted to build on the dramatic price-performance advantages offered by microprocessors. Today, microprocessor-based multiprocessors can offer aggregate performance in the hundreds of millions of instructions per second.

While a number of research projects are focused on building scalable multiprocessors (scalable to hundreds or thousands of processors) using microprocessor technology, most of the focus in

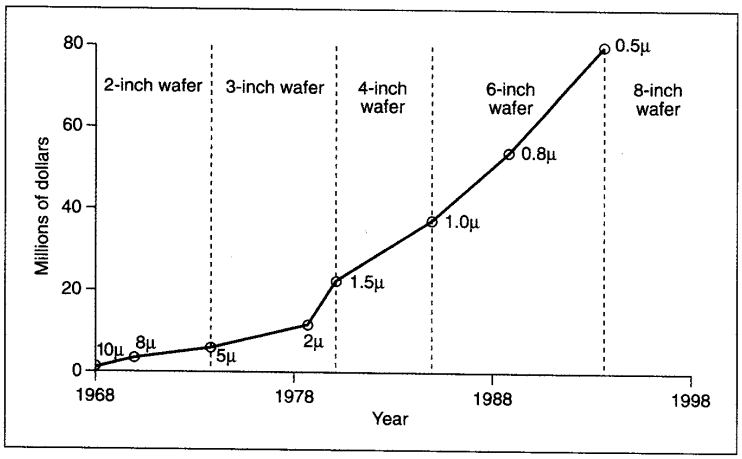


Figure 7. Semiconductor fabrication line capital cost per thousand wafers per week. (μ = microns)

industry is on building small (4-20 processors), bus-based machines that support cache coherency.

These machines have found a role in three different marketplaces:

(1) They are an ideal alternative to timesharing uniprocessors — each user can be assigned to a single processor.
(2) They make good transaction processing systems, using the parallelism among transactions.
(3) They are being used as parallel processors in scientific and engineering applications.

In all three cases, rapid improvements in microprocessor performance have been critical to maximizing the usefulness of the machines.

For ease of programming, these machines have been designed with cache coherency. This simplifies the development of both system software and user applications. The cache coherency algorithms employed rely on using a bus for broadcast and implement a snoopy protocol.[7]

In addition to using microprocessor technology, these machines are based on several critical technology developments. To accommodate several high-performance processors on a memory bus, the architecture must reduce the bus demands of each processor, which is accomplished through the use of very large caches. The use of large caches and a bus interconnect is interdependent: Large caches allow multiple processors to share a bus and memory system, and a bus supports the use of a simple snooping coherency protocol to keep the caches consistent. In the next section, we examine the challenges facing architects as they attempt to scale these machines to more processors and higher performance processors.

Opportunities and challenges

Continuation of the rapid increase in computer performance seen over the past decade will require ongoing improvements in today's technologies as well as the development of new IC technologies and new ideas in computer architecture. Two important forces that will probably accelerate in the 1990s are the importance of design time and the reduction of differences in CPU performance between the smallest and the largest machines.

Technology challenges. Two of the most significant technology challenges in the 1990s will be

(1) continuing to increase integration at historic rates and
(2) developing new base technologies with higher performance, in parallel with the continued increase in integration.

Integration. In the remainder of the decade, a number of technological and economic challenges will face our industry as a result of ever-increasing levels of integration. One obvious technical challenge is the ability to continue patterning finer and finer geometry with optical lithography.

Over the past several years, the imminent demise of optical lithography has been regularly forecast due to feature sizes smaller than the wavelength of the light used — requiring a switch to E-beam or X-ray lithography. However, the trend to smaller feature sizes patterned by optical lithography has continued unabated, primarily through the use of shorter and shorter wavelengths for exposure. Recent advances, such as phase-shift masks, allow feature sizes almost as small as the exposure wavelength to be patterned.

Once the features are patterned, they must be converted into device structures. This will remain very challenging as feature sizes continue to decrease, but operational MOS transistors with gate lengths less than 0.1 microns have already been fabricated in research labs with E-beam equipment.

At the current pace of feature size reduction, feature sizes of 0.2 microns will be common by the end of the decade (see Figure 2). Many of the most challenging problems will arise while fabricating the interconnect layers, rather than the devices. In many modern processes, interconnect already accounts for more masks than device fabrication. This will increase further as limits such as the required current density (which increases with reducing feature sizes) become more and more significant.

Although many approaches may be technologically feasible, they must also be economically feasible to be applied in a market economy. Gordon Moore stated in his invited address at Compcon Spring in 1989 that one undesired result of relentless minimum feature-size scaling is that the cost of building a new fabrication line capable of building a fixed number of wafers is increasing roughly proportional to the inverse feature size. This is shown in Figure 7.

Ignoring the trend to larger wafer sizes and larger die sizes, which roughly cancel each other out, the capital cost required to build an IC of decreasing feature size is increasing as $O(n)$. How-

ever, since the functionality provided by these chips is increasing as $O(n^3)$, the cost per function will still continue to decrease radically. Nonetheless, the increasing fab cost will mean that fewer companies will be able to afford a state-of-the-art line.

Design complexity is another potential problem. However, the level at which ICs are designed has been increasing steadily, from the days of cutting rubylith to the current widespread adoption of logic synthesis techniques. This has greatly reduced the design time per transistor.

Although custom design techniques were emphasized in university VLSI classes in the 1980s, the percentage of chips designed today that are full-custom designs is very small. Even high-volume designs such as the Intel i386 microprocessor contain control logic implemented in standard cells. Further adoption of synthesis techniques should continue at higher levels until at least the mid-1990s.

One of the most visible results of increasing integration is the compression in the range of performance attainable by different implementation techniques. For example, in 1974, the performance of the Intel 8080 was less than 0.1 8-bit MIPS, whereas the IBM 306/168 performed at 3 32-bit MIPS. This is a ratio of more than 100 to 1 in performance (including word size).

Recent microprocessor-based machines have similar peak CPU performance ratios compared to mainframes. As a result, the major determinant of machine performance has become the memory and I/O subsystem, and not the CPU used. For example, workstations are built from microprocessors with relatively little memory and small I/O systems. Minicomputers, such as the VAX 6000 line, have become microprocessor-based machines with more memory and larger I/O subsystems.

A single CPU design will soon dominate the entire range of performance from PCs to mainframes. Obviously, this has major implications for CPU designers. But more importantly, it now means that applications from one performance domain or computing style will be able to run in other domains and computing styles.

Emerging base technologies. Most changes in technology are evolutionary rather than revolutionary. Since the early 1970s, for example, the primary technology used for designing microprocessors has evolved from PMOS to NMOS to CMOS. During this time, several revolutionary technologies such as CCD and magnetic bubble memories that were pursued with great interest have largely disappeared into special-purpose niches. In this section, we consider tradeoffs when using BiCMOS, GaAs FETs, and advanced packaging for implementing computers in the 1990s. More speculative technologies, such as Josephson Junctions and optical computing, are beyond the scope of this article.

When comparing the performance of different technologies, you must be fair. One common mistake is to ignore fan-in and fan-out limitations. For example, a single bipolar ECL gate can combine a NOR gate, multiplexor, latch, and several inverters in one level of logic. In another technology with a faster basic NOR gate, but with fan-in and fan-out limited to 3, the same function will take several levels of logic. This difference in gate functionality may more than negate any speed advantage for technologies with limited gate functionality and fan-out.

BiCMOS is rapidly gaining acceptance for a number of reasons. First, in many cases it is a minor process enhancement to an already existing CMOS process. Bipolar transistors have several advantages over MOSFETs. Their conductance is an exponential function of their base voltage, whereas FET conductance is proportional to the square of the applied gate voltage. This property of bipolar transistors allows them to sense small signal swings much faster than MOS transistors. Thus, one major initial application of bipolar transistors in BiCMOS processes has been in the sense amplifiers of RAM chips.

Another property of bipolar transistors is that they have superior load-driving capability as compared to MOSFETs. This capability has been exploited in many papers that discuss some form of CMOS logic gate coupled with a bipolar output buffer. This application is especially effective in gate-array or standard-cell design styles, where the integration is limited by wiring density (rather than gate density) and performance is limited by long, heavily loaded wires.

The performance improvement claimed by GaAs FETs in the 1980s over CMOS may be largely eroded when compared to BiCMOS processes of the 1990s. Moreover, any technology that competes with a rapidly improving mainline technology having immense volume and applied expertise faces significant challenges. Another early advantage of GaAs substrates was that they have lower capacitance than non-insulating silicon substrates. However, to the extent that circuit performance is limited by wiring delays, the fact that modern VLSI processes have three and four levels of wiring means that capacitance is dominated by capacitance to other wires, and capacitance to the substrate is only a minor factor. Finally, many GaAs circuit families have relatively low gate functionality in comparison to ECL or even CMOS circuit families.

Recently, multichip packaging technology has been receiving widespread interest. In contrast to the ever-increasing computational capacity provided by improvements in lithography, advanced packaging improves system performance by a fixed, constant amount.

To take advantage of the increased performance allowable by advanced interconnection technology, the processor organization must be designed in tandem with the interconnection technology. This can lead to significant problems.

Most multichip packaging techniques improve system performance by only a modest percentage, for example, 30 percent. This percentage may be expected to decrease in the future as more and more functionality is integrated per chip and interchip communication becomes less and less frequent. For example, a microprocessor with no on-chip caches would benefit from faster interchip interconnections on every instruction and data reference. However, a CPU chip with large on-chip caches would only benefit for those few instructions that incur cache misses.

Since integration is rapidly increasing, any slip in machine schedule due to increased machine complexity from the use of multichip modules could easily negate any performance advantage gained by the module.

Finally, multichip modules are more expensive than traditional plastic packaging on PC boards. The challenge for multichip packaging design is to limit the cost of packaging and prevent the use of multichip packaging from delay-

ing machine production. Otherwise the combination of cost and small marginal improvement (in performance versus delivery date) may limit the adoption of multichip packaging techniques to supercomputers and special-purpose applications.

Architectural challenges. In this section, we look at the challenges for building higher performance machines and particularly focus on the most promising directions for maintaining the dramatic improvements of the past decade. We focus on three distinct challenges for computer architects: increasing the speed of uniprocessor machines, building efficient small-to-moderate-scale multiprocessors, and dealing with the growing access time gaps among processors, memories, and I/O systems. As you will see, several of these challenges involve other fundamental problems that must be overcome in the next decade.

Improving uniprocessor performance. Most of the improvements in single processor performance are likely to come from increasing the instruction throughput, either by lowering clock rates or clocks per instruction. Tracking technology closely and taking advantage of new technology improvements will be critical to decreasing the clock cycle.

Lowering the CPI has been a major factor in improving performance in the past 10 years; we expect this to continue. However, the problem becomes more challenging for several reasons stemming from the structure of programs, the implementation hurdles for the various multiple-issue techniques, and other factors that gain importance as the peak instruction throughput increases.

Perhaps the most serious limitations to exploiting more instruction-level parallelism arises from a lack of uniform instruction-level parallelism in programs. Several in-depth studies of the difficulty in exploiting instruction-level parallelism have been carried out, focusing on both general-purpose and scientific programs. This distinction is quite important.

In many scientific programs, including all vectorizable programs, there is a large amount of data-level parallelism (that is, the parallelism is proportional to the size of the data in the application). Such programs have essentially unlimited parallelism, at least in portions of the code. What remains unclear is the best method for exploiting such parallelism. In addition to the multiple instruction issue, both vector machines and SIMD machines provide important methods for exploiting such parallelism.

Uniformity in the number of instructions that can be issued at any point is critical.

Two major studies of more scalar-oriented programs[8,9] have shown the practical level of exploitable parallelism to be around 2 operations per clock. Uniformity in the number of instructions that can be issued at any point is also critical. If the number varies widely, building a CPU to exploit the peaks in the available parallelism will be both ineffective, because of Amdahl's law effects, and inefficient, since much of the machine will be idle most of the time.

Uniformity in the type of instructions used by an application is also critical. All superscalar machines built to date restrict the combination of instructions that can be issued in one clock. For example, a two-way superscalar machine might allow a floating-point operation to be issued together with any other nonfloating-point instruction. Then, if the floating-point instructions do not constitute a uniform 50 percent of the instructions executed in the application, the machine will be underutilized and the speedup will be less than ideal.

Less restricted issue of instruction combinations (as in most superpipelined machines) or the use of dynamic hardware scheduling[10] can help alleviate these performance limitations. The impact of issue restrictions can be crucial — some studies[3] show that machines with lower issue rates can have higher performance if the issue restrictions are less severe. These imbalances together with restrictions on the total instruction-level parallelism available[3,9] have led to typical improvements from these techniques of about 1.5 to 2 times, with nonvectorizable code under realistic design assumptions.

Both the superscalar and superpipelined techniques introduce implementation challenges that need to be overcome. In a superscalar machine, some of the functional units must be duplicated, with the number increasing if issue restrictions are to be minimized. In addition, as the number of instruction issues per cycle increases, we will need to increase the number of register ports.

Because memory references constitute a significant fraction of the instructions, a superscalar machine that wants to issue more than a few instructions per cycle will need to have multiple data-cache ports. All these factors require much larger numbers of gates and more interconnect, which can be the limiting factor in the short term.

In the longer term, the challenge will be to not impede the clock rate while adding these extra units or access ports. Possibly the limiting challenge to increasing the instruction issue rate for a superscalar machine is the difficulty in fetching, decoding, and issuing an ever-larger number of instructions in the same clock cycle. The CPU must check for dependencies among a set of instructions before issuing them in a single cycle; the number of combinations to check grows as n^2 and limits the issue rate. The VLIW approach solves some of these problems, as we discuss shortly.

Superpipelining is essentially the extension of pipelining and thus is limited by the same factors that limit attempts to increase the depth of a pipeline without limit. The primary problem in this regard is the impact of skew, both in data and control.

The skew, which is determined by the technology and the skill of the designers in balancing data and clock paths, usually cannot be scaled down arbitrarily small and, instead, grows as a fraction of the clock as the clock rate is increased. Because of skew, there is a limit to how short the pipeline stages can be before the allowance for skew consumes an unacceptable fraction of the clock period.

A VLIW is a multiple-issue machine in which the compiler takes the responsibility for finding the operations that can be issued together and creating a single instruction containing those operations. Because the compiler does this statically, the hardware does not need to check dependences or issue restric-

tions. The compiler schedules the operations into instructions to avoid these problems.

VLIWs face a similar set of challenges. The question of how much instruction-level parallelism can be exploited statically by a compiler remains open. While VLIWs have demonstrated an ability to do this on vector codes, we need to investigate the question of their usefulness for more general-purpose applications, especially those with memory-access patterns that are not predictable at compile time.

Many of the implementation challenges of a superscalar that tries to issue a large number of instructions in a single cycle apply here. Most particularly, providing many register ports and memory ports becomes a challenge. For this reason, several VLIWs have chosen to avoid the use of data caches and instead build multiported memories that offer higher bandwidth with much longer access latencies. How well this works for a wide range of applications remains to be seen.

In the future, we can expect designers to combine the ideas of superscalar and superpipelining to get the best of each approach. The major challenge for all these approaches is to find ways to uncover and exploit larger amounts of instruction-level parallelism.

One idea that has demonstrated some potential is the ability to do speculative computation. In speculative computation, an operation is optimistically executed before we know whether we will really need its results. This effectively ignores control dependencies, and hence makes highly accurate branch prediction critical.

Wall[11] showed that the amount of parallelism that can be extracted rises substantially, with perfect branch prediction and alias resolution (determining whether two memory addresses refer to the same location). Whether speculative techniques will pay off in practice depends on their implementation cost and future progress on branch prediction and alias resolution.

Challenges in memory system design. Memory systems present machine designers a range of challenges. To design high-performance uniprocessors and multiprocessors, we need sophisticated memory hierarchies that keep the average access time low and provide high bandwidth. Specifically, the challenges lie in three areas:

Some factors indicate that the growth of memory on systems might accelerate.

(1) Memory hierarchies for high-performance processors
(2) Large memories and address spaces
(3) Multiprocessors

Both the latency and the bandwidth of a memory hierarchy become more crucial as performance is increased. The relative importance of low average memory access time increases as performance increases, both because of an increase in the clock rate of the CPU and a decrease in the CPI.

Increasing the CPU clock rate makes the cache miss penalty rise because of the slow rate of improvement in DRAM access times relative to CPU cycle times. In addition, as CPI decreases with the use of multiple-instruction-issue techniques, the proportional effect of a nonzero memory access penalty continues to increase.

Likewise, both of these changes increase the bandwidth requirements on the memory hierarchy. As the clock rate increases in a pipelined machine, the rate at which memory accesses must occur increases linearly. In addition, to maximize the performance benefit of multiple instruction issue and to achieve any benefit beyond a small number of issues per clock, we must be able to increase the number of memory accesses issued per clock. Thus, designers must build memory hierarchies capable of lower effective access times and higher bandwidth.

Cache designers must also try to alleviate the impact of cache misses, whose cost continues to increase as machines get faster. One important technique for decreasing the cost of a miss is a nonblocking cache (also called a lockup-free cache).[12] A nonblocking cache allows other memory accesses to occur while a miss is being handled.

Some machines today have limited support for nonblocking caches. However, the importance of this idea is likely to increase. There are many unresolved issues in the design of systems using nonblocking caches. For example: How many accesses need to be outstanding to achieve the benefits? How much dynamic scheduling of memory-referencing instructions do we need?

Over the past 30 years, the demand for memory address space in computer systems has grown at a rate of between 1/2 bit and 1 bit per year; the growth has been closer to 1 bit per year for the past 15 years. A variety of factors have led to this growth, including the use of high-level languages, added functionality in applications and operating systems, and the trade-off of memory space for CPU cycles by both applications and operating systems.

These trends in growth of memory usage will apparently continue. Indeed, some factors indicate that the growth of memory on systems might even accelerate.

One such factor is the growing gap between CPU speeds and access time to disk. For example, in the past 10 years, average disk access time has improved by less than a factor of 2, while CPU clock rates have increased by more than one order of magnitude and CPU performance has increased by more than two orders of magnitude.

One way to lower the effect of this gap is to aggressively use main memory as a file or disk cache for the I/O system. Because the I/O system has a much larger capacity, the amount of memory we need for caches will be large.

As the access time gap grows, the memory must grow to lower the miss rate. This trend could easily lead to massive memories on systems within five years. Similar pressures exist on memory use for other high-bandwidth I/O systems, such as video and fiber-optic networks.

The use of memory as a cache or buffer, together with the ongoing growth of functionality in software (for example, mapped files), will put pressure on address space limitations. Most architectures today offer 32-bit address spaces. Examining the growth of address space over the past 20 years makes it clear that a variety of environments will require more address space within the next few years. A simple extension of the growth rate curves from the 1970s,

when the 16-bit address space was exhausted, shows the need for more address space in two to three years.

Extending the address space of existing architectures is one of the most difficult design problems architects face. As Gordon Bell once said: "There is one mistake that can be made in computer design that is difficult to recover from: not having enough address bits for memory addressing and memory management."[13]

Some architects have attempted to provide a solution to this problem by adding segmentation to their architecture. However, this only provides a limited solution, since it still requires that each segment be confined to 32 bits.

The compiler and programming difficulties of dealing with a segmented address space significantly limit the usefulness of the extended address space, although some important applications can be accommodated. Experience has shown that machines with a flat, unsegmented address space are more flexible than those with a segmented address space. This leads to the conclusion that architectures will shortly need to provide a larger word size to allow them to easily address and manipulate objects in address spaces larger than 32 bits.

The next natural step is 64 bits — since many machines already handle data objects of that size for floating-point data, though few provide complete support for 64-bit integers. Making this transition will be a major challenge for many architectures. We might need to develop new instruction-set architectures, just as was done when we moved from 16-bit addresses to 32-bit addresses.

The good news is that if demand for growth in the address space does not further accelerate, a 64-bit address space will last a very long time.

Multiprocessors. Four types of challenges surface in building multiprocessors:

(1) The increasing difficulty of building small-scale machines using ever-faster processors.

(2) The challenges that arise when designing machines that scale beyond the small machines that can use a bus interconnect.

(3) The unsolved problems in efficiently and easily programming multiprocessors.

(4) The current lack of quantitative data and insight on the behavior of parallel programs forces the architect to design a machine while lacking sufficient insight into the critical properties of programs that will run on the machine.

This last problem is exacerbated because the properties that are critical to performance are often high-level application characteristics and might depend intimately on how an application was initially written. This lack of data and understanding will only be solved by the development of many more parallel applications, which can then be studied.

As microprocessors get faster, the design of multiprocessors based on those microprocessors becomes more challenging. Consider the task of redesigning a four-processor bus-based machine as the CPU performance goes from 25 MIPS to 100 MIPS, including a factor of 2 in clock rate. The bandwidth demands may nearly quadruple, although larger caches may partially alleviate this bandwidth growth. If the cache miss penalties are not to grow by a factor of 2, the access time to memory over the bus must also improve by a factor of 2. These factors conspire to make the design ever more difficult as the CPUs get faster.

To build next-generation bus-based multiprocessors, we need high-speed, very wide buses using low-overhead protocols and high clock rates (possibly with reduced voltage swing). Likewise, the difficulty of implementing snoopy caches increases, since the cache latency and bandwidth must be larger to accommodate the demands of the faster processor.

To build larger multiprocessors that provide a single uniform address space requires that we use new cache coherency algorithms if we are to maintain the advantages of transparently caching shared data. These new algorithms cannot use the snoopy approach, which is based on broadcasting to every processor. Directory-based coherency algorithms[14,15] seem to offer the best alternative, although machines using these algorithms are still in the prototype stage.

Surely the biggest obstacle to the widespread, general use of multiprocessors is the programming difficulty. This difficulty arises from the challenges of writing programs that run efficiently. To date, this has required detailed knowledge of the hardware design. This complicates the programming problem substantially and, because such details are incorporated into the program to achieve performance, the portability of parallel programs is extremely low. If multiprocessors are to gain broad acceptance for parallel processing, we must simplify the programming task and make it less machine dependent.

During the 1980s, the benefits of strong interaction between computer architecture and IC technology became clear. Architects developed new instruction sets and new organizations specifically driven by technology developments. The role of computer systems as technology users has also shaped the direction of IC technology development to an unprecedented degree. This relationship is likely to become even stronger in the next 10 years. The biggest challenge facing designers may be to understand the breadth of issues from software to IC technology that will affect computers. ■

Acknowledgments

The referees provided many helpful comments that improved this article.

References

1. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass., 1980, p. 35.

2. C.-T. Sah, "Evolution of the MOS Transistor — From Conception to VLSI," *Proc. IEEE*, Vol. 76, No. 10, Oct. 1988, pp. 1,280-1,326.

3. N.P. Jouppi, "The Nonuniform Distribution of Instruction-Level and Machine Parallelism and its Effect on Performance," *IEEE Trans. Computers*, Vol. 38, No. 12, Dec. 1989, pp. 1,645-1,658.

4. J.L. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, San Mateo, Calif., 1990.

5. S.A. Przybylski, *Cache Design: A Performance-Directed Approach*, Morgan Kaufmann, San Mateo, Calif., 1990.

6. M.D. Hill, "A Case for Direct-Mapped Caches," *Computer*, Vol. 21, No. 12, Dec. 1988, pp. 25-40.

7. J.R. Goodman, "Using Cache Memory to Reduce Processor Memory Traffic," *Proc. 10th Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., Order No. 473 (microfiche only), 1983, pp. 124-131.

8. N.P. Jouppi and D.W. Wall, "Available Instruction-Level Parallelism for Superpipelined and Superscalar Machines," *Third Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, IEEE CS Press, Los Alamitos, Calif., Order No. 1936, 1989, pp. 272-282.

9. M.D. Smith, M. Johnson, and M.A. Horowitz, "Limits on Multiple Instruction Issue," *Third Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, IEEE CS Press, Los Alamitos, Calif., Order No. 1936, 1989, pp. 290-302.

10. H.B. Bakoglu et al., "IBM Second-Generation RISC Machine Organization," *Proc. Int'l Conf. Computer Design*, IEEE CS Press, Los Alamitos, Calif., Order No. 1971, 1989, pp. 138-142.

11. D.W. Wall, "Limits of Instruction-Level Parallelism," *Proc. Fourth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, ACM, 1991, pp. 176-188.

12. G. Sohi and M. Franklin, "High-Bandwidth Data Memory Systems for Superscalar Processors," *Proc. Fourth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, ACM, 1991, pp. 53-62.

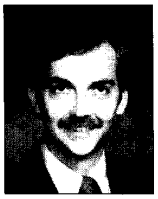
13. W.D. Strecker and C.G. Bell, "Computer Structures: What Have We Learned From the PDP-11?" *Proc. Third Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., Order No. 099 (microfiche only), 1976, p. 187.

14. D. Chaiken et al., "Directory-Based Cache Coherence in Large-Scale Multiprocessors," *Computer*, Vol. 23, No. 6, June 1990, pp. 49-58.

15. D.V. James et al., "Scalable Coherent Interface," *Computer*, Vol. 23, No. 6, June 1990, pp. 74-77.

John L. Hennessy is a professor of electrical engineering and computer science at Stanford University. During a leave from Stanford in 1984-85, he cofounded MIPS Computer Systems, where he continues to participate in industrializing the RISC concept as chief scientist. His research interests lie in exploiting parallelism at all levels to build higher performance computer systems.

Hennessy received the Presidential Young Investigator Award in 1984, was named the Williard and Inez K. Bell Professor of Electrical Engineering and Computer Science in 1984, was elected an IEEE fellow in 1991, and is a member of the IEEE Computer Society.



Norman P. Jouppi has been a member of the research staff of the Digital Equipment Corporation's Western Research Lab and a consulting assistant professor in the Electical Engineering Department of Stanford University since 1984. His research interests include computer architecture, bipolar VLSI design, and VLSI CAD tools.

Jouppi received the BSEE and MSEE degrees from Northwestern University, Evanston, Ill., in 1979 and 1980, respectively, and the PhD in electrical engineering from Stanford University in 1984. He was a National Science Foundation fellow from 1980 to 1983.

Readers can contact Hennessy at the Department of Electrical Engineering, CIS 208, Stanford University, Stanford, CA 94305, and Jouppi at Digital Equipment Corporation, 250 University Ave., Palo Alto, CA 94301.