

---

# DEEP-SUBMICRON MICROPROCESSOR DESIGN ISSUES

---

TO FULLY EXPLOIT SHRINKING FEATURE SIZES AND AVOID BEING OVERWHELMED BY COMPLEXITY, MICROPROCESSOR DESIGNERS MUST KEEP UP WITH TECHNOLOGY TRENDS, UNDERSTAND SPECIFIC APPLICATIONS, AND USE ADVANCED CAD TOOLS.

..... Deep-submicron technology allows billions of transistors on a single die, potentially running at gigahertz frequencies. According to Semiconductor Industry Association (SIA) projections,<sup>1</sup> the number of transistors per chip and the local clock frequencies for high-performance microprocessors will continue to grow exponentially in the near future, as Figure 1 illustrates. This ensures that future microprocessors will become ever more complex.

However, physical and program behavioral constraints will limit the usefulness of this complexity. Physical constraints include interconnect and device limits, as well as practical limits on power and cost. Program behavioral constraints result from program control and data dependencies, and from unpredictable events during execution.

Other challenges include the need for advanced CAD tools to combat the negative effect of greater complexity on design time. Designers will also have to make improvements to preserve computational integrity, reliability, and diagnostic features. Successful implementations will depend on the processor architect's ability to foresee technology trends and understand the changing design trade-offs for specific applications, beginning with the differing requirements for client ver-

sus server processors. This article discusses these trade-offs in light of industry projections and the many considerations affecting deep-submicron technology.

## Limits of technology scaling

Improved microprocessor performance results largely from technology scaling, which lets designers increase the level of integration at higher clock frequencies. While current implementations use feature sizes of about 0.25 micron, devices with feature sizes smaller than 0.1 micron are expected in the next few years. Table 1 shows projected specifications through 2012. As feature sizes shrink, device area shrinks roughly as the square of the scaling factor. Meanwhile, device propagation delay (under constant field assumptions) improves linearly with the decrease in feature size.

Nevertheless, designers face several major technical challenges in the deep-submicron era. The most important is that interconnect delay (especially global interconnect delay) does not scale with feature size. If all three dimensions of a wire are scaled down by the same scaling factor, the interconnect delay remains roughly unchanged. This is because the fringing field component of wire capacitance does not vary with feature size.<sup>2</sup> Thus,

Michael J. Flynn  
Patrick Hung  
Kevin W. Rudd  
Stanford University

**Table 1. Semiconductor Industry Association roadmap summary for high-end processors.**

Specification/year	1997	1999	2001	2003	2006	2009	2012
Feature size (micron)	0.25	0.18	0.15	0.13	0.1	0.07	0.05
Supply voltage (V)	1.8-2.5	1.5-1.8	1.2-1.5	1.2-1.5	0.9-1.2	0.6-0.9	0.5-0.6
Transistors/chip	11M	21M	40M	76M	200M	520M	1.4B
DRAM bits/chip	167M	1.07G	1.7G	4.29G	17.2G	68.7G	275G
Die size (mm <sup>2</sup> )	300	340	385	430	520	620	750
Local clock freq. (MHz)	750	1,250	1,500	2,100	3,500	6,000	10,000
Global clock freq. (MHz)	750	1,200	1,400	1,600	2,000	2,500	3,000
Maximum power/chip (W)	70	90	110	130	160	170	175

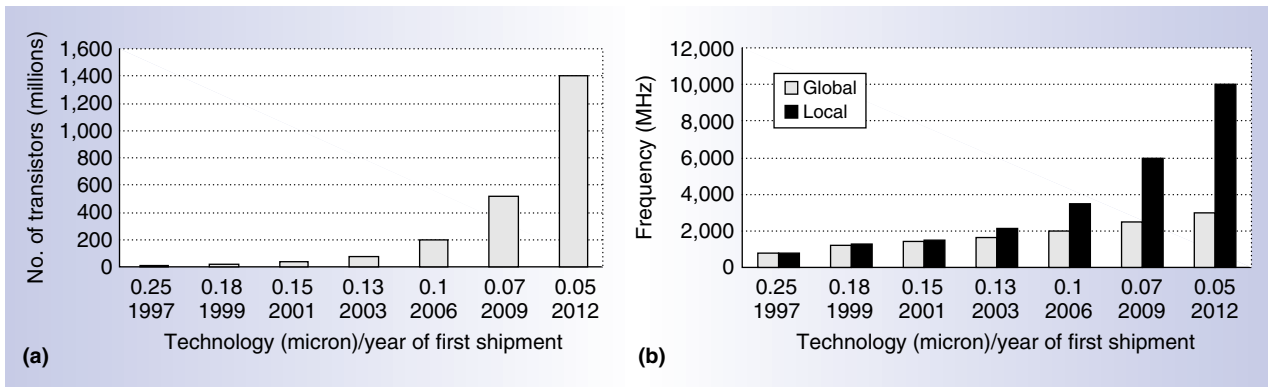


Figure 1. *The National Technology Roadmap for Semiconductors*: (a) total transistors per chip, (b) on-chip local clock.

the interconnect delay decreases far less rapidly than the gate delay and proves more significant in the deep-submicron region.

Increasingly, interconnect delays rather than device delays dominate functional unit designs. This may cause architects and designers to rethink the algorithms used to implement various functional units. Introducing copper metallization and low  $k$  dielectric insulators helps reduce interconnect delay, but these one-time improvements will not suffice in the long run as feature size continues to shrink.

In an effort to minimize interconnect resistance, modern designs scale interconnect height at a much slower pace than interconnect width. Consequently, the aspect ratio ( $T/W$  in Figure 2) should rise gradually from 1.8 at present to 3.0 by the year 2012.<sup>1</sup> This shift reduces wire resistance but also increases the effects of line coupling, from 20% at 0.7 micron to 80% at 0.18 micron. Cross talk between adjacent wires will pose a more serious problem, and wire congestion will ultimately determine interconnect delay and power dissipation. Implementations that use

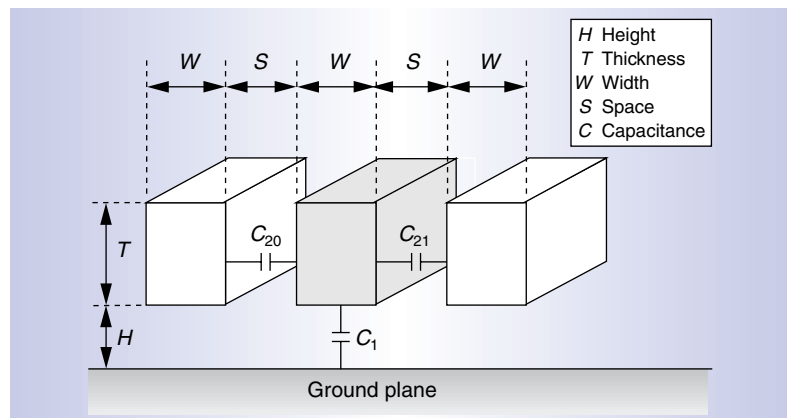


Figure 2. Interconnect capacitances.

more devices in critical paths yet offer less wire congestion and shorter interconnect delays may be preferable to older implementations that simply focus on reducing the number of gate delays.

It is interesting to compare the current SIA projections with those from 1994,<sup>1</sup> which predicted far more modest cycle time improvements. Note that the earlier projections were

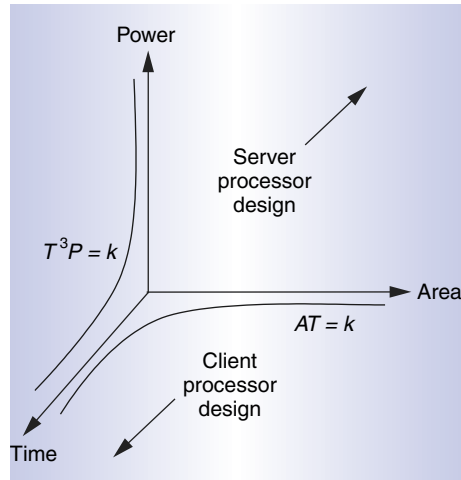


Figure 3. Area, performance (time), and power trade-off trends in server and client processor designs.

generally based on scaling alone. By 1997-98, it became obvious that 1-GHz processors were feasible immediately—not between 2005 and 2010, as projected earlier. The revisions arise from coordinated improvements in scaling, circuit design, and CAD tools.

Feature sizes below 0.1 micron lead to other technical challenges. High-performance processors need special cooling techniques, which could consume as much as 175 W of power. Enabling GHz signals to travel into and out of the chips requires new circuit designs and algorithms. Preventing latch-up and reducing noise coupling may require new materials such as silicon-on-insulator. Similarly, reducing cross talk and DRAM leakage may require low  $k$  as well as high  $k$  materials, respectively. These challenges demand that designers provide whole-system solutions instead of treating logic design, circuit design, and packaging as independent phases of the design process.

### Client versus server processors

In the era of deep-submicron technology, two classes of microprocessors are evolving: client and server processors. The majority of implementations are commodity system-on-chip client processors (including network and embedded processors) devoted to end-user applications such as personal computers. These highly cost-sensitive processors are used extensively in consumer electronics. Individual applications may have specific require-

ments; for example, portable and wireless applications require very low power consumption. The other class consists of high-end server processors, which are performance driven. Here, other parts of the system dominate cost and power issues.

At a fixed feature size, area can be traded off for time. VLSI complexity theorists<sup>3</sup> have shown that an  $AT^n$  bound exists for microprocessor designs, where  $n$  usually falls between 1 and 2. By varying the supply voltage, as we show later, it is also possible to trade off area  $A$  for power  $P$  with a  $PT^3$  bound. Figure 3 shows the possible trade-off involving area, time  $T$ , and power in a processor design. Client and server processors operate in different design regions of this three-dimensional space. The power and area axes are typically optimized for client processors, whereas the time axis is typically optimized for server processors. Until recently, a single design point existed for both servers and clients, based on economy of scale. With added computational resources (in area and time) becoming available through improved feature sizes, the movement to establish separate design points—differentiated by power—seems inevitable.

An important dimension—computational integrity—does not appear in Figure 3. Later we discuss its nature and effects on area, time, and power.

### Time considerations

Microprocessor performance has improved by approximately 50% per year for the past 15 years. This can be attributed to higher clock frequencies, deeper pipelines, and improved exploitation of instruction-level parallelism (ILP). In the deep-submicron era, we can expect performance improvement to result largely from reducing cycle time at the expense of greater power consumption.

#### Cycle time

Processor clock frequencies have increased by approximately 30% per year for the past 15 years, due partly to faster transistors and partly to fewer logic gates per cycle. Traditionally, digital designs have used edge-triggered flip-flops extensively. Such a system's cycle time  $T_c$  is determined by  $T_c = P_{\max} + C$ , where  $P_{\max}$  is the maximum delay required for the combinational logic, and  $C$  is the total

clock overhead, including setup time, clock-to-output delay, and clock skew. For high-end server processors, the SIA predicts that the clock cycle time will decrease from roughly 16 FO4 (fanout-of-four) inverter delays at present to roughly five FO4 inverter delays at 0.05-micron feature size. As a result, clock overhead takes a significant fraction of the cycle time, and flip-flop clocking systems appear infeasible. Fortunately, a number of circuit techniques can improve deep-submicron microprocessor cycle times:

- Several new flip-flop structures, such as sense-amplifier-based, hybrid latch, and semidynamic flip-flop, have been proposed to lower clock overhead.
- Asynchronous logic<sup>4</sup> eliminates the need for a global clock. Average latency depends on  $P_{\text{mean}}$  (average logic delay) instead of  $P_{\text{max}}$  (maximum logic delay), but completion detection and data initialization incur significant overhead.
- Various forms of dynamic logic<sup>5</sup> have been proposed to minimize the effects of clock skew. These techniques reduce clock overhead at the expense of power and scalability.
- Wave pipelining<sup>6</sup> uses  $P_{\text{min}}$  (minimum logic delay) as a storage element to improve cycle time.

We use wave pipelining to illustrate some of the new clocking considerations. For memories and other functional blocks that contain regular interconnect structures, wave pipelining is an attractive choice. The technique relies on the delay inherent in combinatorial logic circuits. Suppose a given logic unit has a maximum interlatch delay of  $P_{\text{max}}$  and a corresponding minimum delay of  $P_{\text{min}}$  with clock overhead  $C$ . Then the fastest achievable cycle time  $\Delta t$  equals  $P_{\text{max}} - P_{\text{min}} + C$ .

As with conventionally clocked systems, system clock rate  $T_c$  is the maximum  $\Delta t_i$  over  $i$  latched stages. Sophisticated CAD tools can ensure balanced path delays and thus improve cycle time. In practice, using special CAD tools lets us set  $P_{\text{min}}$  to within about 80% to 90% of  $(P_{\text{max}} + C)$ . While this would seem to imply clock speedup of more than five times the maximum clock using traditional clocking schemes, environmental issues such as

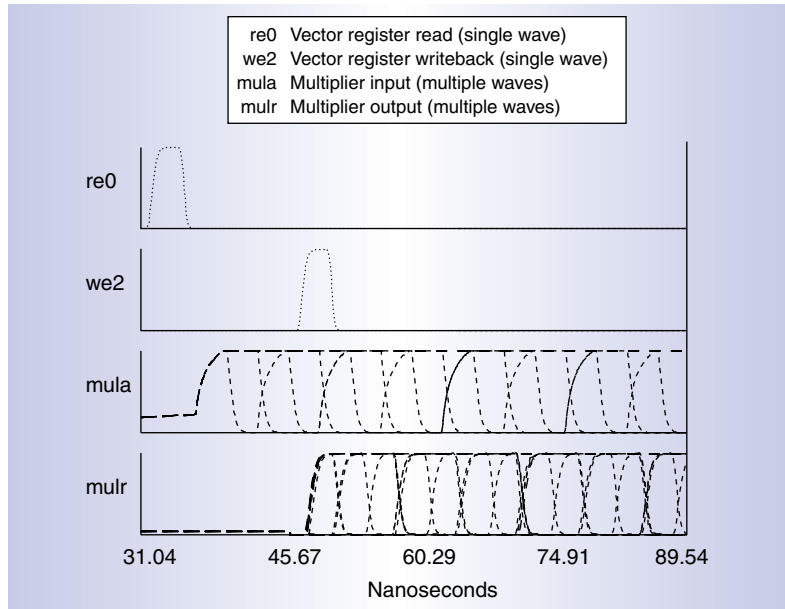


Figure 4. Wave-pipelined vector multiplication.

process variation and temperature gradient across a die restrict realizable clock rate speedup to about three times.<sup>7,8</sup> Figure 4 details the register-to-register waveforms of a wave-pipelined vector multiplier.<sup>7</sup> The achieved rate shown at the bottom of the figure is more than three times faster than the traditional rate determined by the latency between the multiplier input and output (shown in the top two segments of Figure 4).

Wave pipelining also exemplifies how aggressive new techniques offer architects and implementers both benefits and challenges. Although allowing significant clock-rate improvements over traditional pipelines, wave pipelines cannot be stalled without losing the in-flight computations. Because individual waves in the pipeline exist only by virtue of circuit delays, they cannot be controlled between pipeline latches. In the case of wave pipelines, architecturally transparent replay buffers<sup>9</sup> can provide the effect of a stall and extend the applicability of wave pipelining to applications that require stalling the pipeline. Other new techniques may not fit in directly with current architectures and may also require special treatment to be generally applicable.

### Performance versus complexity

Processors with faster cycle times often provide better overall performance, although clock

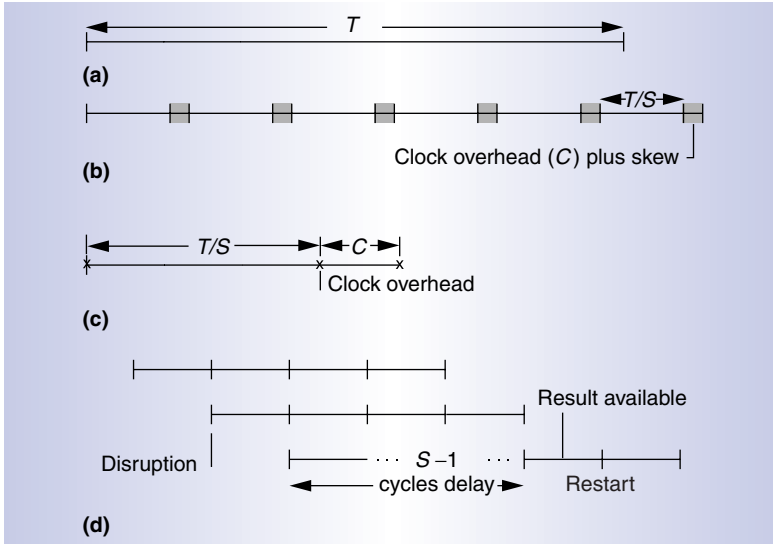


Figure 5. Optimum pipeline stages.

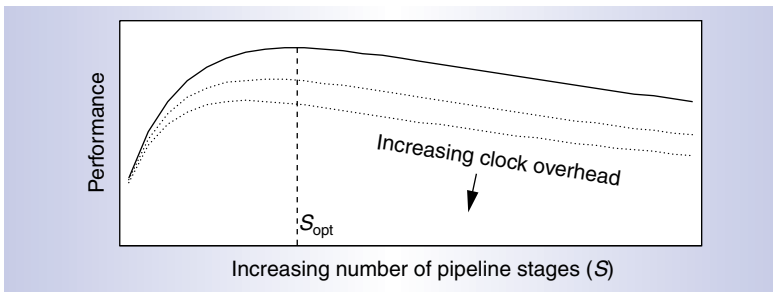


Figure 6. Performance versus number of pipeline stages, given different levels of clock overhead.

overhead and pipeline disruptions during execution may nullify this advantage. Consider the case of a simple pipelined processor, illustrated in Figure 5. Assume that the total time to execute an instruction without clock overhead is  $T$  (Figure 5a). Now,  $T$  is segmented into  $S$  segments to allow pipelining (Figures 5b and 5c). The pipeline completes one instruction per cycle if no interruptions occur; however, disruptions such as unexpected branches result in flushing and restarting the pipeline (Figure 5d). Suppose these interruptions occur with frequency  $b$  and have the effect of invalidating  $S - 1$  instructions. The pipeline throughput  $G$  becomes

$$G = \frac{1}{1 + (S - 1)b} \times \frac{1}{T/S + C}$$

Differentiating  $G$  with respect to  $S$  lets us

determine the optimum number of stages  $S_{opt}$ , as shown by Dubey and Flynn:<sup>10</sup>

$$S_{opt} = \sqrt{\frac{(1 - b)T}{bC}}$$

Intuitively, if we make the cycle time too large and have too few stages in the pipeline, we sacrifice overall performance by not overlapping execution sufficiently. On the other hand, if we make the cycle time too small, we sacrifice overall performance by incurring too much clock overhead and suffering long pipeline breaks. Figure 6 shows the relationship between performance and number of pipeline stages.

Selecting the optimum level of ILP entails an analogous trade-off. The algorithm or application imposes the fundamental limit to the available ILP: you can never exploit more parallelism than the application makes available. This maximal ILP declines with code generation inefficiencies, processor and system resource limitations, and execution disturbances. Suppose the processor can issue a maximum of  $N$  instructions per cycle. Because of control and data dependencies, a fetched instruction often cannot be issued and must wait for prior instructions to complete. Assume the probability that an instruction cannot be issued is proportional to the number of active instructions in the pipeline and that adding one more instruction width to an ILP processor increases the overall execution time by a fraction of  $O$ . If  $N$  operations are issued per cycle, and  $d$  is the average latency of instructions in a scalar processor, we calculate the processor's instructions per cycle (IPC) as

$$IPC = \frac{N}{1 + N \cdot d \cdot (1 + O \cdot N)}$$

as shown by Hung and Flynn.<sup>11</sup> Figure 7 shows the relationship between IPC and ILP ( $N$ ) on the basis of the above equation. By differentiating IPC with respect to  $N$ , we can determine the optimum  $N$  by

$$N_{opt} \approx \sqrt{\frac{1}{d \cdot O}}$$

In Figure 8 we use the MXS superscalar simulator<sup>12</sup> to calculate the IPC of a super-

scalar processor running the Compress benchmark. In this graph, normalized performance is defined as  $IPC \cdot (1 - O \cdot N)$ . Consider the case when the overhead  $O$  is 3%. Because of ILP overhead, increasing the instruction width greater than five times diminishes the actual performance.

The trade-off between VLIW (very large instruction word) and superscalar machines amounts to reducing overhead with a VLIW machine versus reducing disruption with a superscalar machine. Latency tolerance reduces average latency  $d$  and hence improves the ILP available.

For either processor approach (fast  $\Delta t$  or increased ILP), performance is bounded by implementation overhead and program behavior. We can use our understanding of algorithms to extend both of these limits.

### Disruptions

Execution disruptions include execution interruptions (for example, from mispredicted branches or operation exceptions) and result delays (say, from extra computation cycles or cache misses). They can be predicted with varying degrees of accuracy; branches can often be predicted in the high 90% range. However, mispredictions of these events require expensive corrective actions. For example, after a mispredicted branch, it may be necessary to flush pipelines, recover the register state, and restart execution along the correct path.

A large variance in penalties accompanies execution disruptions. A simple mischeduled operation may disrupt only a single cycle; however, a cache miss can disrupt upward of hundreds of cycles in a high-performance system. Thus, any technique whose application can improve the prediction rate or minimize the penalty can yield significant performance benefits.

The two primary ways<sup>9</sup> to manage execution disruptions involve statically scheduled VLIW processors with a software approach or dynamically scheduled superscalar processors with a hardware approach.

The software approach relies on the compiler to prevent as many disruptions as possible by eliminating the possibility of nonerror disruptions in the code schedule. Even system interrupts can be managed in this manner, although this solution is less applicable to gen-

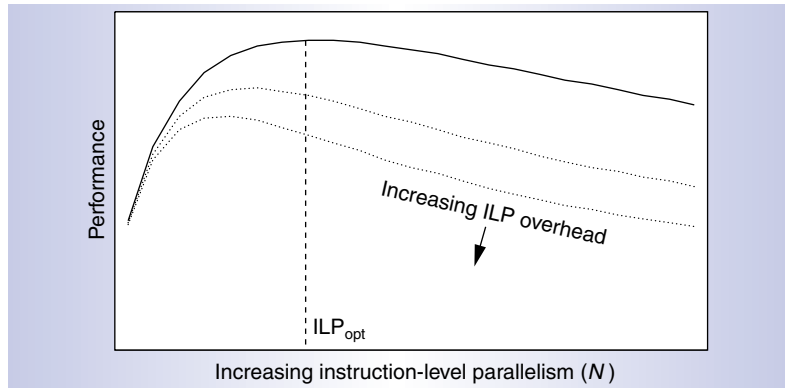


Figure 7. Performance in instructions per cycle versus instruction-level parallelism, given different levels of ILP overhead.

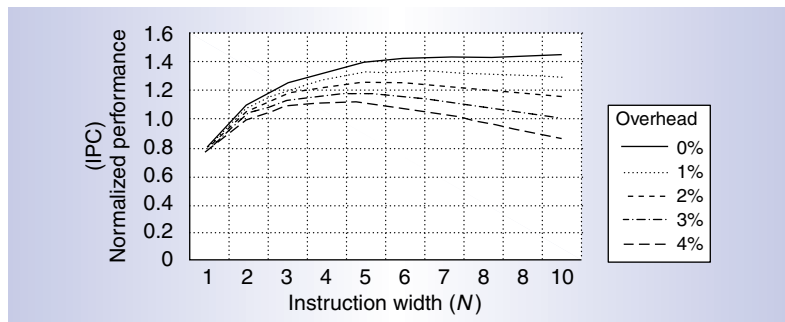


Figure 8. Normalized performance versus instruction-level parallelism (Compress benchmark).

eral-purpose computing. The software approach minimizes hardware complexity, providing both cost and performance benefits; however, the requirement to pessimistically schedule the code often results in suboptimal performance.

The hardware approach relies on hardware to predict, detect, and execute correctly in the presence of disruptions. Hardware-based dynamic prediction (driven by recent execution patterns) can achieve greater accuracy than compiler-based branch prediction (driven by either heuristics or profile information). However, with the complexity of dynamic prediction hardware (say, for branch prediction or data prefetching), this approach rapidly reaches the point of diminishing returns.

### The "memory wall" obstacle

Currently, popular processors have extended techniques such as branch prediction, data and instruction caches, and out-of-order execution to enable greater performance. Unfor-

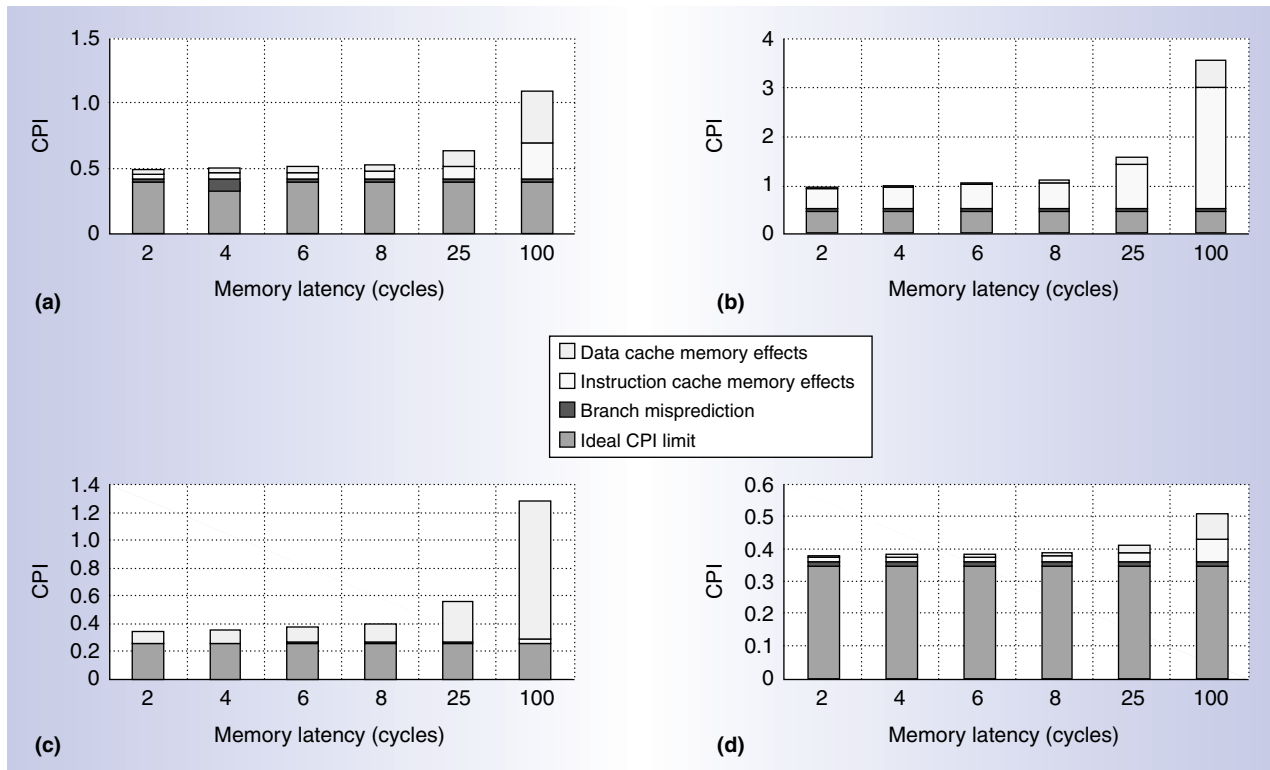


Figure 9. Performance breakdown showing cycles per instruction (CPI) versus memory latency on the four benchmarks (a) espresso, (b) li, (c) alvinn, and (d) ear for an eight-wide VLIW processor with varying memory system penalties.

unately, each of these techniques is reaching the point of diminishing performance returns. Like branch prediction, cache memory already achieves a prediction rate in the high 90% range, and performance is now dominated by the ever-increasing processor-memory speed mismatch.

Figure 9 shows the performance breakdown for an eight-instruction-wide VLIW processor. As memory latency increases, memory-related penalties (the so-called memory wall) rapidly begin to dominate performance.<sup>13</sup> Branch prediction reduces branch penalties, but the memory penalty still dominates. Related results (not shown in the figure) indicate that little would be gained by adding out-of-order-execution capability to such a processor. Despite the effectiveness of hardware approaches, until the memory wall can be controlled, the memory penalty will continue to dominate performance.

### Power considerations

Growing demands for wireless and portable electronic appliances have focused much attention recently on power consumption.

The SIA<sup>1</sup> points to increasingly higher power for microprocessor chips because of their higher operating frequency, higher overall capacitance, and larger size.

At the device level, total power dissipation ( $P_{\text{total}}$ ) has three major sources—switching loss, leakage current, and short-circuit current:<sup>14</sup>

$$P_{\text{total}} = \frac{C \cdot V^2 \cdot \text{freq}}{2} + I_{\text{leakage}} \cdot V + I_{\text{sc}} \cdot V$$

where  $C$  is the device capacitance,  $V$  is the supply voltage,  $\text{freq}$  is the device switching frequency,  $I_{\text{leakage}}$  is the leakage current, and  $I_{\text{sc}}$  is the short-circuit current. Of the three power dissipation sources, switching loss remains dominant.

We can reduce switching loss by lowering the supply voltage. Chen et al.<sup>15</sup> showed that the drain current is proportional to  $(V - V_{\text{th}})^{1.25}$ , where  $V$  is the supply voltage and  $V_{\text{th}}$  is the threshold voltage. If we keep the original design but scale the supply voltage, the parasitic capacitances remain the same, and

the maximum operating frequency is proportional to  $(V - V_{th})^{1.25} / V$ .

Figure 10 shows the relationship between maximum operating frequency and supply voltage. In this figure, we assume  $V_{th}$  is 0.6 V. Between 1- and 3-V supply voltage, the operating frequency is roughly proportional to the supply voltage. Reducing the supply voltage by half also reduces the operating frequency by half, and the total power consumption becomes 1/8 of the original. Thus, if we take an existing design optimized for frequency and modify that design to operate at a lower voltage, the frequency is reduced by approximately the cube root of the original power:

$$\frac{freq_2}{freq_1} \approx \sqrt[3]{\frac{P_2}{P_1}}$$

It is important to understand the distinction between scaling the frequency of an existing design and that of a power-optimized implementation. Power-optimized implementations differ from performance-optimized implementations in several ways. Power-optimized implementations use less chip area not only because of reduced requirements for power supply and clock distributions, but also, and more importantly, because of reduced performance targets. Performance-oriented microprocessor designs consume a great deal of area to achieve marginally improved performance—very large floating-point units, minimum-skew clock distribution networks, or maximally sized caches.

Power dissipation, not performance, is the most critical issue for applications such as portable and wireless client processors running on batteries. Conventional nickel-cadmium battery technology has been replaced by high-energy-density nickel metal hydride (NiMH) and lithium ion technology. However, for safety reasons, energy density is unlikely to improve dramatically in the near future.

For client processors to run on battery power for an extended period, the entire system's power consumption must remain very small (on the order of a microwatt). Sub-threshold circuits operate below  $V_{th}$ ; they offer an ideal way to reduce both the switching loss and the leakage current.<sup>16</sup> If a significant portion of the chip supports communication and other DSP functions, then exploiting the nat-

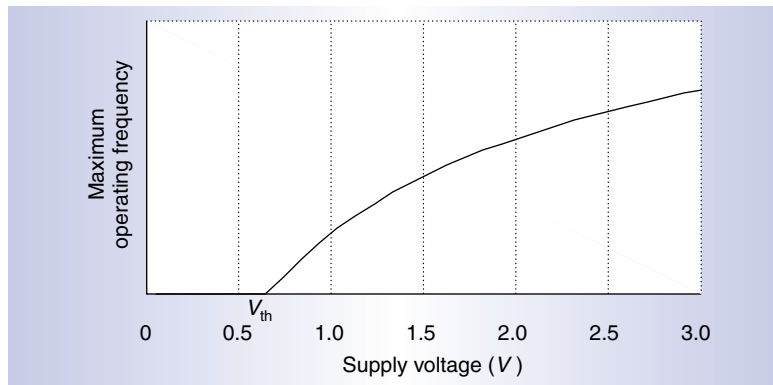


Figure 10. Maximum operating frequency versus supply voltage, given a threshold voltage of 0.6 V.

ural parallelism in these applications (for example, using subword-parallel multimedia instructions) can result in acceptable performance even with a very slow clock frequency.

In the future, to save power, many client processor systems will eliminate disks. Thus, programs will have to be downloaded from server to client memory using wireless or wired communications. Depending on the applications, the DRAM may be located on or off chip. We know that integrating the CPU with embedded DRAM is problematic because the current DRAM process would yield low-performance logic circuits. However, embedded DRAM is ideal for subthreshold logic circuits, which run at a very low clock frequency.

Self-timed and asynchronous circuits have been implemented in the past mainly to boost performance. Because of signaling overhead, asynchronous circuits in processors are undesirable. However, asynchronous circuits may serve a different purpose in the future. These circuits do not require a global clock and can help reduce total power consumption.

Amdahl's law states that the performance gained by improving only part of a computer system is limited by the fraction of time the system uses that particular part. The same principle extends to power dissipation: power reduction achieved by improving only part of a computer system is limited by the fraction of power dissipation attributed to that particular part. If the chip accounts for 20% of the system's total power consumption, reducing the chip power by 50% reduces the total power consumption by only 10%. As a result, power management must be implemented

**Table 2. FUPA components and recently announced results (for FPUs only).**

Processor	Effective latency (ns)	Normalized area (mm <sup>2</sup> )	Normalized effective latency (ns)	FUPA (cm <sup>2</sup> ns)
DEC 21164	12.38	77.51	35.36	27.41
MIPS R10000	14.25	70.08	40.71	28.53
HP PA8000	24.44	81.16	48.89	39.68
Intel P6	28.25	51.56	80.71	41.61
Sun UltraSparc	23.65	133.43	50.32	67.15
AMD K5	66.00	48.42	188.57	91.32

from the system architecture and operating system down to the logic gate level.

### Area considerations

Another important design trade-off entails determining the optimum die size.<sup>17</sup> In the server market, the processor may be a relatively small component in a much more costly system dominated by memory and storage costs. In this design area, an increase in processor cost of 10 times (from, say, a die cost of \$10 to a die cost of \$100) may not significantly affect the overall system's cost. However, even when cost per se is not a design factor, efficient layout and space usage are increasingly important in determining cycle time. McFarland<sup>18</sup> has shown that at 0.1-micron feature size and 1-ns cycle time, a long interconnect delay may make it difficult (or impossible) to have one-cycle cache accesses for caches larger than 32 Kbytes. Using the same assumptions, he suggests that the limit for two-cycle cache access is 128 Kbytes. Thus, even in cost-insensitive server designs, cycle time may still limit chip complexity.

Client processor implementations are extremely cost sensitive, and optimum area use is very important. As technology and costs allow, we will have to increase functionality within the processor to incorporate various signal processing (multimedia and graphics) and memory functions now assigned to separate chips.

Area optimization simply means making the best use of available silicon. Each algorithmic state of the art can be encapsulated in an  $AT^n$  (area-time) curve. In general, we have  $AT^n = k$ , where  $n$  usually ranges from 1 to 2, depending on the nature of communication within the implementation.

Note that  $k$  defines the state-of-the-art, or

“par,” designs at any particular moment. If some implementations use design points in the interior, the resultant design is inferior, or “over par.” Over time,  $k$  (and par) decreases because of advances in technology and in algorithms themselves.

To study algorithms, we can largely eliminate the effect of technology by using

a technology-normalized metric. Along this line, Fu developed a cost-performance metric for evaluating floating-point-unit implementations.<sup>19</sup> The metric, floating-point-unit cost-performance analysis (FUPA), incorporates five key aspects of VLSI systems design: latency, die area, power, minimum feature size, and a profile of applications. FUPA uses technology projections based on scalable device models to identify the design/technology compatibility and lets designers make high-level trade-offs in optimizing FPU designs. Table 2 shows example FUPA applications to specific processors. Processor FPUs with the lowest FUPA define the state of the art.

### Computational integrity

The last basic trade-off is determining the level of computational integrity. When rebooting a personal computer after an application has caused the system to crash, we may wonder about the application or the system or both. However, the observed failure is a retrograde problem solved years ago in hardware with the introduction of user and system states and corresponding memory protection. What's lacking is the determination to implement the solution. In looking ahead to improved models of computational integrity, we should consider

- reliability,
- testability,
- serviceability,
- process recoverability, and
- fail-safe computation.

Reliability is a characteristic of the implementation media. Circuits and cells may fail, but this need not lead immediately to demonstrable faults in the processor. Indeed, small-

er feature sizes may lead to increasing failures over time resulting from electrostatic overstress, electro-migration, and so on. Error correction systems provide an important way to recover from certain modes of device failure. In case of transient errors, error detection systems coupled with instruction retry are a minimum requirement for enabling correct computations.

Testable designs explicitly include accessibility paths, such as scan paths, that enable special validation programs to verify a processor's correct operation over a broad variety of state combinations. Testability is important for continuing test and design validation.

Serviceability allows for ready diagnosis of both transient and permanent failures. It depends on error detection, error scanning on detection, and error logging. The goal is a design that lets us identify degraded paths caused by recoverable but recurring errors.

Process recoverability includes features for instruction retry, process rollback, and, in multiprocessor systems, process migration to another processor.

Fail-safe computation integrates all the above with environmental considerations such as power and temperature. In principle, even power failure should not cause an executing process to abort. Using an uninterruptible power supply or some other backup system lets us save the system state so that computation can resume when power returns.

## Designing the "new" processor

In the deep-submicron era, greater capacity (computational as well as memory) and faster transistors encourage exploration of new architectural domains and execution paradigms. However, wire delays and less dramatic increases in chip bandwidth restrict the range of implementable solutions because of the need to decouple regions on the die and to limit demand for off-chip resources.

Aggressive circuit technologies let architects and designers effectively exploit the benefits of new technology, but alone they do not suffice. Although we focused here on hardware issues, not all solutions will be possible relying solely on hardware. We will also need cooperating software solutions, such as self-adapting applications and algorithms, as well as improvements at the language, compiler, and

## CAD tools

Circuit complexity and constraints will characterize future processor design, making effective design tools essential. On the circuit level, techniques such as self-timed and wave-pipelined circuits become attractive in deep-submicron designs. We see a dire need to develop more efficient and reliable CAD tools to help in designing these difficult circuits. As the number of transistors on a chip increases, the amount of time a designer can spare for each transistor decreases. Designers must rely on CAD tools to ensure that timing and other design requirements are met.

On the logic level, interconnect capacitance will largely determine chip performance and power consumption. Traditional logic optimization tools, which optimize the total gate delay, will not suffice. Synthesis tools must optimize logic design and physical placement at the same time. Similarly, on the system level, design tools must calculate interconnect delay and congestion in very early design stages. Interconnect-driven placement and floor-planning tools are needed to improve routability and to optimize performance and power dissipation.

As system complexity increases and design time decreases, reuse of processor core intellectual property (IP)<sup>1</sup> seems the only way to produce a reliable design within a reasonable time. CAD tools must automatically map the IP implementations to the required specifications and generate development tools— assembler, compiler, simulator, and debugger. Optimization requires high-level cycle-accurate simulators and accurate area estimation tools. In addition to performance modeling and analysis, such a cycle-accurate simulator can help with hardware/software codesign and co-verification.<sup>2,3</sup>

## References

1. Virtual Socket Interface Architecture Document, Virtual Socket Interface Alliance, <http://www.vsi.org/library/vsi-or.pdf>.
2. G. De Micheli and M. Sami, *Hardware/Software Co-Design*, PhD thesis, Kluwer Academic Publishers, Norwell, Mass., 1996.
3. Mentor Graphics Seamless, <http://www.mentorgraphics.com/codesign/main-f/>.

operating system level. To achieve these goals, we need better CAD tools (see the sidebar).

Server processor applications, where chip and system costs are less important than total performance, encompass a wide range of potential requirements, from computation intensive to memory intensive to I/O intensive. Designers may increase computation by constructing ever more capable processor units (super-SISD) or by integrating additional processor units on a given die (super-SIMD, super-MIMD, or super-MISD). (SISD: single-instruction, single-data; SIMD: single-instruction, multiple-data; MIMD: multiple-instruction, multiple-data; MISD: multiple-instruction, single-data.) They may increase memory capacity and performance by placing more of the memory hierarchy on chip. They may increase I/O bandwidth by using improved packaging techniques to increase the number of available pins or by

using direct optical connections to the die. Meeting server performance requirements demands both traditional and nontraditional solutions.

In this cost-is-no-object server region, the need to customize implementations to specific applications may alter manufacturing as well. Although expensive, effective customization may require fabrication microproduction runs to maximize performance. One side effect of such a manufacturing change is the requirement to automate software tool delivery for each implementation. Tensilica took this approach, which delivers not only a custom processor implementation but also customized tools (compiler, assembler, and debugger) that provide basic access to the specific capabilities of a given implementation.

For many server applications, hundreds of processors on a die can be very attractive, possibly even providing sufficient area to include gigabytes of memory. Decoupling requirements (due to long wires, as discussed earlier) limit the overall integration potential, leaving significant area available for redundancy. This redundancy can serve to improve both die yield and computational integrity. In addition, this die area surplus may let designers implement higher performance communication structures; optical communications might help reduce the liability of wire delay across the chip.

Architecturally at least, the challenges for client processors are somewhat simpler than for server processors. The client design point is a processor operating at low power and reduced speed for some applications, perhaps on the order of 100  $\mu$ W and 100 MHz, respectively, with years, not hours, of battery life. A client processor is limited by long wires and hence is partitioned into multiple units: core processor with cache, various signal processors, wireless (RF) capabilities, and cryptographic arithmetic facilities. The resultant system is also bound by computational integrity requirements, but probably not at the same high level as a server. We see the core processor as a generally conventional design, but the supporting signal processors may use vector/VLIW or other suitable forms of ILP to manage their tasks efficiently. Memory may also occupy the same die, as long as we can predetermine the size requirements and allow sufficient space.

As technology scales, important new opportunities emerge for microprocessor architects. The simple, traditional measures of processor performance—cycle time and cache size—become less meaningful in correctly evaluating application performance. The most significant challenges facing the microprocessor architect include

- Creating high-performance server processors jointly or cooperatively with enabling compiler software. Whether the resultant architectures are vector processors, VLIW, or some other type, the processors must actually deliver the specified performance across a spectrum of applications.
- Using advanced CAD tools to design power-sensitive system-on-chip client processors in a very short design time. System issues such as testing and verification become important challenges.
- Improving ways to preserve the integrity of computation, reliability, and diagnostic features.
- Increasing the use of adaptability in various processor structures, such as cache and signal processors. For example, an adaptive cache would not just prefetch, it would prefetch according to a particular program's history of accessing behavior. Similarly, we may see adaptability in arithmetic, where arithmetic functional units can be redefined, perhaps with the assistance of programmable logic elements in functional units, to improve performance for a range of applications with different computational needs.

Understanding technology trends and specific applications is the main criterion for designing efficient and effective processors. Without such understanding, design complexity will quickly become overwhelming, preventing designers from using a die's full capacity.

MICRO

## References

1. *The National Technology Roadmap for Semiconductors*, tech. report, Semiconductor Industry Assn., San Jose, Calif., 1994 and 1997 (updated).
2. H.B. Bakoglu, *Circuit, Interconnections, and*

- Packaging for VLSI*, Addison-Wesley, Reading, Mass., 1990.
3. J.D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Rockville, Md., 1984.
  4. S.H. Unger, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, Somerset, N.J., 1969.
  5. D. Harris, *Skew-Tolerant Circuit Design*, PhD thesis, Stanford Univ., Stanford, Calif., 1999.
  6. L. Cotton, "Maximum Rate Pipelined Systems," *Proc. AFIPS Spring Joint Computer Conf.*, 1969, pp. 581-586.
  7. K. Nowka, *High Performance CMOS System Design Using Wave Pipelining*, PhD thesis, Stanford Univ., Dept. Electrical Eng., 1995.
  8. F. Klass, M. Flynn, and A.J. van de Goor, "Fast Multiplication in VLSI Using Wave Pipelining," *J. VLSI Signal Processing*, Vol. 7, No. 3, May 1994, pp. 233-248.
  9. K.W. Rudd, *VLIW Processors: Efficiently Exploiting Instruction-level Parallelism*, PhD thesis, Stanford Univ., Dept. Electrical Eng., 1999.
  10. P.K. Dubey and M.J. Flynn, "Optimal Pipelining," *J. Parallel and Distributed Computing*, Vol. 8, No. 1, Jan. 1990, pp. 10-19.
  11. P. Hung and M.J. Flynn, *Optimum ILP for Superscalar and VLIW Processors*, Tech. Report CSL-TR-99-737, Stanford Univ., Dept. Electrical Eng., 1999.
  12. J.E. Bennett, *Latency Tolerant Architectures*, PhD thesis, Stanford Univ., Computer Science Dept., 1998.
  13. W. Wulf and S. McKee, "Hitting the Memory Wall: Implications of the Obvious," *ACM Computer Architecture News*, Vol. 13, No. 1, Mar. 1995, pp. 20-24.
  14. A.P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low-Power CMOS Digital Design," *IEEE J. Solid State Circuits*, Vol. 27, No. 4, Apr. 1992, pp. 473-484.
  15. K. Chen et al., "Predicting CMOS Speed with Gate Oxide and Voltage Scaling and Interconnect Loading Effects," *IEEE Trans. Electron Devices*, Vol. 44, No. 11, Nov. 1997, pp. 1,951-1,957.
  16. M. Godfrey, "CMOS Device Modeling for Subthreshold Circuits," *IEEE Trans. Circuits and Systems*, Vol. 39, No. 2, Aug. 1992, pp. 532-539.
  17. M.J. Flynn, *Computer Architecture Pipelined and Parallel Processor Design*, Jones and Bartlett Publishers, Boston, Mass., 1995.
  18. G.W. McFarland, *CMOS Technology Scaling and Its Impact on Cache Delay*, PhD thesis, Stanford Univ., Dept. Electrical Eng., 1997.
  19. S. Fu, *Cost Performance Optimization of Microprocessor*, PhD thesis, Stanford Univ., Dept. Electrical Eng., 1999.

**Michael J. Flynn** is professor of electrical engineering at Stanford University. He was founding chair of both the ACM Special Interest Group on Computer Architecture and the IEEE Computer Society's Technical Committee on Computer Architecture. He received his PhD from Purdue University. He was the 1992 recipient of the ACM/IEEE Eckert-Mauchley Award and the 1995 recipient of the IEEE Computer Society's Harry Goode Memorial Award.

**Patrick Hung** is a PhD candidate in the Stanford Architecture and Arithmetic Group at Stanford University. His research interests include computer arithmetic, microprocessor architecture, and deep-submicron CAD tool design. He received an MSEE from Stanford University and a BSEE from the University of Hong Kong. He is a student member of the IEEE.

**Kevin W. Rudd** is a PhD candidate in the Stanford Architecture and Arithmetic Group. His research interests include high-performance computer architecture, hardware-software design trade-offs, and architectural simulation and modeling. He received a BSEE and an MSEE from Stanford University and anticipates receiving a PhD in 1999.

Direct comments about this article to Patrick Hung, Gates Computer Science Building 3A, Room 332, 353 Serra Mall, Stanford, CA 94305-9030; hung@arith.stanford.edu.