Coming Challenges in Microarchitecture and Architecture

RONNY RONEN, SENIOR MEMBER, IEEE, AVI MENDELSON, MEMBER, IEEE, KONRAD LAI, SHIH-LIEN LU, MEMBER, IEEE, FRED POLLACK, AND JOHN P. SHEN, FELLOW, IEEE

Invited Paper

In the past several decades, the world of computers and especially that of microprocessors has witnessed phenomenal advances. Computers have exhibited ever-increasing performance and decreasing costs, making them more affordable and, in turn, accelerating additional software and hardware development that fueled this process even more. The technology that enabled this exponential growth is a combination of advancements in process technology, microarchitecture, architecture, and design and development tools. While the pace of this progress has been quite impressive over the last two decades, it has become harder and harder to keep up this pace. New process technology requires more expensive megafabs and new performance levels require larger die, higher power consumption, and enormous design and validation effort. Furthermore, as CMOS technology continues to advance, microprocessor design is exposed to a new set of challenges. In the near future, microarchitecture has to consider and explicitly manage the limits of semiconductor technology, such as wire delays, power dissipation, and soft errors. In this paper, we describe the role of microarchitecture in the computer world, present the challenges ahead of us, and highlight areas where microarchitecture can help address these challenges.

Keywords*—*Design tradeoffs, microarchitecture, microarchitecture trends, microprocessor, performance improvements, power issues, technology scaling.

I. INTRODUCTION

Microprocessors have gone through significant changes during the last three decades; however, the basic computational model has not changed much. A program consists of instructions and data. The instructions are encoded in a specific instruction set architecture (ISA). The computational model is still a single instruction stream, sequential execution model, operating on the architecture states (memory and registers). It is the job of the microarchitecture, the logic, and the circuits to carry out this instruction stream in the "best" way. "Best" depends on intended usage—servers, desktop, and mobile—usually categorized as market segments. For example, servers are designed to achieve the highest performance possible while mobile systems are optimized for best performance for a given power. Each market segment has different features and constraints.

A. Fundamental Attributes

The key metrics for characterizing a microprocessor include: performance, power, cost (die area), and complexity.

Performance is measured in terms of the time it takes to complete a given task. Performance depends on many parameters such as the microprocessor itself, the specific workload, system configuration, compiler optimizations, operating systems, and more. A concise characterization of microprocessor performance was formulated by a number of researchers in the 1980s; it has come to be known as the "iron law" of central processing unit performance and is shown below

$$\begin{aligned}\text{Performance} &= 1/\text{Execution Time}\\ &= (\text{IPC} \times \text{Frequency})/\text{Instruction-Count}\end{aligned}$$

where IPC is the average number of instructions completed per cycle, Frequency is the number of clock cycles per second, and Instruction-Count is the total number of instructions executed. Performance can be improved by increasing IPC and/or frequency or by decreasing instruction count. In practice, IPC varies depending on the environment—the application, the system configuration, and more. Instruction count depends on the ISA and the compiler used. For a given executable program, where the instruction

Manuscript received January 1, 2000; revised October 1, 2000.

R. Ronen and A. Mendelson are with the Microprocessor Research Laboratories, Intel Corporation, Haifa 31015, Israel.

K. Lai and S.-L. Lu are with the Microprocessor Research Laboratories, Intel Corporation, Hillsboro, OR 97124 USA.

F. Pollack and J. P. Shen are with the Microprocessor Research Laboratories, Intel Corporation, Santa Clara, CA 95052 USA

Publisher Item Identifier S 0018-9219(01)02069-2.

stream is invariant, the relative performance depends only on IPC $\times$ Frequency. Performance here is measured in million instructions per second (MIPS).

Commonly used benchmark suites have been defined to quantify performance. Different benchmarks target different market segments, such as SPEC [1] and SysMark [2]. A benchmark suite consists of several applications. The time it takes to complete this suite on a certain system reflects the system performance.

Power is energy consumption per unit time, in watts. Higher performance requires more power. However, power is constrained due to the following.

- *Power density and Thermal*: The power dissipated by the chip per unit area is measured in watts/cm^2. Increases in power density causes heat to generate. In order to keep transistors within their operating temperature range, the heat generated has to be dissipated from the source in a cost-effective manner. Power density may soon limit performance growth due to thermal dissipation constraints.
- *Power Delivery:* Power must be delivered to a very large scale integration (VLSI) component at a prescribed voltage and with sufficient amperage for the component to run. Very precise voltage regulator/transformer controls current supplies that can vary within nanoseconds. As the current increases, the cost and complexity of these voltage regulators/transformers increase as well.
- *Battery Life:* Batteries are designed to support a certain watts $\times$ hours. The higher the power, the shorter the time that a battery can operate.

Until recently, power efficiency was a concern only in battery powered systems like notebooks and cell phones. Recently, increased microprocessor complexity and frequency have caused power consumption to grow to the level where power has become a first-order issue. Today, each market segment has its own power requirements and limits, making power limitation a factor in any new microarchitecture. Maximum power consumption is increased with the microprocessor operating voltage (V) and frequency (*Frequency*) as follows:

$$\text{Power} = C \times V^2 \times \text{Frequency}$$

where C is the effective load capacitance of all devices and wires on the microprocessor. Within some voltage range, frequency may go up with supply voltage V ($\text{Frequency} = k \times V^{\alpha-1}$, $\alpha \geq 1$). This is a good way to gain performance, but power is also increased (proportional to $V^{2+\alpha}$). Another important power related metric is the *energy efficiency*. Energy efficiency is reflected by the performance/power ratio and measured in MIPS/watt.

Cost is primarily determined by the physical size of the manufactured silicon die. Larger area means higher (even more than linear) manufacturing cost. Bigger die area usually implies higher power consumption and may potentially imply lower frequency due to longer wires. Manufacturing yield also has direct impact on the cost of each microprocessor.

Complexity reflects the effort required to design, validate, and manufacture a microprocessor. Complexity is affected by the number of devices on the silicon die and the level of aggressiveness in the performance, power and die area targets. Complexity is discussed only implicitly in this paper.

B. Enabling Technologies

The microprocessor revolution owes it phenomenal growth to a combination of enabling technologies: process technology, circuit and logic techniques, microarchitecture, architecture (ISA), and compilers.

Process technology is the fuel that has moved the entire VLSI industry and the key to its growth. A new process generation is released every two to three years. A process generation is usually identified by the length of a metal-oxide-semidconductor gate, measured in micrometers (10^{-6} m, denoted as μm). The most advanced process technology today (year 2000) is 0.18 μm [3].

Every new process generation brings significant improvements in all relevant vectors. Ideally, process technology scales by a factor of $\sim$0.7 all physical dimensions of devices (transistors) and wires (interconnects) including those vertical to the surface and all voltages pertaining to the devices [4]. With such scaling, typical improvement figures are the following:

- 1.4–1.5 times faster transistors;
- two times smaller transistors;
- 1.35 times lower operating voltage;
- three times lower switching power.

Theoretically, with the above figures, one would expect potential improvements such as the following.

- *Ideal Shrink:* Use the same number of transistors to gain 1.5 times performance, two times smaller die, and two times less power.
- *Ideal New Generation:* Use two times the number of transistors to gain three times performance with no increase in die size and power.

In both ideal scenarios, there is three times gain in MIPS/watt and no change in power density (watts/cm^2).

In practice, it takes more than just process technology to achieve such performance improvements and usually at much higher costs. However, process technology is the single most important technology that drives the microprocessor industry. Growing 1000 times in frequency (from 1 MHz to 1 GHz) and integration (from $\sim$10k to $\sim$10M devices) in 25 years was not possible without process technology improvements.

Innovative **circuit** implementations can provide better performance or lower power. New **logic** families provide new methods to realize logic functions more effectively.

Microarchitecture attempts to increase both IPC and frequency. A simple frequency boost applied to an existing microarchitecture can potentially reduce IPC and thus does not achieve the expected performance increase. For

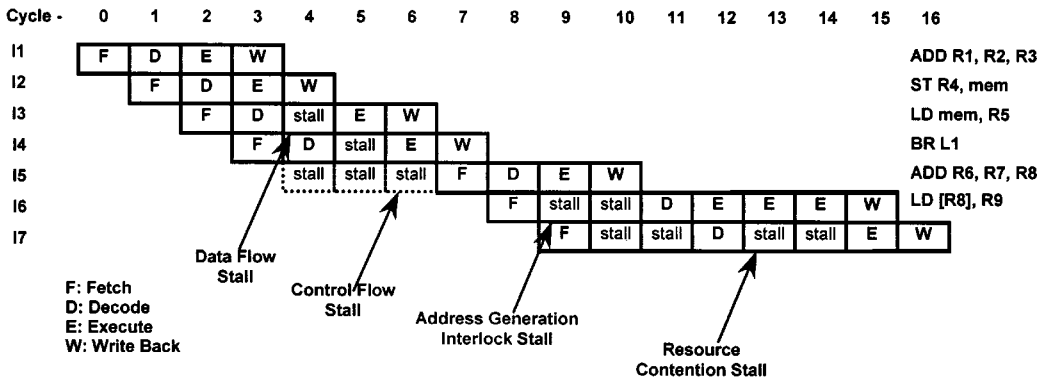

Fig. 1. Impact of different pipeline stalls on the execution flow.

example, memory accesses latency does not scale with microprocessor frequency. Microarchitecture techniques such as caches, branch prediction, and out-of-order execution can increase IPC. Other microarchitecture ideas, most notably pipelining, help to increase frequency beyond the increase provided by process technology.

Modern **architecture** (ISA) and good optimizing **compilers** can reduce the number of dynamic instructions executed for a given program. Furthermore, given knowledge of the underlying microarchitecture, compilers can produce optimized code that lead to higher IPC.

This paper deals with the challenges facing architecture and microarchitecture aspects of microprocessor design. A brief tutorial/background on traditional microarchitecture is given in Section II, focusing on frequency and IPC tradeoffs. Section III describes the past and current trends in microarchitecture and explains the limits of the current approaches and the new challenges. Section IV suggests potential microarchitectural solutions to these challenges.

II. Microarchitecture at a Glance

Microprocessor performance depends on its frequency and IPC. Higher frequency is achieved with process, circuit, and microarchitectural improvements. New process technology reduces gate delay time, thus cycle time, by $\sim$1.5 times. Microarchitecture affects frequency by reducing the amount of work done in each clock cycle, thus allowing shortening of the clock cycle.

Microarchitects tend to divide the microprocessor's functionality into three major components [5].

- *Instruction Supply:* Fetching instructions, decoding them, and preparing them for execution;
- *Execution:* Choosing instructions for execution, performing actual computation, and writing results;
- *Data Supply:* Fetching data from the memory hierarchy into the execution core.

A rudimentary microprocessor would process a complete instruction before starting a new one. Modern microprocessors use pipelining. Pipelining breaks the processing of an instruction into a sequence of operations, called stages. For example, in Fig. 1, a basic four-stage pipeline breaks the instruction processing into fetch, decode, execute, and write-back stages. A new instruction enters a stage as soon as the previous one completes that stage. A pipelined microprocessor with N pipeline stages can overlap the processing of N instructions in the pipeline and, ideally, can deliver N times the performance of a nonpipelined one.

Pipelining is a very effective technique. There is a clear trend of increasing the number of pipe stages and reducing the amount of work per stage. Some microprocessors (e.g., Pentium Pro microprocessor [6]) have more than ten pipeline stages. Employing many pipe stages is sometimes termed deep pipelining or super pipelining.

Unfortunately, the number of pipeline stages cannot increase indefinitely.

- There is a certain clocking overhead associated with each pipe stage (setup and hold time, clock skew). As cycle time becomes shorter, further increase in pipeline length can actually decrease performance [7].
- Dependencies among instructions can require stalling certain pipe stages and result in wasted cycles, causing performance to scale less than linearly with the number of pipe stages.

For a given partition of pipeline stages, the frequency of the microprocessor is dictated by the latency of the slowest pipe stage. More expensive logic and circuit optimizations help to accelerate the speed of the logic within the slower pipe stage, thus reducing the cycle time and increasing frequency without increasing the number of pipe stages.

It is not always possible to achieve linear performance increase with deeper pipelines. First, scaling frequency linearly with the number of stages requires good balancing of the overall work among the stages, which is difficult to achieve. Second, with deeper pipes, the number of wasted cycles, termed *pipe stalls*, grows. The main reasons for stalls are resource contention, data dependencies, memory delays, and control dependencies.

- **Resource contention** causes pipeline stall when an instruction needs a resource (e.g., execution unit) that is currently being used by another instruction in the same cycle.
- **Data dependency** occurs when the result of one instruction is needed as a source operand by another instruction. The dependent instruction has to wait (stall) until all its sources are available.

Table 1
Out-Of-Order Execution Example

Cycle	Scalar / In order	Super-Scalar / In order	Super-Scalar / Out-of-Order
1	Load eax, mem1	Load eax, mem1	Load eax_1, mem1 / Load eax_2, mem3
2	Store mem2, eax	Store mem2, eax / Load eax, mem3	Store mem2, eax_1 / Store mem4. Eax_2
3	Load eax, mem3	Store mem4. Eax	
4	Store mem4. Eax		

- **Memory delays** are caused by memory related data dependencies, sometimes termed *load-to-use delays*. Accessing memory can take between a few cycles to hundreds of cycles, possibly requiring stalling the pipe until the data arrives.
- **Control dependency** stalls occur when the control flow of the program changes. A branch instruction changes the address from which the next instruction is fetched. The pipe may stall and instructions are not fetched until the new fetch address is known.

Fig. 1 shows the impact of different pipeline stalls on the execution flow within the pipeline.

In a 1-GHz microprocessor, accessing main memory can take about 100 cycles. Such accesses may stall a pipelined microprocessor for many cycles and seriously impact the overall performance. To reduce memory stalls at a reasonable cost, modern microprocessors take advantage of the locality of references in the program and use a hierarchy of memory components. A small, fast, and expensive (in $/bit) memory called a *cache* is located on-die and holds frequently used data. A somewhat bigger, but slower and cheaper cache may be located between the microprocessor and the system bus, which connects the microprocessor to the main memory. The main memory is yet slower, but bigger and inexpensive.

Initially, caches were small and off-die; but over time, they became bigger and were integrated on chip with the microprocessor. Most advanced microprocessors today employ two levels of caches on chip. The first level is $\sim$32–128 kB—it typically takes two to three cycles to access and typically catches about 95% of all accesses. The second level is 256 kB to over 1 MB—it typically takes six to ten cycles to access and catches over 50% of the misses of the first level. As mentioned, off-chip memory accesses may elapse about 100 cycles.

Note that a cache miss that eventually has to go to the main memory can take about the same amount of time as executing 100 arithmetic and logic unit (ALU) instructions, so the structure of memory hierarchy has a major impact on performance. Much work has been done in improving cache performance. Caches are made bigger and heuristics are used to make sure the cache contains those portions of memory that are most likely to be used [8], [9].

Change in the control flow can cause a stall. The length of the stall is proportional to the length of the pipe. In a super-pipelined machine, this stall can be quite long. Modern microprocessors partially eliminate these stalls by employing a technique called *branch prediction*. When a branch is fetched, the microprocessor speculates the direction (taken/not taken) and the target address where a branch will go and starts speculatively executing from the predicted address. Branch prediction uses both static and runtime information to make its predictions. Branch predictors today are very sophisticated. They use an assortment of per-branch (local) and all-branches (global) history information and can correctly predict over 95% of all conditional branches [10], [11]. The prediction is verified when the predicted branch reaches the execution stage and if found wrong, the pipe is flushed and instructions are fetched from the correct target, resulting in some performance loss. Note that when a wrong prediction is made, useless work is done on processing instructions from the wrong path.

The next step in performance enhancement beyond pipelining calls for executing several instructions in parallel. Instead of "scalar" execution, where in each cycle only one instruction can be resident in each pipe stage, *superscalar* execution is used, where two or more instructions can be at the same pipe stage in the same cycle. Superscalar designs require significant replication of resources in order to support the fetching, decoding, execution, and writing back of multiple instructions in every cycle. Theoretically, an N-way superscalar pipelined microprocessor can improve performance by a factor of N over a standard scalar pipelined microprocessor. In practice, the speedup is much smaller. Interinstruction dependencies and resource contentions can stall the superscalar pipeline.

The microprocessors described so far execute instructions *in-order*. That is, instructions are executed in the program order. In an in-order processing, if an instruction cannot continue, the entire machine stalls. For example, a cache miss delays all following instructions even if they do not need the results of the stalled load instruction. A major breakthrough in boosting IPC is the introduction of *out-of-order* execution, where instruction execution order depends on data flow, not on the program order. That is, an instruction can execute if its operands are available, even if previous instructions are still waiting. Note that instructions are still fetched in order. The effect of superscalar and out-of-order processing is shown in an example in Table 1 where two memory words mem1 and mem3 are copied into two other memory locations mem2 and mem4.

Out-of-order processing hides some stalls. For example, while waiting for a cache miss, the microprocessor can execute newer instructions as long as they are independent of the load instructions. A superscalar out-of-order microprocessor can achieve higher IPC than a superscalar in-order microprocessor. Out-of-order execution involves dependency analysis and instruction scheduling. Therefore, it takes a longer time (more pipe stages) to process an

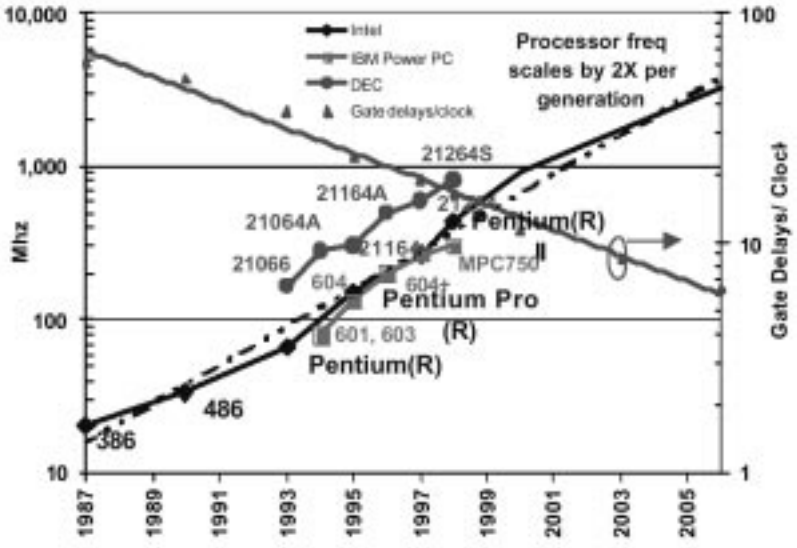

Fig. 2. Processor frequencies over years. (Source: V. De, Intel, ISLPED, Aug. 1999.)

instruction in an out-of- order microprocessor. With a deeper pipe, an out-of-order microprocessor suffers more from branch mispredictions. Needless to say, an out-of-order microprocessor, especially a wide-issue one, is much more complex and power hungry than an in-order microprocessor [12].

Historically, there were two schools of thought on how to achieve higher performance. The "Speed Demons" school focused on increasing frequency. The "Brainiacs" focused on increasing IPC [13], [14]. Historically, DEC Alpha [15] was an example of the superiority of "Speed Demons" over the "Brainiacs." Over the years, it has become clear that high performance must be achieved by progressing in both vectors (see Fig. 4).

To complete the picture, we revisit the issues of performance and power. A microprocessor consumes a certain amount of energy, W_i, in processing an instruction. This amount increases with the complexity of the microprocessor. For example, an out-of-order microprocessor consumes more energy per instruction than an in-order microprocessor.

When speculation is employed, some processed instructions are later discarded. The ratio of useful to total number of processed instructions is η. The total IPC including speculated instructions is therefore IPC/η. Given these observations a number of conclusions can be drawn. The energy per second, hence power, is proportional to the amount of processed instructions per second and the amount of energy consumed per instruction, that is (IPC/η)*Frequency*W_i. The energy efficiency, measured in MIPS/watt, is proportional to η/W_i. This value deteriorates as speculation increases and complexity grows.

One main goal of microarchitecture research is to design a microprocessor that can accomplish a group of tasks (applications) in the shortest amount of time while using minimum amount of power and incurring the least amount of cost. The design process involves evaluating many parameters and balancing these three targets optimally with given process and circuit technology.

III. Microprocessors—Current Trends and Challenges

In the past 25 years, chip density and the associated computer industry has grown at an exponential rate. This phenomenon is known as "Moore's Law" and characterizes almost every aspect of this industry, such as transistor density, die area, microprocessor frequency, and power consumption. This trend was possible due to the improvements in fabrication process technology and microprocessor microarchitecture. This section focuses on the architectural and the microarchitectural improvements over the years and elaborates on some of the current challenges the microprocessor industry is facing.

A. Improving Performance

As stated earlier, performance can be improved by increasing IPC and/or frequency or by decreasing the instruction count. Several architecture directions have been taken to improve performance. Reduced instruction set computer (RISC) architecture seeks to increase both frequency and IPC via pipelining and use of cache memories at the expense of increased instruction count. Complex instruction set computer (CISC) microprocessors employ RISC-like internal representation to achieve higher frequency while maintaining lower instruction count. Recently, the very long instruction word (VLIW) [16] concept was revived with the Explicitly Parallel Instruction Computing (EPIC) [17]. EPIC uses the compiler to schedule instruction statically. Exploiting parallelism statically can enable simpler control logic and help EPIC to achieve higher IPC and higher frequency.

1) Improving Frequency via Pipelining: Process technology and microarchitecture innovations enable doubling the frequency increase every process generation. Fig. 2 presents the contribution of both: as the process improves, the frequency increases and the average amount of work done in pipeline stages decreases. For example, the number of gate delays per pipe stage was reduced by about three

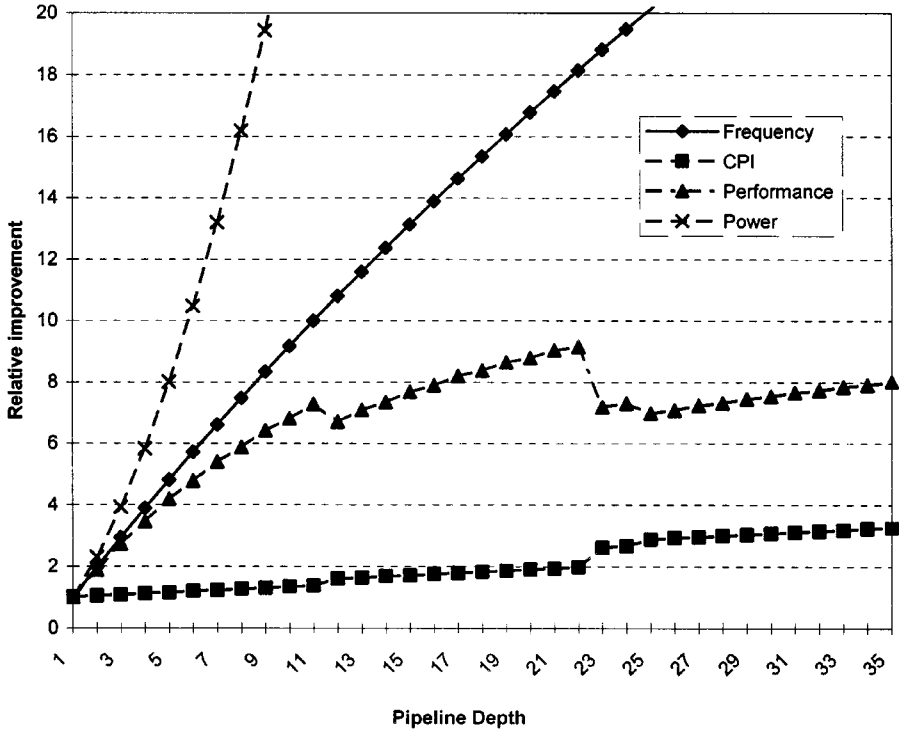

Fig. 3. Frequency and performance improvements—synthetic model. (Source: E. Grochowski, Intel, 1997.)

times over a period of ten years. Reducing the stage length is achieved by improving design techniques and increasing the number of stages in the pipe. While in-order microprocessors used four to five pipe stages, modern out-of-order microprocessors can use over ten pipe stages. With frequencies higher than 1 GHz, we can expect over 20 pipeline stages.

Improvement in frequency does not always improve performance. Fig. 3 measures the impact of increasing the number of pipeline stages on performance using a synthetic model of an in-order superscalar machine. Performance scales less than frequency (e.g., going from 6 to 12 stages yields only a 1.75 times speedup, from 6 to 23 yields only 2.2 times). Performance improves less than linearly due to cache misses and branch mispredictions. There are two interesting singular points in the graph that deserve special attention. The first (at pipeline depth of 13 stages) reflects the point where the cycle time becomes so short that two cycles are needed to reach the first level cache. The second (at pipeline depth of 24 stages) reflects the point where the cycle time becomes extremely short so that two cycles are needed to complete even a simple ALU operation. Increasing the latency of basic operations introduces more pipeline stalls and impacts performance significantly. Please note that these trends are true for any pipeline design though the specific data points may vary depending on the architecture and the process. In order to keep the pace of performance growth, one of the main challenges is to increase the frequency without negatively impacting the IPC. The next sections discuss some IPC related issues.

2) Instruction Supply Challenges: The instruction supply is responsible for feeding the pipeline with useful instructions. The rate of instructions entering the pipeline depends on the fetch bandwidth and the fraction of useful instructions in that stream. The fetch rate depends on the effectiveness of the memory subsystem and is discussed later along with data supply issues. The number of useful instructions in the instruction stream depends on the ISA and the handling of branches. Useless instructions result from: 1) control flow change within a block of fetched instructions, leaving the rest of the cache block unused; and 2) branch misprediction brings instructions from the wrong path that are later discarded. On average, a branch occurs every four to five instructions. Hence, appropriate fetch bandwidth and accurate branch prediction are crucial.

Once instructions are fetched into the machine they are decoded. RISC architectures, using fixed length instructions, can easily decode instructions in parallel. Parallel decoding is a major challenge for CISC architectures, such as IA32, that use variable length instructions. Some implementations [18] use speculative decoders to decode from several potential instruction addresses and later discard the wrong ones; others [19] store additional information in the instruction cache to ease decoding. Some IA32 implementations (e.g., the Pentium II microprocessor) translate the IA32 instructions into an internal representation (micro-operations), allowing the internal part of the microprocessor to work on simple instructions at high frequency, similar to RISC microprocessors.

3) Efficient Execution: The front-end stages of the pipeline prepare the instructions in either an instruction

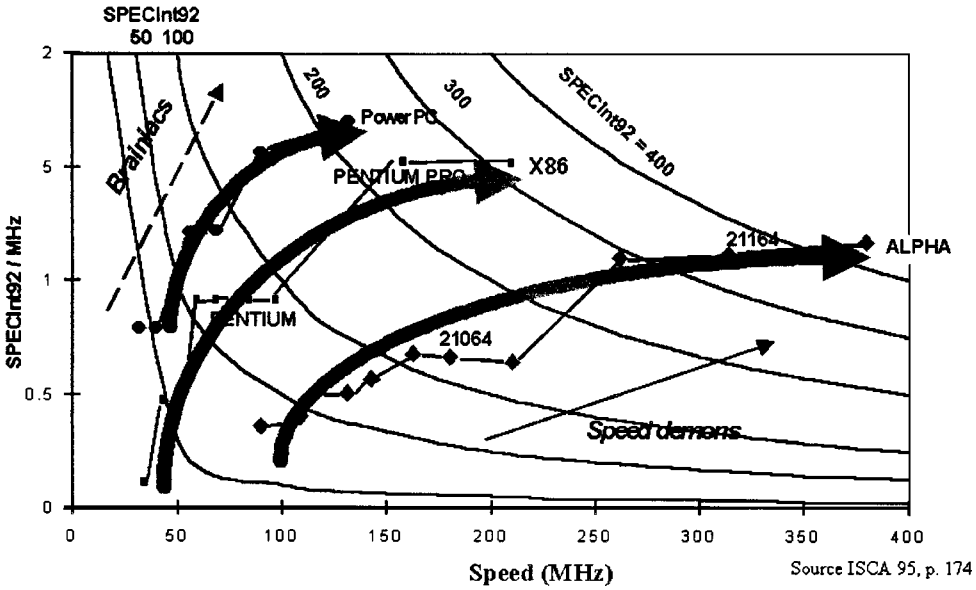

Fig. 4. Landscape of microprocessor families.

window [20] or reservation stations [21]. The execution core schedules and executes these instructions. Modern microprocessors use multiple execution units to increase parallelism. Performance gain is limited by the amount of parallelism found in the instruction window. The parallelism in today's machines is limited by the data dependencies in the program and by memory delays and resource contention stalls.

Studies show that in theory, high levels of parallelism are achievable [22]. In practice, however, this parallelism is not realized, even when the number of execution units is abundant. More parallelism requires higher fetch bandwidth, a larger instruction window, and a wider dependency tracker and instruction scheduler. Enlarging such structures involves polynomial complexity increase for less than a linear performance gain (e.g., scheduling complexity is on the order of O^3 of the scheduling window size [23]). VLIW architectures [16] such as IA64 EPIC [17] avoid some of this complexity by using the compiler to schedule instructions.

Accurate branch prediction is critical for deep pipelines in reducing misprediction penalty. Branch predictors have become larger and more sophisticated. The Pentium microprocessor [18] uses 256 entries of 2-bit predictors (the predictor and the target arrays consume ~15 kB) that achieve ~85% correct prediction rate. The Pentium III microprocessor [24] uses 512 entries of two-level local branch predictor (consuming ~30 kB) and yields ~90% prediction rate. The Alpha 21 264 [25] uses a hybrid multilevel selector predictor with 5120 entries (consuming ~80 kB) and achieves ~94% accuracy.

As pipelines become deeper and fetch bandwidth becomes wider, microprocessors will have to predict multiple branches in each cycle and use bigger multilevel branch prediction structures similar to caches.

B. Accelerating Data Supply

All modern microprocessors employ memory hierarchy. The growing gap between the frequency of the microprocessor that doubles every two to three years and the main memory access time that only increases ~7% per year impose a major challenge. The latency of today's main memory is ~100 ns, which approximately equals 100 microprocessor cycles. The efficiency of the memory hierarchy is highly dependent on the software and varies widely for different applications.

The size of cache memories increases according to Moore's Law. The main reason for bigger caches is to support a bigger working set. New applications such as multimedia and communication applications use larger data structures, hence bigger working sets, than traditional applications. Also, the use of multiprocessing and multithreading in modern operating systems such as WindowsNT and Linux causes frequent switches among applications. This results in further growth of the active working set.

Increasing the cache memory size increases its access time. Fast microprocessors, such as the Alpha or the Pentium III microprocessors, integrate two levels of caches on the microprocessor die to get improved average access time to the memory hierarchy. Embedded microprocessors integrate bigger, but slower dynamic random access memory (DRAM) on the die. DRAM on die involves higher latency, manufacturing difficulty, and software complexity and is, therefore, not attractive for use in current generation general-purpose microprocessors. Prefetching is a different technique to reduce access time to memory. Prefetching anticipates the data or instructions the program will access in the near future and brings them to the cache ahead of time. Prefetching can be implemented as a hardware mechanism or can be instrumented with software. Many microprocessors use a simple hardware prefetching [26] mechanism to bring ahead "one instruction cache line" into the cache. This mechanism is very efficient for manipulating instruction streams, but less effective in manipulating data due to cache pollution. A different approach uses ISA extensions; e.g., the Pentium III microprocessor *prefetch*

instruction hints to the hardware, to prefetch a cache line. To implement prefetching, the microarchitecture has to support a "nonblocking" access to the cache memory hierarchy.

C. Frequency Versus IPC

SPEC rating is a standard measure of performance based on total execution time of a SPEC benchmark suite. Fig. 4 plots the "landscape" of microprocessors based on their performance. The horizontal axis is the megahertz rating of a microprocessor's frequency. The vertical axis is the ratio of SpecINT/MHz, which roughly corresponds to the IPC assuming instruction count remains constant. The different curves represent different levels of performance with increasing performance as we move toward curves in the upper right corner. All points on the same curve represent the same performance level, i.e., SPEC rating. Performance can be increased by either increasing the megahertz rating (moving toward the right) or by increasing the SpecINT/MHz ratio (moving toward the top) or by increasing both. For a given family of microprocessors with the same ISA (and, hence, the same instruction count), the SpecINT/MHz ratio is effectively the measure of their relative IPC.

For example, let us examine the curve that represents the Intel IA32 family of microprocessors. The first point in the curve represents the Intel386 microprocessor. The next point represents the Intel486 microprocessor. The main improvement between these two microprocessors is due to the improvement of IPC. This is obtained through the pipelined design of the Intel486 microprocessor and the introduction of the L1 cache.

The Pentium microprocessor was the first superscalar machine Intel introduced; it also featured branch prediction and split caches. Performance gain came from parallelism as well as reduced stalls. Subsequent proliferation of the Pentium microprocessor involved frequency boosts and relatively small microarchitectural modifications, leaving the IPC almost unchanged.

The next level of performance appears with the Pentium Pro microprocessor. This microprocessor, followed later by the Pentium II and Pentium III microprocessors, is a deeply pipelined superscalar out-of-order microprocessor, which simultaneously improved both frequency and IPC.

Other families of microprocessors show similar trends. New microarchitecture can boost both IPC and frequency while new process technology typically boosts frequency only. In some cases (see the Alpha 21 064), higher frequency may even reduce IPC, but overall performance is still increased.

D. Power Scaling

Each new microprocessor generation introduces higher frequency and higher power consumption. With three times less energy per gate switch and 1.5 times frequency increase, simple shrinking of a microprocessor to a new process can reduce power consumption close to two times. A new microarchitecture, on the other hand, increases work per instruction and wastes energy on wrongly speculated instructions, hence reducing energy efficiency and increasing the total power consumption of the microprocessor.

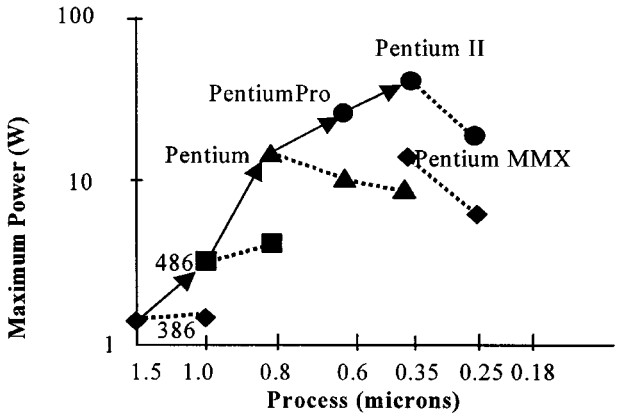

Fig. 5. Maximum power consumption. (Source: S. Borkar [4].)

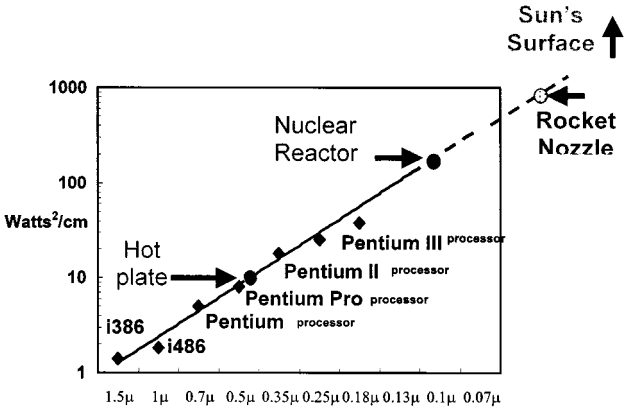

Fig. 6. Power density evolution. (Source: S. Borkar [4].)

Fig. 5 supports the above observation. When a new generation of microprocessors is introduced, it consumes twice the power of the previous generation using the same process. After a while, the microprocessor moves to a new process and improves both power and frequency. Extrapolation of the power dissipation numbers suggests that microprocessors may consume 2000 W in the near future! Up until now, power density was handled using packaging solution to dissipate heat. Fig. 6 indicates that power density will soon become a major problem. Packaging alone will not be able to address power density in a cost-effective manner. Note that chips in 0.6-μm technology used to have power density similar to a (cooking) hot plate (10 W/cm^2). If the trend continues, soon we may experience a chip with the power density of a nuclear power plant or even a rocket nozzle. Furthermore, local hot spots have significantly higher power density than the average, making the situation even worse. Potential solutions to the power density problems are addressed in Section IV.

E. Application Specific Enhancements

The energy crisis encourages the development of application-specific enhancements (ASEs), also termed

focus-MIPS, which aim to achieve better performance and better MIPS for specific classes of applications. Intel's MMX technology and streaming single instruction multiple data (SIMD) extensions are good examples. These SIMD extensions allow a single instruction to perform the same operation on two, four, or eight data elements at the same time, potentially improving performance and reducing power appropriately. Other microprocessor manufacturers have introduced similar extensions—AMD 3DNOW! [27], Sun VIS [28], Motorola Altivec [29] technology, and more.

F. Coming Challenges

Until recently, microarchitects and designers used the increased transistor budget and shorter gate delay to focus mainly on developing faster, bigger, and hotter microprocessors. Lower cost and lower power microprocessors were made available by *waterfalling*—a process shrink that takes yesterday's hot and expensive microprocessor and makes it today's low-power and inexpensive microprocessor. This will change. Power consumed by future high-performing microprocessors will reach levels that cannot suit mobile computers even after a process shrink.

Single thread performance does not scale with frequency and area. Memory latencies and bandwidth do not scale as well either, slowing down memory-bound applications. Deeper pipes allow higher frequency at the cost of longer stalls due to branch misprediction and load-to-use delays. A design challenge is to minimize stalls while still increasing instruction-level parallelism (ILP).

Controlling power becomes a major challenge. Process technology reduces the switching power of a single device, but higher frequency and increased device density increase power dissipation at a faster rate. Power density is reaching levels close to nuclear reactors. Power is becoming the limiting factor in microprocessor design. Microarchitecture should help reduce power.

ASE may be a good power efficient design alternative but at a cost. Developing ASEs requires a lot of work in defining new architectures—writing new development tools and porting existing applications to the new architecture.

With frequencies in excess of 1 GHz, even moving at the speed of light, signals can only travel a short distance. In the future, it may take several cycles for a signal to travel across the chip. Wire delays exhibit an inherent conflict of size and time: bigger structures that may improve performance (e.g., caches and branch predictors) may take longer to access. Incorporating large structures that provide higher ILP without sacrificing frequency will be one of the main challenges.

IV. Future Directions

In this section, we present potential directions and solutions to address these challenges. Many solutions often require difficult tradeoffs. In particular, we can exploit more performance, but at the cost of power and area. While numerous techniques are proposed, the applicability of a certain technique to a certain market segment has to be examined according to the intended usage of the microprocessor.

A. Single Thread Performance Improvements

As stated earlier, performance is dictated by frequency and IPC. Process technology is likely to continue to enable frequency scaling close to 1.5 times per generation. Scaling frequency beyond process advancement requires reduction in the number of logic levels per pipe stage. This number is already low and reducing it further is a complex task with questionable gain. Certain operations, usually those with a very short dependency path, do benefit from higher frequency. Other operations scale more easily by running several of them in parallel rather than by trying to pipeline them or run them faster, even at the cost of duplicating the logic. To address this, there may be different clock domains on the same microprocessor. Simple execution units (e.g., adders) may run at a higher frequency, enabling very fast results forwarding to dependent instructions, thus cutting the program critical path and improving performance. Other operations such as decoding multiple RISC instructions are more easily implemented using slower parallel decoders than faster pipelined ones.

Many techniques have the potential to increase IPC. Techniques are described according to the function they influence—instruction supply, execution, and data supply.

1) Instruction Supply: Instruction supply can be improved by reducing the number of stalls (no instructions are fetched) and by increasing the fetch bandwidth (the number of instructions fetched per cycle). The major sources for stalls are instruction cache misses and mispredicted branches. Instruction cache stalls can be addressed by general memory hierarchy optimization methods. Some relevant techniques are mentioned under data supply later.

Mispredicted branches create long stalls. Even at 5% misprediction rate, shaving an additional percent off may improve performance of a deeply pipelined microprocessor by 4%! More sophisticated *adaptive* or *hybrid branch predictors* have been proposed [30].

Accepting that some branches will be mispredicted, several mechanisms have been proposed to reduce the misprediction penalty. One direction is to reduce the length of the pipeline by maintaining a *decoded instruction cache*. Especially for the IA-32 architecture, eliminating the decoding stage following a misprediction can reduce stall penalty significantly. A similar, but potentially more cost-effective approach suggests holding a decoded version of the alternate program path only.

A more ambitious idea, termed *eager execution*, suggests executing both the taken and the not taken paths of difficult to predict branches [31]. When the branch is resolved, the wrong path is flushed. Eager execution should use a confidence mechanism to determine the likelihood of a branch to be mispredicted [32]. Eager execution may boost performance, but it is a very inefficient technique. Both paths are executed and consume power with only one doing useful work. Moreover, the ability to perform

both paths without performance penalty requires additional resources—increasing cost and complexity.

Another goal is to increase the number of fetched instructions per cycle. To do that in a sustainable manner, one needs to fetch more than one contiguous basic block at a time. The initial attempt is to try to perform multiple branch predictions and multiple block prefetching. These techniques increase fetch bandwidth but involve multiporting and complex steering logic to arrange instructions coming from several different cache lines. *Trace caches*, *block caches*, and *extended basic block caches* (XBCs) were proposed as better alternatives that provide both high bandwidth and low latencies [33]–[36]. These cache-like structures combine instructions from various basic blocks and hold them together. Optionally, the instructions can be stored in decoded form to reduce latency. Traces are combined based on runtime behavior. Traces take time to build but, once used, exhibit very fast fetching and execution on every usage.

Several studies are extending the concept of trace caches and annotate or reorder instructions within the traces (*scheduled trace cache*). This enables faster dependency tracking and instruction scheduling, gaining higher execution bandwidth and lower latency at the cost of more complicated trace building and less flexibility in accessing portions of the trace [37].

Instruction fusion reduces the amount of work by treating several (e.g., two) instructions as a single combined instruction, best thought of as carpooling. The longer the two instructions can travel together in the pipeline, fewer resources are needed and less power is consumed. In particular, in cases where we can build an execution unit that can execute dependent fused instructions together, we can reduce the program critical path and gain performance [38].

2) Execution: Increased instruction-level parallelism starts with a wider machine. More execution units, a larger out-of-order instruction window, and the ability to process more dependency tracking and instruction scheduling per cycle are required, but are not enough. With out-of-order execution, the main limiting factor is not lack of resources; it is data dependency imposed by the data flow among instructions.

For years, data flow was perceived as a theoretical barrier that cannot be broken. Recently, several techniques, collectively termed as *beyond data flow*, showed that this barrier can be broken, resulting in more parallelism and increased performance. Some of these techniques use *super speculation*: they predict results, addresses, or relations among instructions to cut dependencies.

Value prediction [39]–[41] is probably the best example in this domain. Value prediction tries to predict the result of an incoming instruction based on previously computed results and, optionally, the program path (control flow) leading to that instruction. When a prediction is made, the instructions that depend on that result can be dispatched without waiting for the actual computation of the predicted result. The actual computation is done for verification only. Of course, when the prediction is wrong, already executed dependent instructions must be reexecuted. Since mispredictions cost both performance and power, the rate of misprediction has to be minimized.

While one would assume that the probability of predicting a value out of 2^{16} or 2^{32} potential values is rather low, studies show that about 40% and more of the results can be correctly predicted. The simplest predictor is a *last-value predictor*, which predicts that the last result of an instruction will also be the result when the same instruction is executed again. More sophisticated predictors that identify patterns of values achieve even higher prediction rate. A 40% correct prediction rate does not mean that the other 60% are wrong. Confidence mechanisms are used to make sure we predict only those instructions that are likely to be predicted correctly, reducing the number of mispredictions to 1%–2% and less.

Similarly, *memory-address prediction* predicts the memory address needed by a load instruction in case the load address cannot be computed since, for example, it contains a register that is not known yet. When the predictor provides an address with reasonable confidence, the load can be issued immediately, potentially cutting the load-to-use delay. Predicting store addresses can also improve performance. Several address predictors were developed to predict various common address patterns [42].

Instruction reuse is a family of nonspeculative techniques that try to trade off computation with table lookup. The simplest (and oldest) form of this technique is *value cache* [43], where long latency operations (e.g., divide) are cached along with their operands and results. Future occurrences are looked up in the cache for match before execution. The more interesting form of instruction reuse [44] tries to examine a sequence of incoming instructions along with their combined input data to produce the combined result without actually computing it. The technique is nonspeculative and, thus, can provide both performance and power reduction, but it does require large amount of storage.

Memory addresses are computed at execution time, thus dependency among memory accesses is not simple to determine. In particular, the microprocessor cannot determine whether a certain load accesses the same address as a previous store. If the addresses are different, the load can be advanced ahead of the store. If they are the same, the store can forward the result directly to the load without requiring the data to be written first to memory. Microprocessors today tend to take the conservative approach in which all previous store addresses have to be known before the load can be issued. Memory disambiguation techniques [45] have been devised to predict, with high confidence, whether a load collides with a previous store to allow advancing or value forwarding.

Memory disambiguation can be taken a step forward. If the exact load-store pair can be predicted, the load can bypass the store completely, taking its data directly from the producer of the value to be stored. Furthermore, in such a case, the consumers of the load, that is, the instructions that use the loaded value can also take their operands directly from the producer, bypassing both the store and loads and boosting performance significantly [46]. A generalized version of this concept, termed *unified renaming*, uses a value identity de-

tector to determine whether two future results will actually be the same [47]. If so, all references to the later result are converted to reference the former result, thus collapsing the program critical path. Loads following stores are covered by this technique since the load result is identical to the source of the store result.

Register tracking can also be used to avoid dependencies. For example, in the IA-32 architecture, the stack pointer is frequently changed by push/pop operations that add/subtract a constant from the stack pointer value. Every push/pop depends on all previous push/pops. The *register tracking* mechanism computes the stack pointer values at decode, making them all independent of one another [48].

3) Data Supply: Probably the biggest performance challenge today is the speed mismatch between the microprocessor and memory. If outstanding cache misses generated by a single thread, task, or program can be grouped and executed in an overlapped manner, overall performance is improved. The term memory-level parallelism (MLP) [49] refers to the number of outstanding cache misses that can be generated and executed in an overlapping manner. Many microarchitectural techniques have been devised to increase MLP.

Many of the techniques mentioned in the execution section above increase MLP. Value prediction, memory address prediction, and memory disambiguation enable speculative advancing of loads beyond what a conservative ordering would allow, potentially resulting in overlapping loads. Load-store forwarding and unified renaming eliminate dependencies among memory operations and increase memory parallelism. Surely, more resources such as a larger instruction window, more load/store execution units, and more memory ports will increase MLP.

A different technique to increase MLP is *data prefetching*. Data prefetching attempts to guess which portions of memory will be needed and to bring them into the microprocessor ahead of time. In floating-point and multimedia applications, the memory access pattern consists of several simple contiguous *data streams*. Special hardware tracks memory accesses looking for such patterns [50]. When detected, an access to a line so patterned triggers the fetch of the next cache line into an ordinary cache or a prefetch buffer. More sophisticated techniques (e.g., *context-based prefetching* [51]) can also improve other types of applications. Prefetching algorithms try to consider the specific memory-paging behavior [synchronous DRAM (SDRAM), rambus DRAM (RDRAM)] to optimize performance and power. Prefetching costs extra hardware, may overload the buses, and may thrash the cache. Thus, prefetch requests should be issued only when there is enough confidence that they will be used.

Embedded DRAM and similar process/circuit techniques attempt to increase the amount of memory on die. If enough on-die memory is available, it may be used not only as a cache, but also as a faster memory. Operating system/application software may adapt known methods from networked/microprocessor/nonuniform memory access systems to such architectures. Big memory arrays, to a certain degree, are also considered more energy efficient than complex logic. Since memories are more energy efficient than logic, DRAM on die may become an attractive feature in the future.

Of course, much effort is being put into improving traditional caches. With the increase in microprocessor frequency, more levels of memories will be integrated on the die; on the other hand, the size of the L1 cache will be reduced to keep up with the ultrahigh frequency. Such memory organization may cause a coherency problem within the chip. This problem may be resolved using similar methods that resolve the coherency problem in a multiprocessor (MP) system today. In addition, researches will continue to look for better replacement algorithms and additional structures to hold streaming data so as to eliminate cache thrashing.

4) Pushing ILP/MLP Further: Previous sections discuss ways to increase parallelism in a single thread. The next level of parallelism is achieved by running several threads in parallel, thus achieving increased parallelism at a coarser grain level. This section briefly covers several ideas that aim at speeding up single thread performance by splitting it into several threads. Improving parallelism by running several independent threads simultaneously is addressed at the end of this section.

Selective preprocessing of instructions was suggested to reduce branch misprediction and cache miss stalls. The idea is to use a separate "processor" or thread to run ahead of the actual core processor and execute only those instructions that are needed to resolve the difficult-to-predict branches or compute the addresses of loads which are likely to cause cache misses. Early execution of such loads acts very much like data prefetch. Precomputed correct control information is passed to the real thread, which can reduce the number of branch mispredictions.

Dynamic multithreading [52] attempts to speculatively execute instructions following a call or loop-end along with the called procedure or the loop body (*thread speculation*). Dependencies among the executed instruction blocks may be circumvented, by value prediction. With sufficient resources, useful work can be done speculatively. Even if some of the work is found erroneous and has to be redone, prefetching side effects may help performance.

The *multiscalar* technique [53] extends the concept of speculative threads beyond dynamic multithreading. The classic *multiscalar* idea calls for the help of the user and/or the compiler to define potential threads and to provide dependency information. A thread is ideally a self-contained piece of work with as few inputs and outputs as possible. A separate mechanism verifies the user/compiler provided dependencies (e.g., memory conflicts).

B. Process Technology Challenges

Besides pushing the performance envelope of a single-thread instruction stream, microarchitects are faced with some process technology related challenges.

1) Power Challenge: So far we have mainly considered the active power and ignored the *leakage power*. Leakage power is generated by current running in gates and wires

even when there is no activity. There are three leakage sources: 1) the subthreshold drain-source leakage; 2) junction leakage; and 3) gate-substrate leakage. To gain performance, lower threshold voltage (Vt) is used. Lower Vt, however, increases leakage power (subthreshold). Also, higher power densities increase device temperature, which in turn increases leakage power (junction). As technology scales, oxide thickness is also scaled which causes more leakage power (gate). As such, controlling power density is becoming even more crucial in future deep submicron technologies. When adding complexity to exploit performance, one must understand and consider the power impact. In addition, one needs to target the reduction of standby power dissipation. There are three general approaches to power reduction: 1) increase energy efficiency; 2) conserve energy when a module is not in use; and 3) recycle and reuse. Following are several possible suggested solutions.

Trading frequency for IPC is generally perceived as energy efficient. Higher IPC with lower frequency and a lower voltage can achieve the same performance but with less power. This is not always accurate. Usually, higher IPC requires additional complexity that involves additional work to be done on each instruction (e.g., dependency tracking). Moreover, many instructions are speculatively processed and later discarded or reexecuted (e.g., wrong data was predicted). Some performance-related techniques (e.g., load bypassing) do reduce the work performed per instruction and thus are more energy friendly. Better predictors with confidence-level evaluation can be combined to control speculation and avoid the waste of energy.

It is common sense to shut down circuits when they are not in use to conserve energy. Shutting down logic and circuits can be applied to part or the whole microprocessor. However, the sharp current swings associated with frequent gating cause "di/dt" noise. This noise is also referred to as the simultaneous switching noise and is characterized by the equation: $dV = Ldi/dt$, where dV is the voltage fluctuation, L is the inductance of the wire supplying the current, di is the current change, and dt is the rise or fall time of gates. One potential area of research is to determine the right time to shut down blocks to avoid sharp current swings. Prediction algorithms based on program dynamic behavior may be useful.

Dynamic voltage/frequency scaling [54] can reduce energy consumption in microprocessors without impacting the peak performance. This approach varies the microprocessor voltage or frequency under software control to meet dynamically varying performance requirements. This method conserves energy when maximum throughput is not required. Intel's SpeedStep technology [55] and Transmeta's LongRun technology [56], [57] both use dynamic voltage/frequency scaling.

Some microarchitecture ideas can actually reduce the number of instructions [58] processed, thus saving energy. Instruction reuse, mentioned earlier, saves previously executed results in a table and can trade repeated execution with table lookup, potentially increasing performance and saving energy.

2) Wire Delays: Wiring is becoming a bottleneck for frequency improvement. Wires do not scale like gates. Wire delay depends on the wire resistance, which is inversely proportional to the cross section of a wire, and the wire spacing distance causing the coupling capacitance between sides of adjacent wires. If all dimension of a wire are scaled (width, thickness, and length), wire delay remains fixed (in nanoseconds) for the scaled length [59]. Increased frequency and complexity cause wire delays to grow relative to the cycle time.

Improvements in interconnection technology such as copper wires, new materials (lower dielectric), and interconnect configurations will provide the high performance anticipated for one or two more generations [60]. However, eventually while short local wires will continue to be of little cost, long global interconnects will be more and more expensive. Microarchitecture should reduce the usage of long signal paths. Clustering is considered a general approach for wire delay reduction. Large structures (e.g., large register files) are divided into several structures. Intracluster communication is short (single cycle), while intercluster communication may take longer (several cycles). Microarchitecture techniques can reduce intercluster traffic [61]. This concept is extended to chip multiprocessors [(CMPs), to be discussed later].

A more radical approach in coping with on-chip wire delay is to eliminate global on-chip communication completely. One of the global communications is the clock. There has been a resurgence of interests in *asynchronous* processor design. There are many advantages claimed by researchers, such as modularity in design, ease of composability with simplified interfacing, and exploitation of average performance of all the individual components. But the complete elimination of the clock seems too drastic. A more practical approach suggests systems employing the *globally asynchronous* but *locally synchronous* (GALS) clocking approach.

Another way to maintain high frequency is to convert long data wires into synchronous pipelines. One such approach is the counterflow microarchitecture [62], in which results are pipelined and flow in an opposing direction as instructions.

3) Soft Error Rate: Soft errors are inevitable with technology scaling. Alpha particles and secondary cosmic rays, especially neutrons, occur naturally [63]. When they strike, they generate electron-hole pairs. A soft error occurs when the amount of electrons collected exceeds the critical charge that upsets the stored data. This stored data can be either a memory bit or a dynamic node of a Domino circuit. As we reduce supply voltage and capacitance, it takes a smaller charge ($Q = CV$) to upset the stored data. Thus, the soft error rate (SER) increases by two times per generation. For a static latch, the induced noise scales with the supply voltage, but the current used to restore the logic is scaled by 0.7 every generation. Thus, the susceptibility increases by 43% per generation. Currently (0.25-μm process), a typical on-chip SRAM array has an acceptable SER rate of about one failure per few decades [64]. This will become a problem in a few generations. Attempts to reduce the soft error rate by increasing

the capacitance on the node will impede performance and increase power consumption.

Soft errors cannot be prevented, but effort should be made to detect and correct them. Methods for detecting and recovering storage errors are well known and widely deployed. Parity bits can detect single bit errors. Error correcting code (ECC) such as Hamming code can correct single-bit errors and detect double-bit errors. ECC has been widely used in DRAM main memory and has appeared in larger second-level caches. Organizing arrays in a distributed way reduces the effect of an alpha particle strike to a single-bit error per word. Detecting and recovering from errors in logic are more difficult. There is no known general simple inexpensive way to detect logic errors such as parity bits. A simple yet expensive way is to duplicate the entire logic, compare the results, and redo the work if results disagree. Sohi and Franklin [65] proposed executing each instruction twice in a time redundancy manner . Rotenberg [66] proposed executing a redundant thread on a simultaneous multithreading (SMT) or trace processor. A more general idea [67] suggests building a less complex, lower frequency but highly parallel checker that will verify a group of results at a time. Such a checker can be highly parallel since it has all the operands and results needed for the verification, so it does not have to reexecute the instructions according to their original dependencies. A variant of this checker idea is suggested to also help address some corner design mistakes that rarely happens and are not uncovered during the design validation effort. Such a checker can detect a mismatch due to a design flaw. In that case, it is assumed that the checker, being simpler, produces the right solution and execution is restarted following the erroneous instruction.

C. Cutting the Gordian Knot—Higher Level Parallelism

Several approaches have been proposed to go beyond optimizing single-thread performance (latency) and to exploit overall higher performance (throughput) at better energy efficiency.

1) Simultaneous Multithreading: Even with super speculation, the realized parallelism is still limited and more resources yield only diminishing return. SMT suggests sharing the resources among instructions coming from several threads. SMT microprocessors can run threads coming from either multithreaded parallel programs or individual independent programs in a multiprogramming workload [68]. SMT provides more parallelism by exploiting the parallelism from each thread as well as the combined parallelism among the threads. SMT allows executing useful instructions from an alternate thread rather than overly speculating or stalling a single thread. As such, an SMT microprocessor has better performance and is more energy efficient than a single-threaded microprocessor. An SMT microprocessor can better overlap stalls due to branch mispredictions or cache misses. SMT increases the combined ILP and MLP of all running threads. SMT, however, cannot increase and sometimes may even decrease the performance of a single thread. Implementation complexity of SMT is also an issue.

2) Chip Multiprocessor (CMP): CMP is a simple yet very powerful technique to obtain more performance in a power-efficient manner [69]. The idea is to put several microprocessors on a single die. The performance of small-scale CMP scales close to linear with the number of microprocessors and is likely to exceed the performance of an equivalent MP system. CMP is an attractive option to use when moving to a new process technology. New process technology allows us to shrink and duplicate our best existing microprocessor on the same silicon die, thus doubling the performance at the same power. Since interchip signals are quite confined and rare, CMP also addresses the issue of wire delays (Section IV-B2). Duplicating an existing microprocessor into CMP is a relatively easy task since microarchitecture work is usually limited to speed up the interchip communication. Such work involves, e.g., using faster on chip buses or sharing the large on-chip caches. As with SMT, CMP increases the combined ILP and MLP of all running processes; but, as opposed to SMT, single-thread performance is not affected. With time, operating systems and applications will better utilize multithreading capabilities, making CMP and SMT attractive solutions. Note that CMP and SMT can coexist—a CMP die can integrate several SMT microprocessors.

3) Architecture: As mentioned earlier, an order-of-magnitude improvement in performance and energy has been achieved by introducing ASEs such as Intel's MMX technology and streaming SIMD extensions as well as digital signal processor (DSP) architectures. This trend will continue. One can envision a system with a general-purpose microprocessor taking care of the entire control program while DSP-like extensions or a DSP coprocessor (optionally integrated on the same die along with the main processor) handles communication and streaming multimedia input.

The available transistor budget calls for integrating more functions on chip, reducing the number of components in the system overall. Such integration allows higher performance at a lower overall system cost. This proximity allows, for example, integrating memory controllers, graphics engines, and embedded DRAM on the same die and providing a very fast communication among all these components [70].

D. Putting it All Together

We have presented many techniques for increasing performance, reducing power, overcoming wire delay and SER, and reducing costs. Unfortunately, most of the techniques involve difficult tradeoffs. Which techniques are better? The answer depends on the desired target parameters and the market segment the product is aimed at. Power is going to be a major challenge. With power getting the center stage in all market segments, deployment of other enhancements, which involve increase in power, may slow down.

One thing is sure: with increased level of diversification, the same microarchitecture will no longer be employed to target different segments by relying on process technology

shrink. That is, today's high-end microprocessor may not be used as tomorrow's low-power/low-cost solutions.

V. Conclusion

Looking forward, besides the current challenges to increase computation power, new ones are emerging: power, reliability, and wire delay. Moreover, we are approaching the point of diminishing returns in increasing single-thread general-purpose performance. Yet, as described above, there are a number of microarchitecture and architecture techniques that can address these challenges. Thus, improvement in the computation power of microprocessors will probably continue over the next five to seven years as these techniques work their way into microprocessor products. For the last 20 years, there has always been a performance "wall" five to seven years out into the future. Today is no different. Historically, breakthroughs have always been found that pushed back the wall. However, as time passes, it has become harder and harder to push the wall back. Are the ultimate limits of computing approaching? There is no conclusive evidence to argue in either the positive or negative direction. However, it is our firm belief that if anything, we are closer to a new beginning than an end.

References

[1] SPEC, Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/

[2] BAPCo. SYSmark® for Windows NT ver. 4.0. [Online]. Available: http://www.bapco.com/nt4-0.htm

[3] S. Yang *et al.*, "A high performance 180nm generation logic technology," in *Proc. IEDM*, Dec. 1998, pp. 197–200.

[4] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, pp. 23–29, July–Aug. 1999.

[5] Y. N. Patt, S. J. Patel, M. Evers, D. H. Friendly, and J. Stark, "One billion transistors, one uniprocessor, one chip," *IEEE Computer*, vol. 30, pp. 51–57, Sept. 1997.

[6] D. B. Papworth, "Tuning the Pentium Pro microarchitecture," *IEEE Micro*, vol. 16, pp. 8–15, Apr. 1996.

[7] M. Flynn, P. Hung, and K. W. Rudd, "Deep-submicron microprocessor design issues," *IEEE Micro*, vol. 19, pp. 11–22, July-Aug. 1999.

[8] A. J. Smith, "Cache memories," *Computing Surveys*, vol. 14, no. 3, pp. 473–530, Sept. 1982.

[9] S. A. Przybylski, *Cache and Memory Hierarchy Design: A Performance-Directed Approach*. San Mateo, CA: Morgan Kaufmann, 1990.

[10] T.-Y. Yeh and Y. N. Patt, "Branch history table indexing to prevent pipeline bubbles in wide-issue superscalar processors," in *Proc. 26th Annu. Int. Symp. Microarchitecture*, Dec. 1993, pp. 164–175.

[11] S. McFarling, "Combining branch predictors," Compaq/Western Research Laboratories, Tech. Note TN-36, June 1993.

[12] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," in *Proc. 24th. Int. Symp. Computer Architecture*, June 1997, pp. 1–13.

[13] L. Gwennap, "Speed kills? Not for RISC processors," *Microprocessor Rep.*, vol. 7, Mar. 8, 1993.

[14] ——, "Brainiacs, speed demons, and farewell," *Microprocessor Rep.*, vol. 13, Dec. 27, 1999.

[15] P. E. Gronowski *et al.*, "High-performance microprocessor design," *IEEE J. Solid-State Circuits*, vol. 33, pp. 676–686, May 1998.

[16] J. A. Fisher, "Very long instruction word architectures and ELI-512," in *Proc. 10th Symp. Computer Architecture*, June 1983, pp. 140–150.

[17] M. S. Schlansker and B. R. Rau, "EPIC: Explicitly parallel instruction computing," *Computer*, vol. 33, no. 2, pp. 37–45, Feb. 2000.

[18] A. Saini, "Design of the Intel Pentium processor," in *Proc. ICCD*, Oct. 1993, pp. 258–261.

[19] B. Shriver and B. Smith, *The Anatomy of a High-Performance Microprocessor—A System Perspective*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1998.

[20] J. E. Thornton, "Parallel operation in the control data 6600," in *Proc. AFIPS Fall Joint Computer Conf., Part II*, vol. 26, 1964, pp. 33–40.

[21] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units," *IBM J. Res. Dev.*, vol. 11, pp. 25–33, Jan. 1967.

[22] D. W. Wall, "Limits of instruction-level parallelism," Western Research Laboratory, Digital Equipment, Corp., Res. Rep. 93/6, 1993.

[23] M. Bekerman, A. Mendelson, and G. Sheaffer, "Performance and hardware complexity tradeoffs in designing multithreaded architectures," in *Proc. Parallel Architectures and Compilation Techniques (PACT)*, Oct. 1996, pp. 24–34.

[24] S. Fischer *et al.*, "A 600 MHz IA-32 microprocessor with enhanced data streaming for graphics and video," in *IEEE Int. Solid-State Circuits Conf.*, Feb. 1999, pp. 98–101.

[25] R. E. Kessler, "The Alpha 21 264 microprocessor," *IEEE Micro*, vol. 19, pp. 24–36, Mar./Apr. 1999.

[26] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Proc. Int. Symp. Computer Architecture*, May 1990, pp. 364–373.

[27] S. Oberman *et al.*, "AMD 3DNow! technology: Architecture and implementations," *IEEE Micro*, vol. 19, pp. 37–48, Mar.–Apr. 1999.

[28] P. Ranganathan *et al.*, "Performance of image and video processing with general-purpose processors and media ISA extensions," in *Proc. 26th Int. Symp. Computer Architecture*, May 1999, pp. 124–135.

[29] J. Tyler *et al.*, "AltiVec™: Bringing vector technology to the PowerPC™ processor family," in *IEEE Int. Conf. Performance, Computing Communications*, Feb. 1999, pp. 437–444.

[30] P.-Y. Chang, E. Hao, and Y. N. Patt, "Alternative implementations of hybrid branch predictors," in *Proc. 28th Annu. Int. Symp. Microarchitecture*, Nov./Dec. 1995, pp. 252–257.

[31] A. K. Uht, V. Sindagi, and K. Hall, "Disjoint eager execution: An optimal form of speculative execution," in *Proc. 28th Annu. Int. Symp. Microarchitecture*, Nov./Dec. 1995, pp. 313–325.

[32] E. Jacobsen, E. Rotenberg, and J. Smith, "Assigning confidence to conditional branch predictions," in *Proc. 29th Annu. Int. Symp. Microarchitecture*, Dec. 1996, pp. 142–152.

[33] A. Peleg and U. Weiser, "Dynamic flow instruction cache memory organized around trace segments independent of virtual address line," U.S. Patent 5 381 533, 1994.

[34] E. Rotenberg, S. Bennett, and J. E. Smith, "Trace cache: A low latency approach to high bandwidth instruction fetching," in *Proc. 29th Annu. Int. Symp. Microarchitecture*, Dec. 1996, pp. 24–34.

[35] S. Jourdan, L. Rappoport, Y. Almog, M. Erez, A. Yoaz, and R. Ronen, "Extended block cache," in *Proc. HPCA 6*, Jan. 2000, pp. 61–70.

[36] B. Black, B. Rychlik, and J. P. Shen, "The block-based trace cache," in *Proc. 26th Int. Symp. Computer Architecture*, May 1999, pp. 196–207.

[37] D. H. Friendly, S. J. Patel, and Y. N. Patt, "Putting the fill unit to work: Dynamic optimizations for trace cache microprocessors," in *Proc. 31st Annu. Int. Symp. Microarchitecture*, Nov./Dec. 1998, pp. 173–181.

[38] N. Malik, R. J. Eickemeyer, and S. Vassiliadis, "Architectural effects on dual instruction issue with interlock collapsing ALUs," in *Proc. 12th Annu. Int. Phoenix Conf. Computers Communications*, Mar. 1993, pp. 42–48.

[39] M. H. Lipasti and J. P. Shen, "Exceeding the dataflow limit via value prediction," in *Proc. 29th Annu. Int. Symp. Microarchitecture*, Dec. 1996, pp. 226–237.

[40] F. Gabbay and A. Mendelson, "Using value prediction to increase the power of speculative execution hardware," *ACM Trans. Comput. Syst.*, vol. 16, no. 3, pp. 234–270, Aug. 1998.

[41] Y. Sazeides and J. E. Smith, "The predictability of data values," in *Proc. 30th Annu. Int. Symp. Microarchitecture*, Dec. 1997, pp. 248–258.

[42] M. Bekerman, S. Jourdan, R. Ronen, G. Kirshenboim, L. Rappoport, A. Yoaz, and U. Weiser, "Correlated load-address predictors," in *Proc. 26th Int. Symp. Computer Architecture*, May 1999, pp. 54–63.

[43] S. E. Richardson, "Caching function results: Faster arithmetic by avoiding unnecessary computation," Sun Microsystems Laboratories, Tech. Rep. SMLI TR-92-1, Sept. 1992.

[44] A. Sodani and G. S. Sohi, "Dynamic instruction reuse," in *Proc. 24th Int. Symp. Computer Architecture*, June 1997, pp. 235–245.

[45] A. Yoaz, M. Erez, R. Ronen, and S. Jourdan, "Speculation techniques for improving load related instruction scheduling," in *Proc. 26th Int. Symp. Computer Architecture*, May 1999, pp. 42–53.
[46] A. Moshovos and G. S. Sohi, "Streamlining inter-operation memory communication via data dependence prediction," in *Proc. 30th Annu. Int. Symp. Microarchitecture*, Dec. 1997, pp. 235–245.
[47] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, and A. Yoaz, "A novel renaming scheme to exploit value temporal locality through physical register reuse and unification," in *Proc. 31st Annu. Int. Symp. Microarchitecture*, Nov. 1998, pp. 216–225.
[48] M. Bekerman, A. Yoaz, F. Gabbay, S. Jourdan, M. Kalaev, and R. Ronen, "Early load address resolution via register tracking," in *Proc. 27th Int. Symp. Computer Architecture*, June 2000, pp. 306–315.
[49] A. Glew, "MLP yes! ILP no," in *Proc. ASPLOS*, San Jose, CA, Oct. 1998.
[50] T.-F. Chen and J.L. Baer, "Effective hardware based data prefetching for high-performance processors," *IEEE Trans. Computers*, vol. 44, pp. 609–623, May 1995.
[51] DRAM-Page Based Prediction and Prefetching, G. Kedem and H. Yu. [Online]. Available: http://kedem.cs.duke.edu/HPMA/Prefetching/index.html
[52] H. Akkary and M. A. Driscoll, "A dynamic multithreading processor," in *Proc. 31st Annu. Int. Symp. Microarchitecture*, Nov./Dec. 1998, pp. 226–236.
[53] G. S. Sohi, S. E. Breach, and T. N. Vijaykumar, "Multiscalar processors," in *Proc. 22nd Int. Symp. Computer Architecture*, June 1995, pp. 414–425.
[54] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," in *Proc. Int. Symp. Low Power Electronics Design*, Aug. 1998, pp. 76–81.
[55] Pentium® III Processor Mobile Module: Mobile Module Connector 2 (MMC-2) Featuring Intel® Speedstep™) Technology [Online]. Available: http://developer.intel.com/design/mobile/datashts/243356.htm
[56] L. Geppert and T. S. Perry, "Transmeta's magic show [microprocessor chips]," *IEEE Spectrum*, vol. 37, pp. 26–33, May 2000.
[57] T. R. Halfhill, "Transmeta breaks X86 low-power barrier," *Microprocessor Rep.*, vol. 14, Feb. 14, 2000.
[58] J. Bunda, D. Fussell, and W. C. Athas, "Energy-efficient instruction set architecture for CMOS microprocessors," in *Proc. 28th Hawaii Int. Conf. System Sciences, Vol. II*, Jan. 1995, pp. 298–305.
[59] D. Matzke, "Will physical scalability sabotage performance gains," *IEEE Computer*, vol. 30, pp. 37–39, Sept. 1997.
[60] M. T. Bohr, "Interconnect scaling-the real limiter to high performance ULSI," in *Proc. Int. Electron Devices Meeting*, Dec. 1995, pp. 241–244.
[61] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic, "The multicluster architecture: Reducing cycle time through partitioning," in *Proc. 30th. Annu. Symp. Microarchitecture*, Dec. 1997, pp. 149–159.
[62] M. F. Miller, K. J. Janik, and S. L. Lu, "Non-stalling counterflow architecture," in *Proc. 4th Int. Symp. High-Performance Computer Architecture*, Feb. 1998, pp. 334–341.
[63] L. Lantz II, "Soft errors induced by alpha particles," *IEEE Trans. Rel.*, vol. 45, pp. 174–179, June 1996.
[64] C. Dai, N. Hakim, S. Hareland, J. Maiz, and S.-W. Lee, "Alpha-SER modeling and simulation for sub-0.25 μm CMOS technology," in *Proc. Symp. VLSI Technology*, June 1999, pp. 81–82.
[65] G. S. Sohi, M. Franklin, and K. K. Saluja, "A study of time-redundant fault tolerance techniques for high-performance pipelined computers," in *Proc. FTCS*, June 1989, pp. 436–443.
[66] E. Rotenberg, "AR-SMT: A microarchitectural approach to fault tolerance in microprocessors," in *Proc. FTCS*, 1999, pp. 84–91.
[67] T. M. Austin, "DIVA: A reliable substrate for deep submicron microarchitecture design," in *Proc. 32nd Annu. Int. Symp. Microarchitecture*, Nov. 1999, pp. 196–207.
[68] J. L. Lo, J. S. Emer, H. M. Levy, R. L. Stamm, D. M. Tullsen, and S. J. Eggers, "Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading," *ACM Trans. Comput. Syst.*, vol. 15, pp. 322–354, Aug. 1997.
[69] K. Diefendorff, "Power4 focuses on memory bandwidth," *Microprocessor Rep.*, vol. 13, Oct. 6, 1999.
[70] T. R. Halfhill, "National unveils appliance on a chip," *Microprocessor Rep.*, vol. 13, Aug. 2, 1999.

Ronny Ronen (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science from the Technion, Israel Institute of Technology, Haifa, Israel, in 1978 and 1979, respectively.

He is currently a Principal Engineer with the Microprocessor Research Laboratories, Intel Corporation, Haifa, Israel, focusing on microarchitecture research. He has been with Intel for over 20 years and has led the compiler and performance simulation activities in the Intel Israel Software Department.

Avi Mendelson (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science from the Technion, Israel Institute of Technology, Haifa, Israel, in 1979 and 1982, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Massachusetts, Amherst, in 1990.

He was with the Electrical Engineering Department at the Technion from 1990 to 1997 and with National Semiconductor from 1997 to 1999. He is currently a Research Staff Member with the Microprocessor Research Laboratories, Intel Corporation, Haifa, Israel. His current research interests include computer architecture, operating systems, and system-level performance analysis.

Dr. Mendelson is a Member of the ACM.

Konrad Lai received the B.S. degree in electrical engineering from Princeton University, Princeton, NJ, in 1976, and the M.S. degree in computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1978.

He is currently a Principal Researcher and manages the Microarchitecture Laboratory of the Microprocessor Research Laboratories, Intel Corporation, Hillsboro, OR. He has been with Intel for over 20 years, working on microprocessor, memory, and system architecture. He holds 25 patents and has authored or coauthored eight papers.

Mr. Konrad is a Member of ACM and the IEEE Computer Society.

Shih-Lien Lu (Member, IEEE) received the B.S. degree from the University of California, Berkeley, in 1980, and the M.S. and Ph.D. degrees from the University of California, Los Angeles, in 1984 and 1991, respectively.

From 1985 to 1991, he was with the Information Science Institute, University of Southern California, working on the MOSIS project. He joined the Department of Electrical and Computer Engineering, Oregon State University, Corvallis, as an Assistant Professor in 1991 and became an Associate Professor in 1996. Since 1999, he has been with the Microprocessor Research Laboratories, Intel Corporation, Hillsboro, OR. His current research interests include computer microarchitecture, embedded systems, self-timed circuits, and very large scale integration systems design.

Dr. Lu received the Oregon State University College of Engineering Carter Award for outstanding and inspirational teaching in 1995 and Oregon State University College of Engineering/Electrical and Computer Engineering Engelbrecht Award in 1996. He is a Member of the IEEE Computer Society.

Fred Pollack received the B.S. degree in mathematics from the University of Florida in 1970. He was enrolled in the Ph.D. computer science program at Carnegie Mellon University, Pittsburgh, PA, from 1970 to 1977, and completed all requirements except his thesis.

He retired from Intel Corporation, Santa Clara, CA, in February 2001, after 23 years. He was most recently the Director of Intel Architecture Strategic Planning. From May 1999 to October 2000, he was the Director of Intel's Microprocessor Research Laboratories, which focuses on several different areas including computer architecture, compilers, circuits, graphics, video, security, and speech and handwriting recognition. From 1992 to early 1999, he was Director of the MAP group in MPG, the division that is responsible for all Intel platform architecture and performance analysis in which he was also responsible for directing the planning of Intel's future microprocessors. From 1990 to 1992, he was the Architecture Manager for the Pentium Pro microprocessor. He joined Intel in 1978. Earlier assignments included Manager of the i960 architecture and Chief Architect for an advanced object-oriented distributed operating system.

Mr. Pollack was named an Intel Fellow in December 1992.

John P. Shen (Fellow, IEEE) received the B.S. degree from the University of Michigan, Ann Arbor, and the M.S. and Ph.D. degrees from the University of Southern California, Los Angeles, all in electrical engineering.

He currently directs the Microarchitecture Laboratory of the Microprocessor Research Laboratories, Intel Corporation, in Hillsboro, OR, and Santa Clara, CA. He spent several years in the aerospace industry working at Hughes and TRW. From 1981 to 2000, he was on the Faculty of the Electrical and Computer Engineering Department at Carnegie Mellon University, where he headed the Microarchitecture Research Team (CMuART). He is currently writing a textbook on "Superscalar Processor Design."