Power-Aware Microarchitecture:

Design and Modeling Challenges for Next-Generation Microprocessors

THE ABILITY TO ESTIMATE POWER CONSUMPTION DURING EARLY-STAGE DEFINITION AND TRADE-OFF STUDIES IS A KEY NEW METHODOLOGY ENHANCEMENT. OPPORTUNITIES FOR SAVING POWER CAN BE EXPOSED VIA MICROARCHITECTURE-LEVEL MODELING, PARTICULARLY THROUGH CLOCK-GATING AND DYNAMIC ADAPTATION.

David M. Brooks
Pradip Bose
Stanley E. Schuster
Hans Jacobson
Prabhakar N. Kudva
Alper Buyuktosunoglu
John-David Wellman
Victor Zyuban
Manish Gupta
Peter W. Cook
IBM T.J. Watson
Research Center

Power dissipation limits have emerged as a major constraint in the design of microprocessors. At the low end of the performance spectrum, namely in the world of handheld and portable devices or systems, power has always dominated over performance (execution time) as the primary design issue. Battery life and system cost constraints drive the design team to consider power over performance in such a scenario.

Increasingly, however, power is also a key design issue in the workstation and server markets (see Gowan et al.)[1] In this high-end arena the increasing microarchitectural complexities, clock frequencies, and die sizes push the chip-level—and hence the system-level—power consumption to such levels that traditionally air-cooled multiprocessor server boxes may soon need budgets for liquid-cooling or refrigeration hardware. This need is likely to cause a break point—with a step upward—in the ever-decreasing price-performance ratio curve. As such, a design team that considers power consumption and dissipation limits early in the design cycle and can thereby adopt an inherently lower power microarchitectural line will have a definite edge over competing teams.

Thus far, most of the work done in the area of high-level power estimation has been focused at the register-transfer-level (RTL) description in the processor design flow. Only recently have we seen a surge of interest in estimating power at the microarchitecture definition stage, and specific work on power-efficient microarchitecture design has been reported.[2-8]

Here, we describe the approach of using energy-enabled performance simulators in early design. We examine some of the emerging paradigms in processor design and comment on their inherent power-performance characteristics.

Power-performance efficiency

See the "Power-performance fundamentals" box. The most common (and perhaps obvious) metric to characterize the power-performance efficiency of a microprocessor is a simple ratio, such as MIPS (million instructions per second)/watts (W). This attempts to quantify efficiency by projecting the performance achieved or gained (measured in MIPS) for every watt of power consumed. Clearly, the higher the number, the "better" the machine is.

While this approach seems a reasonable

Power-performance fundamentals at the microarchitecture level

At the elementary transistor gate level, we can formulate total power dissipation as the sum of three major components: switching loss, leakage, and short-circuit loss.[1-4]

$$PW_{\text{device}} = (1/2) C V_{DD} V_{\text{swing}}\, af + I_{\text{leakage}} V_{DD} + I_{sc} V_{DD} \quad \text{(A)}$$

Here, C is the output capacitance, V_{DD} is the supply voltage, f is the chip clock frequency, and a is the activity factor ($0 < a < 1$) that determines the device switching frequency. V_{swing} is the voltage swing across the output capacitor. I_{leakage} is the leakage current, and I_{sc} is the average short-circuit current. The literature often approximates V_{swing} as equal to V_{DD} (or simply V for short), making the switching loss around $(1/2)CV^2af$. Also for current ranges of V_{DD} (say, 1 volt to 3 volts) switching loss, $(1/2)CV^2af$ remains the dominant component.[2] So as a first-order approximation for the whole chip we may formulate the power dissipation as

$$PW_{\text{chip}} = (1/2)\left[\sum_i C_i V_i^2 a_i f_i\right] \quad \text{(B)}$$

C_i, V_i, a_i, and f_i are unit- or block-specific average values. The summation is taken over all blocks or units i, at the microarchitecture level (instruction cache, data cache, integer unit, floating-point unit, load-store unit, register files, and buses). For the voltage range considered, the operating frequency is roughly proportional to the supply voltage; C remains roughly the same if we keep the same design, but scale the voltage. If a single voltage and clock frequency is used for the whole chip, the formula reduces to

$$PW_{\text{chip}} = V^3 \left(\sum_i K_i^v a_i\right) = f^3 \left(\sum_i K_i^f a_i\right) \quad \text{(C)}$$

where K's are unit- or block-specific constants. If we consider the worst-case activity factor for each unit i—that is, if $ai = 1$ for all i, then

$$PW_{\text{chip}} = K_v V^3 = K_f f^3 \quad \text{(D)}$$

where K_v and K_f are design-specific constants.

That equation leads to the so-called cube-root rule.[2] This points to the single most efficient method for reducing power dissipation for a processor designed to operate at high frequency: reduce the voltage (and hence the frequency). This is the primary mechanism of power control in Transmeta's Crusoe chip (http://www.transmeta.com). There's a limit, however, on how much V_{DD} can be reduced (for a given technology), which has to do with manufacturability and circuit reliability issues. Thus, a combination of microarchitecture and circuit techniques to reduce power consumption—without necessarily employing multiple or variable supply voltages—is of special relevance.

Performance basics

The most straightforward metric for measuring performance is the execution time of a representative workload mix on the target processor. We can write the execution time as

$$T = PL \times CPI \times CT = PL \times CPI \times (1/f) \quad \text{(E)}$$

Here, PL is the dynamic path length of the program mix, measured as the number of machine instructions executed. CPI is the average processor cycles per instruction incurred in executing the program mix, and CT is the processor cycle time (measured in seconds per cycle) whose inverse determines clock frequency f. Since performance increases with decreasing T, we may formulate performance PF as

$$PF_{\text{chip}} = K_{pf}\, f = K_{pv} V \quad \text{(F)}$$

Here, the K's are constants for a given microarchitecture-compiler implementation. The K_{pf} value stands for the average number of machine instructions executed per cycle on the machine being measured. PF_{chip} in this case is measured in MIPS.

Selecting a suite of publicly available benchmark programs that everyone accepts as being representative of real-world workloads is difficult. Adopting a noncontroversial weighted mix is also not easy. For the commonly used SPEC benchmark suite (http://www.specbench.org) the SPECmark rating for each class is derived as a geometric mean of execution time ratios for the programs within that class. Each ratio is calculated as the speedup with respect to execution time on a specified reference machine. If we believe in the particular benchmark suite, this method has the advantage of allowing us to rank different machines unambiguously from a performance viewpoint. That is, we can show the ranking as independent of the reference machine used in such a formulation.

References

1. V. Zyuban, "*Inherently Lower-Power High Performance Super Scalar Architectures*," PhD dissertation, Univ. of Notre Dame, Dept. of Computer Science and Engineering, 2000.
2. M.J. Flynn et al., "Deep-Submicron Microprocessor Design Issues," *IEEE Micro*, Vol. 19, No. 4, July/Aug. 1999, pp. 11-22.
3. S. Borkar, "Design Challenges of Technology Scaling," *IEEE Micro*, Vol. 19, No. 4, July/Aug. 1999, pp. 23-29.
4. R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *IEEE J. Solid-State Circuits*, Vol. 31, No. 9, Sept. 1996, pp. 1277-1284.

choice for some purposes, there are strong arguments against it in many cases, especially when it comes to characterizing higher end processors. Performance has typically been the key driver of such server-class designs, and cost or efficiency issues have been of secondary importance. Specifically, a design team may well choose a higher frequency design point (which

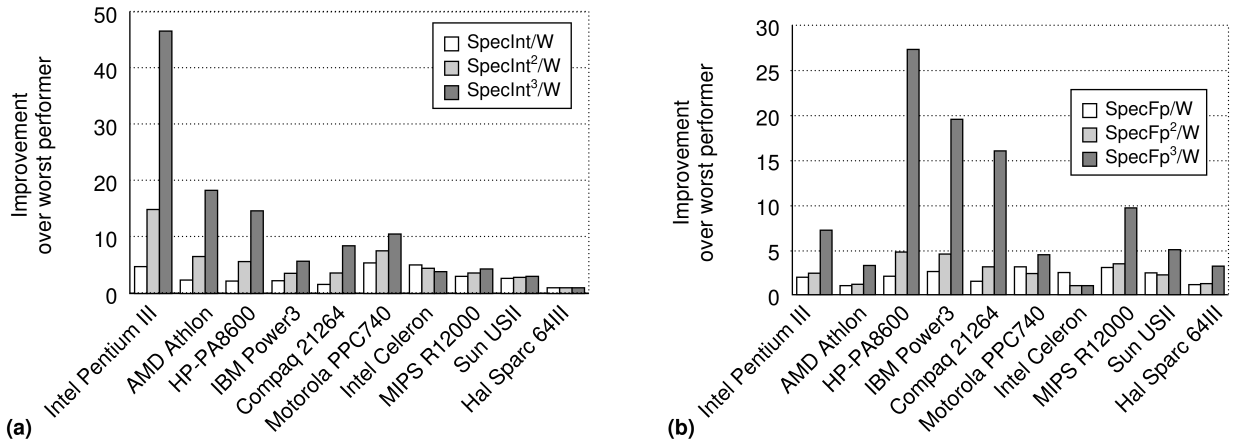

Figure 1. Performance-power efficiencies across commercial processor products. The newer processors appear on the left.

meets maximum power budget constraints) even if it operates at a much lower MIPS/W efficiency compared to one that operates at better efficiency but at a lower performance level. As such, $(\text{MIPS})^2/\text{W}$ or even $(\text{MIPS})^3/\text{W}$ may be the metric of choice at the high end.

On the other hand, at the lowest end, where battery life (or energy consumption) is the primary driver, designers may want to put an even greater weight on the power aspect than the simplest MIPS/W metric. That is, they may just be interested in minimizing the power for a given workload run, irrespective of the execution time performance, provided the latter doesn't exceed some specified upper limit.

The MIPS metric for performance and the watts value for power may refer to average or peak values, derived from the chip specifications. For example, for a 1-gigahertz (10^9 cycles/sec) processor that can complete up to 4 instructions per cycle, the theoretical peak performance is 4,000 MIPS. If the average completion rate for a given workload mix is p instructions/cycle, the average MIPS would equal 1,000 times p. However, when it comes to workload-driven evaluation and characterization of processors, metrics are often controversial. Apart from the problem of deciding on a representative set of benchmark applications, fundamental questions persist about ways to boil down performance into a single (average) rating that's meaningful in comparing a set of machines.

Since power consumption varies, depending on the program being executed, the benchmarking issue is also relevant in assigning an average power rating. In measuring power and performance together for a given program execution, we may use a fused metric such as the power-delay product (PDP) or energy-delay product (EDP).[9] In general, the PDP-based formulations are more appropriate for low-power, portable systems in which battery life is the primary index of energy efficiency. The MIPS/W metric is an inverse PDP formulation, where delay refers to average execution time per instruction. PDP, being dimensionally equal to energy, is the natural metric for such systems.

For higher end systems (workstations) the EDP-based formulation is more appropriate, since the extra delay factor ensures a greater emphasis on performance. The $(\text{MIPS})^2/\text{W}$ metric is an inverse EDP formulation. For the highest performance server-class machines, it may be appropriate to weight the delay part even more. This would point to the use of $(\text{MIPS})^3/\text{W}$, which is an inverse ED^2P formulation. Alternatively, we may use $(\text{CPI})^3 \times \text{W}$ (the cube of cycles per instruction times power) as a direct ED^2P metric applicable on a per-instruction basis.

The energy $\times$ $(\text{delay})^2$ metric, or performance^3/power formula, is analogous to the cube-root rule,[10] which follows from constant voltage-scaling arguments (see Equations A through D in the "Power-performance fundamentals" box). Clearly, to formulate a voltage-invariant power-performance characterization metric, we need to think in terms of performance^3/power.

When we are dealing with the SPEC benchmarks, we may evaluate efficiency as $(\text{SPECrating})^x$/W, or $(\text{SPEC})^x$/W for short; where exponent value x (equaling 1, 2, or 3) may depend on the class of processors being compared.

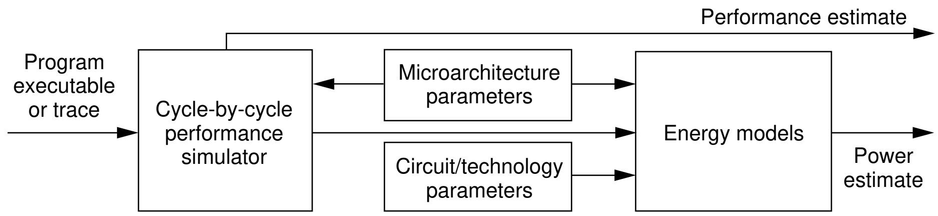

Figure 2. PowerTimer high-level block diagram.

Figure 1a,b shows the power-performance efficiency data for a range of commercial processors of approximately the same generation. (See http://www.specbench.org, the Aug. 2000 *Microprocessor Report,* and individual vendor Web pages, for example, http://www.intel.com/design/pentiumiii/datashts/245264.htm.)

In Figure 1a,b the latest available processor is plotted on the left and the oldest on the right. We used SPEC/W, SPEC^2/W, and SPEC^3/W as alternative metrics, where SPEC stands for the processor's SPEC rating. (See earlier definition principles.) For each category, the worst performer is normalized to 1, and other processor values are plotted as improvement factors over the worst performer.

The data validates our assertion that—depending on the metric of choice, and the target market (determined by workload class and/or the power/cost)—the conclusion drawn about efficiency can be quite different. For performance-optimized, high-end processors, the SPEC^3/W metric seems fairest, with the very latest Intel Pentium III and AMD Athlon offerings (at 1 GHz) at the top of the class for integer workloads. The older HP-PA 8600 (552 MHz) and IBM Power3 (450 MHz) still dominate in the floating-point class. For power-first processors targeted toward integer workloads (such as Intel's mobile Celeron at 333 MHz), SPEC/W seems the fairest.

Note that we've relied on published performance and "max power" numbers; and, because of differences in the methodologies used in quoting the maximum power ratings, the derived rankings may not be completely accurate or fair. This points to the need for standardized methods in reporting maximum and average power ratings for future processors so customers can compare power-performance efficiencies across competing products in a given market segment.

Microarchitecture-level power estimation

Figure 2 shows a block diagram of the basic procedure used in the power-performance simulation infrastructure (PowerTimer) at IBM Research. Apart from minor adaptations specific to our existing power and performance modeling methodology, at a conceptual level it resembles the recently published methods used in Princeton University's Wattch,[4] Penn State's SimplePower,[5] and George Cai's model at Intel.[8]

The core of such models is a classical trace- or execution-driven, cycle-by-cycle performance simulator. In fact, the power-performance models[4-6,8] are all built upon Burger and Austin's widely used, publicly available, parameterized SimpleScalar performance simulator.[11]

We are building a tool set around the existing, research- and production-level simulators[12,13] used in the various stages of the definition and design of high-end PowerPC processors. The nature and detail of the energy models used in conjunction with the workload-driven cycle simulator determine the key difference for power projections.

During every cycle of the simulated processor operation, the activated (or busy) microarchitecture-level units or blocks are known from the simulation state. Depending on the particular workload (and the execution snapshot within it), a fraction of the processor units and queue/buffer/bus resources are active at any given cycle.

We can use these cycle-by-cycle resource usage statistics—available easily from a trace- or execution-driven performance simulator—to estimate the unit-level activity factors. If accurate energy models exist for each modeled resource, on a given cycle and if unit i is accessed or used, we can estimate the corresponding energy consumed and add it to the net energy spent overall for that unit. So, at the end of a simulation, we can estimate the total energy spent on a unit basis as well as for the whole processor.

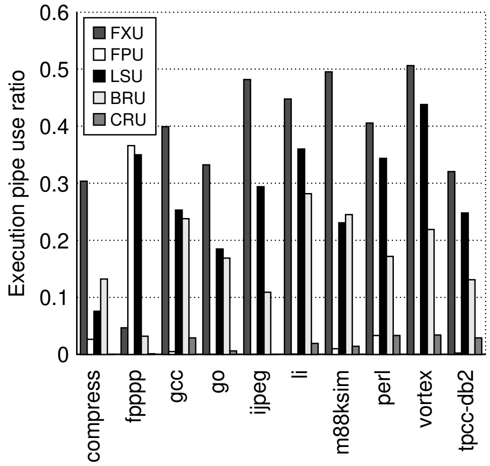

Figure 3. Unit-level usage data for a current-generation server-class superscalar processor.

Since the unit-level and overall execution cycle counts are available from the performance simulation, and since the technology implementation parameters (voltage, frequency) are known, we can estimate average and peak power dissipation profiles for the whole chip, in the context of the input workload. Of course, this kind of analysis implicitly assumes that we can use clock-gating techniques at the circuit level (for example, see Tiwari et al.[14]) to selectively turn off dynamic power consumption in inactive functional units and on-chip storage/communication resources.

Figure 3 shows the unit-level usage data for some of the key functional units, based on a detailed, cycle-accurate simulation of a processor similar in complexity and functionality to a current generation superscalar such as the Power4 processor.[15] The data plot shown is obtained by running simulations using a representative sampled trace repository. This covers a range of benchmark workloads (selections from SPEC95 and TPC-C) commonly used in such designs. The functional units tracked in this graph are the floating-point unit (FPU), integer (fixed-point) unit (FXU), branch unit (BRU), load-store unit (LSU), and "logic on condition register" unit (CRU).

This simulation data indicates the potential power savings gained by selectively "gating off" unused processor segments during the execution of a given workload mix. For example, the FPU is almost totally inactive for gcc, go, ijpeg, li, vortex, and tpcc-db2. The maximum unit use is about 50% (FXU for m88ksim and vortex).

The potential here is especially significant if we consider the distribution of power within a modern processor chip. In an aggressive superscalar processor that doesn't employ any form of clock gating, a design's clock tree, drivers, and clocked latches (pipeline registers, buffers, queues) can consume up to 70% of the total power. Even with the registers and cache structures accounted for separately, the clock-related power for a recent high-performance Intel CPU is reported to be almost half the total chip power.[14] Thus, the use of gated clocks to disable latches (pipeline stages) and entire units when possible is potentially a major source of power savings.

Depending on the exact hardware algorithm used to gate off clocks, we can depict different characteristics of power-performance optimization. In theory, for example, if the clock can be disabled for a given unit (perhaps even at the individual pipeline stage granularity) on every cycle that it's not used, we'd get the best power efficiency, without losing any or much performance. However, such idealized clock gating provides many implementation challenges for designers of high-end processors due to considerations such as

- the need to include additional on-chip decoupling capacitance to deal with large current swings ($L \times [di/dt]$ effects), where L stands for inductance and di/dt denotes current gradients over time;
- more-complicated timing analysis to deal with additional clock skew and the possibility of additional gate delays on critical timing paths, and
- more effort to test and verify the microprocessor.

Despite these additional design challenges, some commercial high-end processors have begun to implement clock gating at various levels of granularity, for example, the Intel Pentium series and Compaq Alpha 21264. The need for pervasive use of clock gating in our server-class microprocessors has thus far not outweighed the additional design complexity. For the future, however, we continue to look at methods to implement clock gating with a simplified design cost. Simpler designs may implement gated clocks in localized hot spots or in league with dynamic feedback data. For example, upon detection that the power-hungry floating-point unit has been inactive over the last, say, 1,000 cycles, the execution pipes within that unit may be gated off in stages, a pipe at a time to minimize current surges. (See Pant et al.[16])

Also for a given resource such as a register file, enabling only the segment(s) in use—depend-

Adaptive microarchitectures

Current-generation microarchitectures are usually not adaptive; that is, they use fixed-size resources and a fixed functionality across all program runs. The size choices are made to achieve best overall performance over a range of applications. However, an individual application's requirements not well matched to this particular hardware organization may exhibit poor performance. Even a single application run may exhibit enough variability during execution, resulting in uneven use of the chip resources. The result may often be low unit use (see Figure 3 in the main text), unnecessary power waste, and longer access latencies (than required) for storage resources. Albonesi et al.[1,2] in the CAP (Complexity-Adaptive Processors) project at the University of Rochester, have studied the power and performance advantages in dynamically adapting on-chip resources in tune with changing workload characteristics. For example, cache sizes can be adapted on demand, and a smaller cache size can burn less power while allowing faster access. Thus, power and performance attributes of a pair (program, machine) can both be enhanced via dynamic adaptation.

Incidentally, a static solution that attempts to exploit the fact that most of the cache references can be trapped by a much smaller "filter cache"[3] can also save power, albeit at the expense of a performance hit caused by added latency for accessing the main cache.

Currently, as part of the power-aware microarchitecture research project at IBM, we are collaborating with the Albonesi group in implementing aspects of dynamic adaptation in a prototype research processor. We've finished the high-level design (with simulation-based analysis at the circuit and microarchitecture levels) of an adaptive issue queue.[4] The dynamic workload behavior controls the queue resizing. The transistor and energy overhead of the added counter-based monitoring and decision control logic is less than 2%.

Another example of dynamic adaptation is one in which on-chip power (or temperature) and/or performance levels are monitored and fed back to control the progress of later instruction processing. Manne et al.[5] proposed the use of a mechanism called "pipeline gating" to control the degree of speculation in modern superscalars, which employ branch prediction. This mechanism allows the machine to stall specific pipeline stages when it's determined that the processor is executing instructions in an incorrectly predicted branch path. This determination is made using "confidence estimation" hardware to assess the quality of each branch prediction. Such methods can drastically reduce the count of misspeculated executions, thereby saving power. Manne[5] shows that such power reductions can be effected with minimal loss of performance. Brooks and colleagues[6] have used power dissipation history as a proxy for temperature, in determining when to throttle the processor. This approach can help put a cap on maximum power dissipation in the chip, as dictated by the workload at acceptable performance levels. It can help reduce the thermal packaging and cooling solution cost for the processor.

References

1. D.H. Albonesi, "The Inherent Energy Efficiency of Complexity-Adaptive Processors," presented at the ISCA-25 Workshop on Power-Driven Microarchitecture, 1998; http://www.ccs.rochester.edu/projects/cap/cappublications.htm.
2. R. Balasubramonian et al., "Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures," to be published in *Proc. 33*rd *Ann. Int'l Symp. Microarchitecture, (Micro-33),* IEEE Computer Society Press, Los Alamitos, Calif., 2000.
3. J. Kin, M. Gupta, and W. Mangione-Smith, "The Filter Cache: An Energy-Efficient Memory Structure," *Proc. IEEE Int'l Symp. Microarchitecture*, IEEE CS Press, 1997, pp. 184-193.
4. A. Buyuktosunoglu et al., "An Adaptive Issue Queue for Reduced Power at High Performance," IBM Research Report RC 21874, Yorktown Heights, New York, Nov. 2000.
5. S. Manne, A. Klauser, and D. Grunwald, "Pipeline Gating: Speculation Control for Energy Reduction," *Proc. 25th Ann. Int'l Symp. Computer Architecture (ISCA-25)*, 1998, IEEE CS Press, pp. 132-141.
6. D. Brooks and M. Martonosi, "Dynamic Thermal Management for High Performance Microprocessors," to be published in *Proc. Seventh Int'l Symp. High Performance Computer Architecture HPCA-7*, IEEE CS Press, 2001.

ing on the number of ports that are active at a given time—might conserve power. Alternatively, at a coarser grain an entire resource may be powered on or off, depending on demand.

Of course, even when a unit or its part is gated off, the disabled part may continue to burn a small amount of static power. To measure the overall advantage, we must carefully account for this and other overhead power consumed by the circuitry added to implement gated-clock designs. In Wattch,[4] Brooks et al. report some of the observed power-performance variations, with different mechanisms of clock-gating control. Other dynamic adaptation schemes (see the "Adaptive microarchitectures" box) of on-chip resources to conserve power are also of interest in our ongoing power-aware microarchitecture research.

Asynchronous circuit techniques such as interlocked pipeline CMOS (IPCMOS) provide the benefits of fine-grain clock gating as a natural and integrated design aspect. (See the "Low-power, high-performance circuit techniques" box.) Such techniques have the further advantage that current surges are automatically minimized because each stage's clock is gen-

Low-power, high-performance circuit techniques

Stanley Schuster, Peter Cook, Hans Jacobson, and Prabhakar Kudva

Circuit techniques that reduce power consumption while sustaining high-frequency operation are a key to future power-efficient processors. Here, we mention a couple of areas where we've made recent progress.

IPCMOS

Chip performance, power, noise, and clock synchronization are becoming formidable challenges as microprocessor performances move into the GHz regime and beyond. Interlocked Pipelined CMOS (IPCMOS),[1] a new asynchronous clocking technique, helps address these challenges.

Figure A (reproduced from Schuster et al.[1]) shows a typical block (D) interlocked with all the blocks with which it interacts. In the forward direction, dedicated Valid signals emulate the worst-case path through each driving block and thus determine when data can be latched within the typical block. In the reverse direction, Acknowledge signals indicate that data has been received by the subsequent blocks and that new data may be processed within the typical block. In this interlocked approach local clocks are generated only when there's an operation to perform.

Measured results on an experimental chip demonstrate robust operation for IPCMOS at 3.3 GHz under typical conditions and 4.5 GHz under best-case conditions in 0.18-micron, 1.5-V CMOS technology. Since the locally generated clocks for each stage are active only when the data sent to a given stage is valid, power is conserved when the logic blocks are idling. Furthermore, with the simplified clock environment it's possible to design a very simple single-stage latch that can capture and launch data simultaneously without the danger of a race.

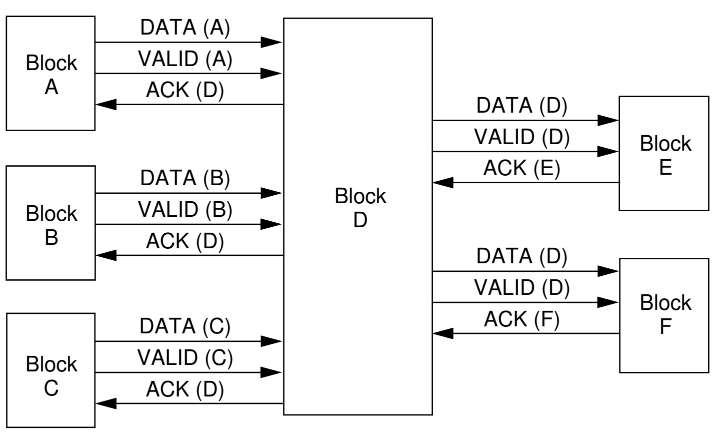

Figure A. Interlocking at block level.

The general concepts of interlocking, pipelining and asynchronous self-timing are not new and have been proposed in a variety of forms.[2,3] However the techniques used in those approaches are too slow, especially for macros that receive data from many separate logic blocks. IPCMOS achieves high-speed interlocking by combining the function of a static NOR and an input switch to perform a unique cycle-dependent AND function. Every local clock circuit has a strobe circuit that implements the asynchronous interlocking between stages. (See Schuster et al.[1] for details). A significant power reduction results when there's no operation to perform and the local clocks turn off. The clock transitions are staggered in time, reducing the peak *di/dt*—and therefore noise—compared to a conventional approach with a single global clock. The IPCMOS circuits show robust operation with large variations in power supply voltage, operating temperature, threshold voltage, and channel length.

Low-power memory structures

With ever-increasing cache sizes and out-of-order issue structures, the energy dissipated by memory circuitry is becoming a significant part of a microprocessor's total on-chip power consumption.

Random access memories (RAMs) found in caches and register files consume power mainly due to the high-capacitance bitlines on which data is read and written. To allow fast access to large memories, these bitlines are typically implemented as precharged structures with sense amplifiers. Such structures combined with the high load on the bitlines tend to consume significant power.

Techniques typically used to improve access time, such as dividing the memory into separate banks, can also be used as effective means to reduce the power consumption. Only the sub-bitline in one bank then needs to be precharged and discharged on a read, rather than the whole bitline, saving energy. Dividing often-unused bitfields (for example, Immediates) into segments and gating the wordline drive buffers with valid signals associated with these segments may in some situations further save power by avoiding having to drive parts of the high-capacitance wordline and associated bitlines.

Content addressable memories (CAMs)[4] store data and provide an associative search of their contents. Such memories are often used to implement register files with renaming support and issue queue structures that require support for operand and instruction

erated locally and asynchronously with respect to other pipeline stages. Thus, techniques such as IPCMOS have the potential of exploiting the unused execution pipe resources (depicted in Figure 3) to the fullest. They burn clocked latch power strictly on demand at every pipeline stage, without the inductive noise problem.

The usefulness of microarchitectural power estimators hinges on the accuracy of the underlying energy models. We can formulate such models for given functional unit blocks (the integer or floating-point execution data path), storage structures (cache arrays, register files, or buffer space), or communication tag matching. The high energy dissipation of CAM structures physically limits the size of such memories.[4] The high power consumption of CAMs is a result of the way the memory search is performed. The search logic for a CAM entry typically consists of a matchline to which a set of wired-XNOR functions are connected. These XNOR functions implement the bitwise comparison of the memory word and an externally supplied data word. The matchline is precharged high and is discharged if a mismatch occurs. In the use of CAMs for processor structures such as the issue queue, mismatches are predominant. Significant power is thus consumed without performing useful work. In addition, precharge-structured logic requires that the high-capacitance taglines that drive the match transistors for the externally supplied word must be cleared before the matchline can be precharged. This results in further wase of power.

AND-structured CAM match logic has been proposed to reduce power dissipation since it discharges only the matchline if there's actually a match. However, AND logic is inherently slower than wired-XNOR logic for wide memory words. New CAM structures that can offer significantly lower power while offering performance comparable to wired-XNOR CAMs have recently been studied as part of our research in this area.

References

1. S. Schuster et al., "Asynchronous Interlocked Pipelined CMOS Circuits Operating at 3.3-4.5 GHz," *Proc. 2000 IEEE Int'l Solid-State Circuits Conf. (ISSCC)*, IEEE Press, Piscataway, N.J., 2000, pp. 292-293.
2. I. Sutherland, "Micropipelines," *Comm. ACM*, Vol. 32, No. 6, ACM Press, New York, June 1989.
3. C. Mead and L. Conway, *Introduction To VLSI Systems*, Addison-Wesley Publishing Company, Reading, Mass., 1980.
4. K.J. Schultz, "Content-Addressable Memory Core Cells: A Survey," *Integration, the VLSI J.,* Vol. 23, 1997, pp. 171-188.

bus structures (instruction dispatch bus or result bus) using either

- circuit-level or RTL simulation of the corresponding structures with circuit and technology parameters germane to the particular design, or
- analytical models or equations that formulate the energy characteristics in terms of the design parameters of a particular unit or block.

Prior work[4,6] discusses the formulation of analytical capacitance equations that describe microarchitecture building blocks. Brooks et al.,[4] for example, provides categories of structures modeled for power: 1) array structures including caches, register files, and branch prediction tables; 2) content-addressable memory (CAM) structures including those used in issue queue logic, translation look-aside buffers (TLBs); 3) combinational logic blocks and wires including the various functional units and buses; and 4) clocking structures including the corresponding buffers, wires, and capacitive loads.

Brooks et al. validated the methodology for forming energy models by comparing the energy models for array structures with schematic and layout-level circuit extractions for array structures derived from a commercial Intel microprocessor. They were found to be accurate within 10%. This is similar to the accuracy reported by analytical cache delay models that use a similar methodology. On the other hand, since the core simulation occurred at the microarchitectural abstraction, the speed was 1,000 times faster when compared to existing layout-level power estimation tools.

If detailed power and area measurements for a given chip are possible, we could build reasonably accurate energy models based on power density profiles across the various units. We can use such models to project expected power behavior for follow-on processor design points by using standard technology scaling factors. These power-density-based energy models have been used in conjunction with trace-driven simulators for power analysis and trade-off studies in some Intel processors.[8] We've developed various types of energy models: 1) power-density-based models for design lines with available power and area measurements from implemented chips; and 2) analytical models in terms of microarchitecture-level design parameters such as issue width, number of physical registers, pipeline stage lengths, misprediction penalties, cache geometry parameters, queue/buffer lengths, and so on.

We formulated the analytical energy behav-

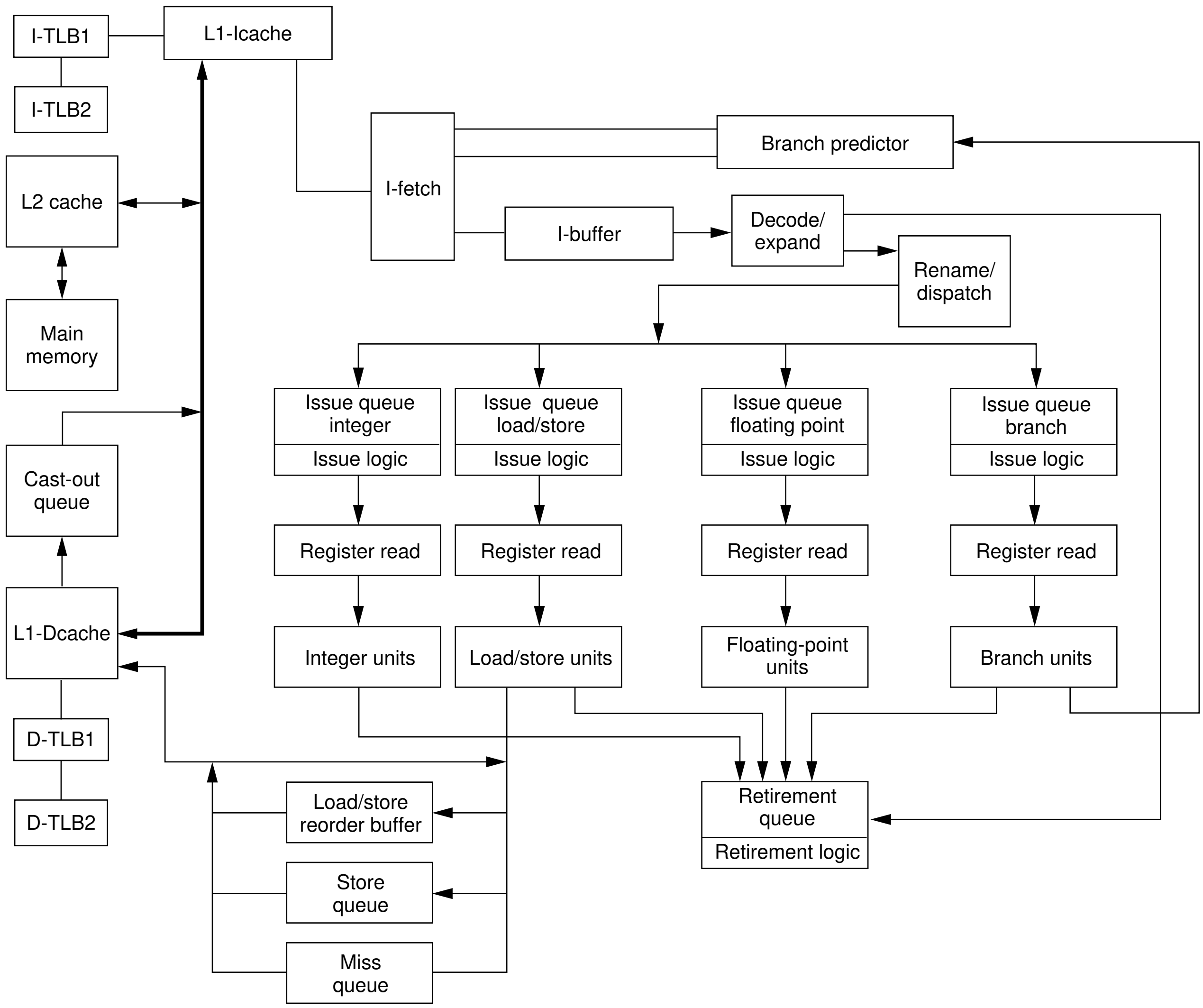

Figure 4. Processor organization modeled by the Turandot simulator.

iors based on simple area determinants and validated the constants with circuit-level simulation experiments using representative test vectors. Data-dependent transition variations aren't considered in our initial energy model formulations, but we will factor them into the next simulator version.

We consider validation to be an integrated part of the methodology. Our focus is on building models that rate highly for relative accuracy. That is, our goal is to enable designers to explore early-stage microarchitecture options that are inherently superior from a power-efficiency standpoint. Absolute accuracy of early-stage power projection for a future processor isn't as important, so long as tight upper bounds are established early.

Power-performance trade-off result

In our simulation toolkit, we assumed a high-level description of the processor model. We obtained results using our current version of PowerTimer, which works with presilicon performance and energy models developed for future, high-end PowerPC processors.

Base microarchitecture model

Figure 4 shows the high-level organization of our modeled processor. The model details make it equivalent in complexity to a modern, out-of-order, high-end microprocessor (for example, the Power4 processor[15]). Here, we assume a generic, parameterized, out-of-order superscalar processor model adopted in a research simulator called Turandot.[12,13] (Figure 4 is essentially reproduced from Moudgill et al.[13])

As described in the Turandot validation paper, this research simulator was calibrated against a pre-RTL, detailed, latch-accurate processor model (referred to here as the R-model[13]). The R-model served as a validation reference in a real processor development project. The R-model is a custom simulator written in C++ (with mixed VHDL "interconnect code"). There is a virtual one-to-one correspondence of signal names between the R-model and the actual VHDL (RTL) model. However, the R-model is about two orders of magnitude faster than the RTL model and considerably more flexible. Many microarchitecture parameters can be varied, albeit within restricted ranges. Turandot, on the other hand is a classical trace/execution-driven simulator, written in C, which is one to two orders of magnitude faster than the R-model. It supports a much greater number and range of parameter values (see Moudgill et al.[13] for details).

Here, we report power-performance results using the same version of R-model as that used in Moudgill et al.[13] That is, we decided to use our developed energy models first in conjunction with the R-model to ensure accurate measurement of the resource use statistics within the machine. To circumvent the simulator speed limitations, we used a parallel workstation cluster (farm). We also post-processed the performance simulation output and fed the average resource use statistics to the energy models to get the average power numbers. Looking up the energy models on every cycle during the actual simulation run would have slowed the R-model execution even further. While it would've been possible to get instantaneous, cycle-by-cycle energy consumption profiles through such a method, it wouldn't have changed the average power numbers for entire program runs.

Having used the detailed, latch-accurate reference model for our initial energy characterization, we could look at the unit- and queue-level power numbers in detail to understand, test, and refine the various energy models. Currently, we have reverted to using an energy-model-enabled Turandot model, for fast CPI versus power trade-off studies with full benchmark traces. Turandot lets us experiment with a wider range and combination of machine parameters.

Our experimental results are based on the SPEC95 benchmark suite and a commercial TPC-C trace. All workload traces were collected on a PowerPC machine. We generated SPEC95 traces using the Aria tracing facility within the MET toolkit.[13] We created the SPEC trace repository by using the full reference input set, however, we used sampling to reduce the total trace length to 100 million instructions per benchmark program. In finalizing the choice of exact sampling parameters, the performance team also compared the sampled traces against the full traces in a systematic validation study. The TPC-C trace is a contiguous (unsampled) trace collected and validated by the processor performance team at IBM Austin and is about 180 million instructions long.

Data cache size and effect of scaling techniques

We evaluated the relationship between performance, power, and level 1 (L1) data cache size. We varied the cache size by increasing the number of cache lines per set while leaving the line size and cache associativity constant. Figure 5a,b (next page) shows the results of increasing the cache size from the baseline architecture. This is represented by the points corresponding to the current design size; these are labeled 1x on the x-axis in those graphs.

Figure 5a (next page) illustrates the variation of relative CPI with the L1 data cache size (CPI and related power-performance metric values in Figures 5 and 6 are relative to the baseline machine values, that is, points 1x on the x-axes.) Figure 5b shows the variation when considering the metric $(\text{CPI})^3 \times$ power. From Figure 5b, it's clear that the small CPI benefits of increasing the data cache are outweighed by the increases in power dissipation due to larger caches.

Figure 5b shows the $[(\text{CPI})^3 \times \text{power}]$ metric with two different scaling techniques. The first technique assumes that power scales linearly with the cache size. As the number of lines doubles, the power of the cache also doubles. The second scaling technique is based on data from a study[17] of energy optimizations within multilevel cache architectures: cache power dissipation for conventional caches with sizes ranging from 1 Kbyte to 64 Kbytes. In the second scaling technique, which we call non-lin, the cache power is scaled with the ratios presented in the same study. The increase in cache power by dou-

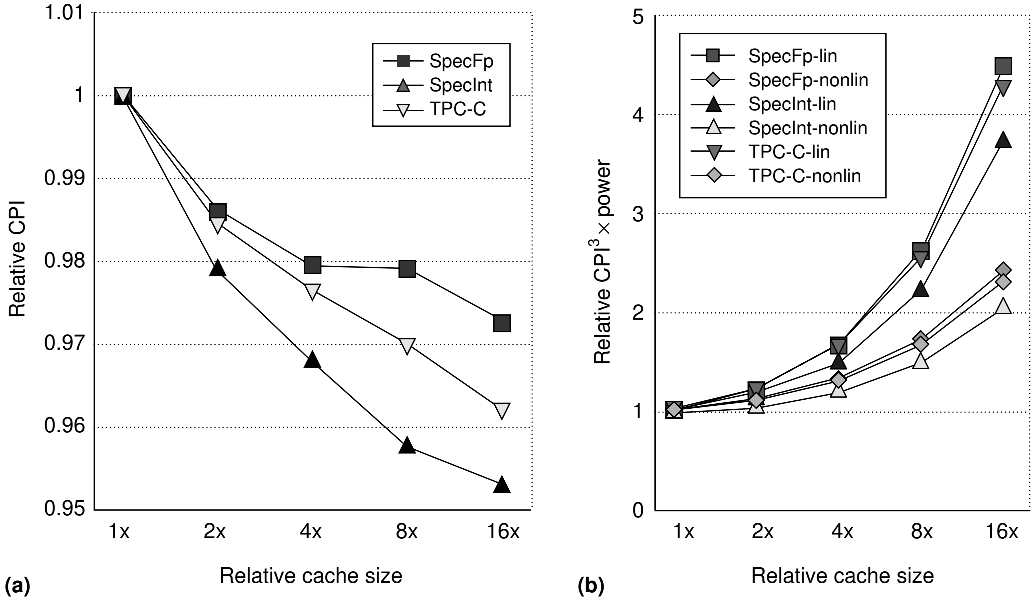

Figure 5. Variation of performance (a) and power-performance (b) with cache size.

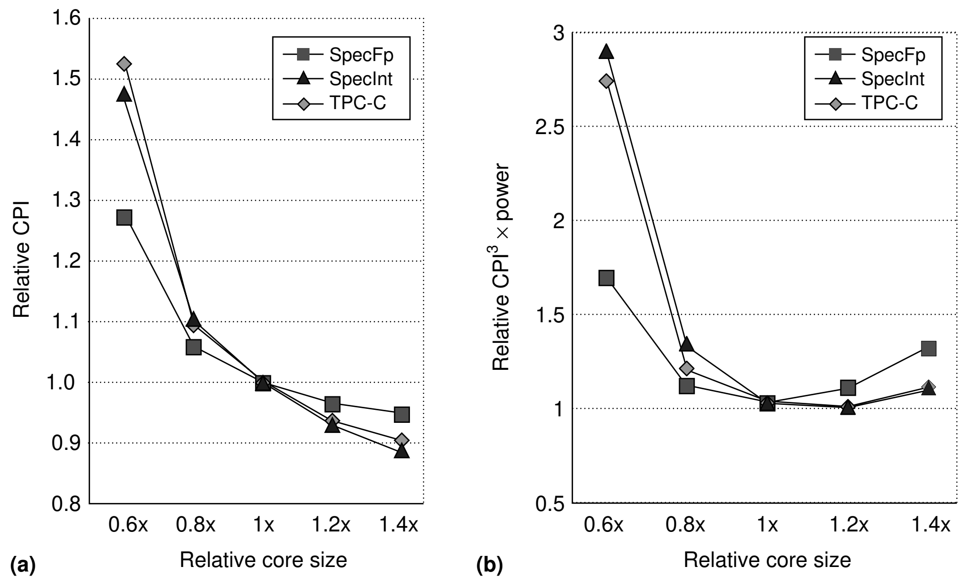

Figure 6. Variation of performance (a) and power-performance (b) with core size (ganged parameters).

bling cache size using this technique is roughly 1.46x, as opposed to the 2x with the simple linear scaling method. Obviously, the choice of scaling technique can greatly affect the results. However, with either scaling choice, conventional cache organizations (that is, cache designs without partial array shutdowns to conserve power) won't scale in a power-efficient manner. Note that the curves shown in Figure 5b assume a given, fixed circuit/technology generation; they show the effect of adding more cache to an existing design.

Ganged sizing

Out-of-order superscalar processors of the class considered rely on queues and buffers to efficiently decouple instruction execution and increase performance. The pipeline depth and the resource size required to support decoupled execution combine to determine the machine performance. Because of this decoupled execution style, increasing the size of one resource without regard to other machine resources may quickly create a performance bottleneck. Thus, we considered the effects of varying multiple parameters rather than just a single parameter in our modeled processor.

Figure 6a,b shows the effects of varying all of the resource sizes within the processor core. These include issue queues, rename registers, branch predictor tables, memory disambiguation hardware, and the completion table. For the buffers and queues, the number of entries in each resource is scaled by the values specified in the charts (0.6x, 0.8x, 1.2x, and 1.4x). For the instruction cache, data cache, and branch prediction tables, the sizes of the structures are doubled or halved at each data point.

From Figure 6a, we can see that performance increased by 5.5% for SPECfp, 9.6% for SPECint, and 11.2% for TPC-C as the size of the resources within the core is increased by 40% (except for the caches that are 4x larger). The configuration had a power dissipation of 52% to 55% higher than the baseline core. Figure 6b shows that the most power-efficient core microarchitecture is somewhere between the 1x and 1.2x cores.

Power-efficient microarchitecture design ideas and trends

With the means to evaluate power efficiency during early-stage microarchitecture evaluations, we can propose and characterize processor organizations that are inherently energy efficient. Also, if we use the right efficiency metric in the right context (see earlier discussion), we can compare alternate points and rank them using a single power-performance efficiency number. Here, we first examine the power-performance efficiency trend exhibited by the current regime of superscalar processors. The "SMT/CMP differences and energy efficiency issues" box uses a simple loop-oriented example to illustrate the basic performance and power characteristics of the current superscalar regime. It also shows how the follow-on paradigms such as simultaneous multithreading (SMT) may help correct the decline in power-performance efficiency measures.

SMT/CMP differences and energy-efficiency issues

Consider the floating-point loop kernel shown in Table A. The loop body consists of seven instructions labeled A through G. The final instruction is a conditional branch that causes control to loop back to the top of the loop body. Labels T through Z are used to tag the corresponding instructions for a parallel thread when considering SMT and CMP. The lfdu/stfdu instructions are load/store instructions with update where the base address register (say, r1, r2, or r3) is updated to hold the newly computed address.

We assumed that the base machine (refer to Figure 4 in the main text) is a four-wide superscalar processor with two load-store units supporting two floating-point pipes. The data cache has two load ports and a separate store port. The two load-store units (LSU0 and LSU1) are fed by a single issue queue LSQ; similarly, the two floating-point units (FPU0 and FPU1) are fed by a single issue queue FPQ. In the context of the loop just shown, we essentially focus on the LSU-FPU subengine of the whole processor.

Assume that the following high-level parameters (latency and bandwidth) characterizing the base superscalar machine:

- Instruction fetch bandwidth fetch_bw of two times *W* is eight instructions per cycle.
- Dispatch/decode/rename bandwidth equals W, which is four instructions/cycle; dispatch stalls beyond the first branch scanned in the instruction fetch buffer.
- Issue bandwidth from LSQ (reservation station) lsu_bw of *W*/2 is two instructions/cycle.
- Issue bandwidth from FPQ fpu_bw of *W*/2 is two instructions/cycle.
- Completion bandwidth compl_bw of *W* is four instructions/cycle.
- Back-to-back dependent floating-point operation issue delay fp_delay is one cycle.

continued on p. 38

Table A. Example loop test case.

Parallel instruction tags	Instruction	Load/store instruction	Description
T	A	fadd fp3, fp1, fp0	; add FPR fp1 and fp0; store into target register fp3
U	B	lfdu fp5, 8(r1)	; load FPR fp5 from memory address: 8 + contents of GPR r1
V	C	lfdu fp4, 8(r3)	; load FPR fp4 from memory address: 8 + contents of GPR r3
W	D	fadd fp4, fp5, fp4	; add FPR fp5 and fp4; store into target register fp4
X	E	fadd fp1, fp4, fp3	; add FPR fp4 and fp3; store into target register fp1
Y	F	stfdu fp1, 8(r2)	; store FPR fp1 to memory address: 8 + contents of GPR r2
Z	G	bc loop_top	; branch back conditionally to top of loop body

Single-core, wide-issue, superscalar processor chip paradigm

One school of thought envisages a continued progression along the path of wider, aggressively speculative superscalar paradigms. (See Patt et al.[18] for an example.) Researchers continue to innovate in efforts to exploit

continued from p. 37

- The best-case load latency from fetch to write back is five cycles
- The best-case store latency, from fetch to writing in the pending store queue is four cycles. (A store is eligible to complete the cycle after the address-data pair is valid in the store queue.)
- The best-case floating-point operation latency from fetch to write back is seven cycles (when the FPQ issue queue is bypassed because it's empty).

Load and floating-point operations are eligible for completion (retirement) the cycle after write back to rename buffers. For simplicity of analysis assume that the processor uses in-order issue from issue queues LSQ and FPQ.

In our simulation model, superscalar width *W* is a ganged parameter, defined as follows:

- *W* = (fetch_bw/2) = disp_bw = compl_bw.
- The number of LSU units, ls_units, FPU units, fp_units, data cache load ports, l_ports, and data cache store ports, the term s_ports, vary as follows as *W* is changed: ls_units = fp_units = l_ports = max [floor(*W*/2), 1]. s_ports = max [floor(l_ports/2), 1].

We assumed a simple analytical energy model in which the power consumed is a function of parameters *W*, ls_units, fp_units, l_ports, and s_ports. In particular, the power (*PW*) in pseudowatts is computed as

$$PW = W^{0.5} + \text{ls\_units} + \text{fp\_units} + \text{l\_ports} + \text{s\_ports}.$$

Figure B shows the performance and performance/power ratio variation with superscalar width. The MIPS values are computed from the CPI values, assuming a 1-GHz clock frequency.

The graph in Figure B1 shows that a maximum issue width of four could be used to achieve the best (idealized) CPI performance. However as shown in Figure B2, from a power-performance efficiency viewpoint (measured as a performance over power ratio in this example), the best-case design is achieved for a width of three. Depending on the sophistication and accuracy of the energy model (that is, how power varies with microarchitectural complexity) and the exact choice of the power-performance efficiency metric, the maximum value point in the curve in Figure B2 will change. However beyond a certain superscalar width, the power-performance efficiency will diminish continuously. Fundamentally, this is due to the single-thread ILP limit of the loop trace.

Note that the resource sizes (number of rename buffers, reorder buffer size, sizes of various other queues, caches, and so on) are assumed to be large enough that they're effectively infinite for the purposes of our running example. Some of the actual sizes assumed for the base case (*W* = 4) are

- completion (reorder) buffer size cbuf_size of 32,
- load-store queue size lsq_size of 6,
- floating-point queue size fpq_size of 8, and
- pending store queue size psq_size of 16.

As indicated in the main text, the microarchitectural trends beyond the current superscalar regime are effectively targeted toward the goal of extending the processing efficiency factors. That is, the complexity growth must ideally scale at a slower rate than performance growth. Power consumption is one index of complexity; it also determines packaging and cooling costs. (Verification cost and effort is another important index.) In that sense, striving to ensure that the power-performance efficiency metric of choice is a nondecreasing function of time is a way of achieving complexity-effective designs (see reference 3 in the main text reference list).

Let's examine one of the paradigms, SMT, discussed in the main text to understand how this may affect our notion of power-performance efficiency. Table B shows a snapshot of the steady-state, cycle-by-cycle, resource use profile for our example loop trace executing on the baseline, with a four-wide superscalar machine (with a fp_delay of 1). Note the completion (reorder) buffer, CBUF, LSQ, LSU0, LSU1, FPQ, FPU0, FPU1, the cache access ports (C0, C1: read and C3: write), and the pending store queue. The use ratio for a queue, port, or execution pipe is measured as the number of valid entries/stages divided by the total for that resource. Recall from Figure B that the steady-state CPI for this case is 0.28.

The data in Table B shows that the steady-state use of the CBUF is 53%, the LSU0/FPU0 pipe is fully used (100%), and the average use of the

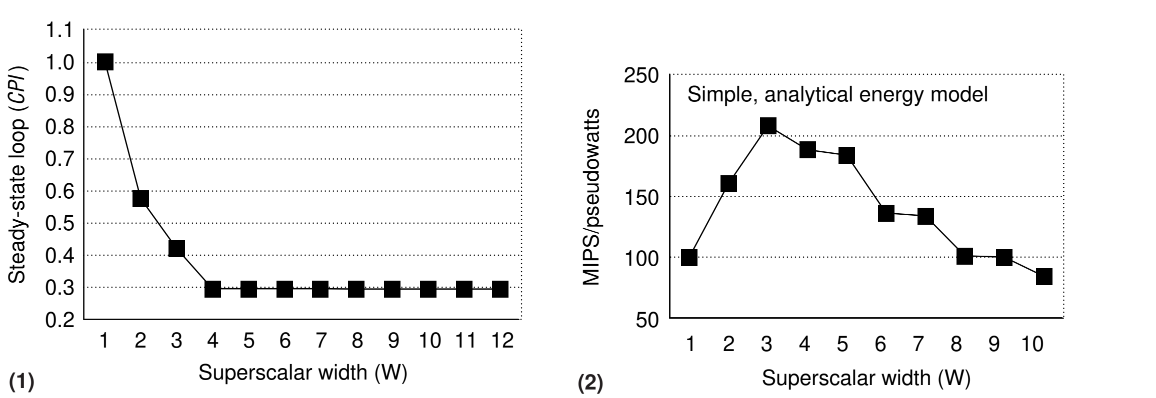

Figure B. Loop performance (1) and performance/power variation (2) with issue width.

Table B. Steady-state, cycle-by-cycle resource use profile (*W* = 4 superscalar). C: cache access port, CBUF: completion (reorder) buffer, PSQ: pending store queue

Cycle	CBUF	LSQ	LSU0	LSU1	FPQ	FPU0	FPU1	C0	C1	C3	PSQ
n	0.53	0	1	0.5	0	1	0.6	1	1	0	0.13
n+1	0.53	0	1	0.5	0	1	0.4	0	0	1	0.13
n+2	0.53	0	1	0.5	0	1	0.6	1	1	0	0.13
n+3	0.53	0	1	0.5	0	1	0.4	0	0	1	0.13
n+4	0.53	0	1	0.5	0	1	0.6	1	1	0	0.13
n+5	0.53	0	1	0.5	0	1	0.4	0	0	1	0.13
n+6	0.53	0	1	0.5	0	1	0.6	1	1	0	0.13
n+7	0.53	0	1	0.5	0	1	0.4	0	0	1	0.13
n+8	0.53	0	1	0.5	0	1	0.6	1	1	0	0.13
n+9	0.53	0	1	0.5	0	1	0.4	0	0	1	0.13

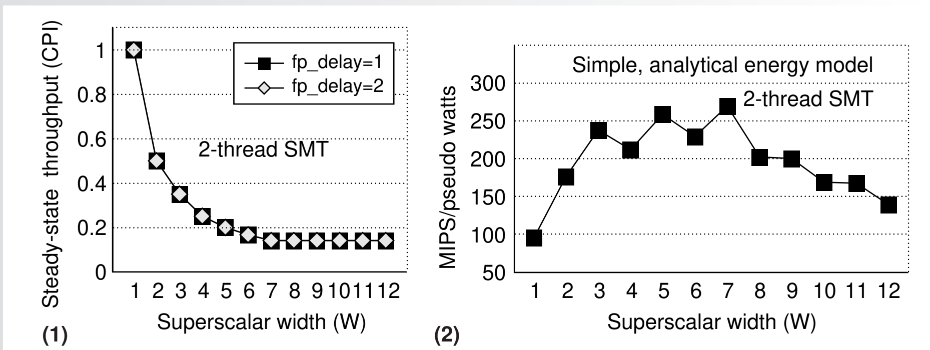

Figure C. Performance (1) and power-performance variation (2) with *W* for a two-thread SMT.

LSU1/FPU1 pipe is 50% [(0.6 + 0.4)/2 = 0.5 for FPU1]. The LSQ and FPQ are totally unused (0%) because of the queue bypass facility and the relative lack of dependence stalls. The average read/write port use of the data cache is roughly 33%; and the pending store queue residency is a constant 2 out of 16 (= 0.125, or roughly 13%). Due to fundamental ILP limits, the CPI won't decrease beyond W = 4, while queue/buffer sizes and added pipe use factors will go on decreasing as *W* is increased. Clearly, power-performance efficiency will be on a downward trend (see Figure B). (Of course, here we assume maximum processor power numbers, without clock gating or dynamic adaptation to bring down power).

With SMT, assume that we can fetch from two threads (simultaneously, if the instruction cache has two ports, or in alternate cycles if the instruction cache has one port). Suppose two copies of the same loop program (see Table A) are executing as two different threads. So, thread-1 instructions A-B-C-D-E-F-G and thread-2 instructions T-U-V-W-X-Y-Z are simultaneously available for dispatch and subsequent execution on the machine. This facility allows the use factors, and the net throughput performance to increase, without a significant increase in the maximum clocked power. This is because the width isn't increased, but the execution and resource stages or slots can be filled up simultaneously from both threads.

The added complexity in the front end—of maintaining two program counters (fetch streams) and the global register space increase alluded to before—adds to the power a bit. On the other hand, the core execution complexity can be relaxed somewhat without a performance hit. For example, we can increase the fp_delay parameter to reduce core complexity, without performance degradation.

Figure C shows the expected performance and power-performance variation with *W* for the two-thread SMT processor. The power model assumed for the SMT machine is the same as that of the underlying superscalar, except that a fixed fraction of the net power is added to account for the SMT overhead. (Assume the fraction added is linear in the number of threads in an *n*-thread SMT.)

Figure C2 shows that under the assumed model, the power-performance efficiency scales better with *W*, compared with the base superscalar (Figure B1).

In a multiscalar-type CMP machine, different iterations of a loop program could be initiated as separate tasks or threads on different core processors on the same chip. Thus in a two-way multiscalar CMP, a global task sequencer would issue threads A-B-C-D-E-F-G and T-U-V-W-X-Y-Z derived from the same user program in sequence to two cores.

Register values set in one task are forwarded in sequence to dependent instructions in subsequent tasks. For example, the register value in fp1 set by instruction E in task 1 must be communicated to instruction T in task 2. So instruction T must stall in the second processor until the value communication has occurred from task 1. Execution on each processor proceeds speculatively, assuming the absence of load-store address conflicts between tasks. Dynamic memory address disambiguation hardware is required to detect violations and restart task executions as needed.

If the performance can be shown to scale well with the number of tasks, and if each processor is designed as a limited-issue, limited-speculation (low-complexity) core, we can achieve better overall scalability of power-performance efficiency.

single-thread instruction-level parallelism (ILP). Value prediction advances (see Lipasti et al.[18]) promise to break the limits imposed by data dependencies. Trace processors (Smith et al.[18]) ease the fetch bandwidth bottleneck, which can otherwise impede scalability.

Nonetheless, increasing the superscalar width beyond a certain limit seems to yield diminishing gains in performance, while continuously reducing the performance-power efficiency metric (for example, SPEC^3/W). Thus, studies show that for the superscalar paradigm going for wider issue, speculative hardware in a core processor ceases to be viable beyond a certain complexity point. (See an illustrative example in the "SMT/CMP differences and energy efficiency issues" box). Such a point certainly seems to be upon us in the processor design industry. In addition to power issues, more complicated, bigger cores add to the verification and yield problems, which all add up to higher cost and delays in time-to-market goals.

Advanced methods in low-power circuit techniques and adaptive microarchitectures can help us extend the superscalar paradigm for a few more years. (See the earlier "Low-power, high-performance circuit techniques" and "Adaptive microarchitectures" boxes.) Other derivative designs, such as those for multicluster processors and SMT, can be more definitive paradigm shifts.

Multicluster superscalar processors

Zyuban[6] studied the class of multicluster superscalar processors as a means of extending the power-efficient growth of the basic superscalar paradigm. Here, we provide a very brief overview of the main results and conclusions of Zyuban's work (see also Zyuban and Kogge[7]). We'll use elements of this work in the modeling and design work in progress within our power-aware microarchitecture project.

As we've implied, the desire to extract more and more ILP using the superscalar approach requires the growth of most of the centralized structures. Among them are instruction fetch logic, register rename logic, register file, instruction issue window with wakeup and selection logic, data-forwarding mechanisms, and resources for disambiguating memory references. Zyuban analyzed circuit-level implementation of these structures. His analysis shows that in most cases, the energy dissipated per instruction grows in a superlinear fashion. None of the known circuit techniques solves this energy growth problem. Given that the IPC performance grows sublinearly with issue width (with asymptotic saturation), it's clear why the classical superscalar path will lead to increasingly power-inefficient designs.

One way to address the energy growth problem at the microarchitectural level is to replace a classical superscalar CPU with a set of clusters, so that all key energy consumers are split among clusters. Then, instead of accessing centralized structures in the traditional superscalar design, instructions scheduled to an individual cluster would access local structures most of the time. The primary advantage of accessing a collection of local structures instead of a centralized one is that the number of ports and entries in each local structure is much smaller. This reduces access latency and lowers power.

Current high-performance processors (for example, the Compaq Alpha 21264 and IBM Power4) certainly have elements of multiclustering, especially in terms of duplicated register files and distributed issue queues. Zyuban proposed and modeled a specific multicluster organization. Simulation results showed that to compensate for the intercluster communication and to improve power-performance efficiency significantly, each cluster must be a powerful out-of-order superscalar machine by itself. This simulation-based study determined the optimal number of clusters and their configurations, for a specified efficiency metric (the EDP).

The multicluster organization yields IPC performance inferior to a classical superscalar with centralized resources (assuming equal net issue width and total resource sizes). The latency overhead of intercluster communication is the main reason behind the IPC shortfall. Another reason is that centralized resources are always better used than distributed ones. However, Zyuban's simulation data shows that the multicluster organization is potentially more energy-efficient for wide-issue processors with an advantage that grows with the issue width.[7] Given the same power budget, the multicluster organization allows configurations that can deliver higher performance than the best configurations with the centralized design.

VLIW or EPIC microarchitectures

The very long instruction word (VLIW) paradigm has been the conceptual basis for Intel's future thrust using the Itanium (IA-64) processors. The promise here is (or was) that much of the hardware complexity could be moved to the software (compiler). The latter would use global program analysis to look for available parallelism and present explicitly parallel instruction computing (EPIC) execution packets to the hardware. The hardware could be relatively simple in that it would avoid much of the dynamic unraveling of parallelism necessary in a modern superscalar processor. For inferences regarding performance, power, and efficiency metrics for this microarchitecture class, see the September-October 2000 issue of *IEEE Micro,* which focuses on the Itanium.

Chip multiprocessing

Server product groups such as IBM's PowerPC division have relied on chip multiprocessing as the future scalable paradigm. The Power4 design[15] is the first example of this trend. Its promise is to build multiple processor cores on the same die or package to deliver scalable solutions at the system level.

Multiple processor cores on a single die can operate in various ways to yield scalable performance in a complexity- and power-efficient manner. In addition to shared-memory chip multiprocessing (see Hammond et al.[18]), we may consider building a multiscalar processor,[19] which can spawn speculative tasks (derived from a sequential binary) to execute concurrently on multiple cores. Intertask communication can occur via register value forwarding or through shared memory. Dynamic memory disambiguation hardware is required to detect memory-ordering violations. On detecting such a violation, the offending task(s) must be squashed and restarted. Multiscalar-type paradigms promise scalable, power-efficient designs, provided the compiler-aided task-partitioning and instruction scheduling algorithms can be improved effectively.

Multithreading

Various flavors of multithreading have been proposed and implemented in the past as a means to go beyond single-thread ILP limits. One recent paradigm that promises to provide a significant boost in performance with a small increase in hardware complexity is simultaneous multithreading.[20] In SMT, the processor core shares its execution-time resources among several simultaneously executing threads (programs). Each cycle, instructions can be fetched from one or more independent threads and injected into the issue and execution slots. Because the issue and execution resources can be filled by instructions from multiple independent threads in the same cycle, the per-cycle uses of the processor resources can be significantly improved, leading to much greater processor throughput. Compaq has announced that its future 21x64 designs will embrace the SMT paradigm.

For occasions when only a single thread is executing on an SMT processor, the processor behaves almost like a traditional superscalar machine. Thus the same power reduction techniques are likely to be applicable. When an SMT processor is simultaneously executing multiple threads, however, the per-cycle use of the processor resources should noticeably increase, offering fewer opportunities for power reduction via such traditional techniques as clock gating. An SMT processor—when designed specifically to execute multiple threads in a power-aware manner—provides additional options for power-aware design.

The SMT processor generally provides a boost in overall throughput performance, and this alone will improve the power-performance ratio for a set of threads, especially in a context (such as server processors) of greater emphasis on the overall (throughput) performance than on low power. Furthermore, a processor specifically designed with SMT in mind can provide even greater power performance efficiency gains.

Because the SMT processor takes advantage of multiple execution threads, it could be designed to employ far less aggressive speculation in each thread. By relying on instructions from a different thread to provide increased resource use when speculation would have been used in a single-threaded architecture (and accepting higher throughput over the multiple threads rather than single-thread latency performance), the SMT processor can spend more of its effort on nonspeculative instructions. This inherently implies a greater power efficiency per thread;

Compiler support

Manish Gupta

Figure D shows a high-level organization of the IBM XL family of compilers. The front ends for different languages generate code in a common intermediate representation called W-code. The Toronto Portable Optimizer (TPO) is a W-code-to-W-code transformer. It performs classical data flow optimizations (such as global constant propagation, dead-code elimination, as well as loop and data transformations) to improve data locality (such as loop fusion, array contraction, loop tiling, and unimodular loop transformations.)[1,2] The TPO also performs parallelization of codes based on both automatic detection of parallelism and use of OpenMP directives for parallel loops and sections. The back end for each architecture performs machine-specific code generation and optimizations.

The loop transformation capabilities of the TPO can be used on array-intensive codes (such as SPECfp-type codes and multimedia codes[2]) to reduce the number of memory accesses and, hence, reduce the energy consumption. Recent research (such as the work by Kandemir et al.[3]) indicates that the best loop configuration for performance need not be the best from an energy point of view. Among the optimizations applied by the back end, register allocation and instruction scheduling are especially important for achieving power reduction.

Power-aware register allocation involves not only reducing the number of memory accesses but also attempting to reduce the amount of switching activity by exploiting any known relationships between variables that are likely to have similar values. We can use power-aware instruction scheduling to increase opportunities for clock gating of CPU components, by trying to group instructions with similar resource use, as long as the impact on performance is kept to a reasonable limit.

References

1. U. Banerjee, *Loop Transformations for Restructuring Compilers*, Kluwer Academic Publishers, Boston, Mass., 1997.
2. C. Kulkarni et al., "Interaction Between Data-Parallel Compilation and Data Transfer and Storage Cost for Multimedia Applications," *Proc. EuroPar 99,* Lecture Notes in Computer Science, Vol. 1685, Springer-Verlag, Sep. 1999, pp. 668-676.
3. M. Kandemir et al., "Experimental Evaluation of Energy Behavior of Iteration Space Tiling," to appear in *Proc. 13th Workshop on Languages and Compilers for Parallel Computing*, Lecture Notes on Computer Science, Springer-Verlag, 2001, pp.141-156.

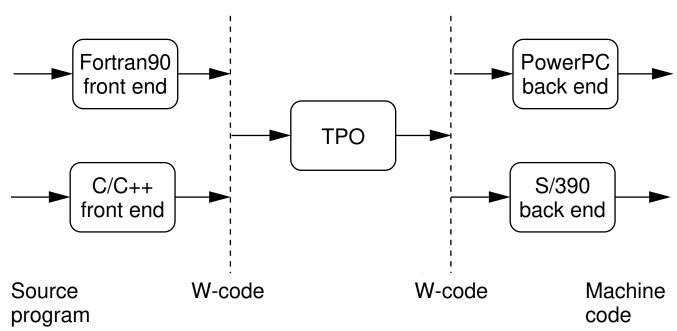

Figure D. Architecture of IBM XL compilers. (TPO: Toronto Portable Optimizer)

that is, the power expended in the execution of useful instructions weighs better against misspeculated instructions on a per-thread basis. This also implies a somewhat simpler branch unit design (for example, fewer resources devoted to branch speculation) that can further aid in the development by reducing design complexity and verification effort.

SMT implementations require an overhead in terms of additional hardware to maintain a multiple-thread state. This increase in hardware implies some increase in the processor's per-cycle power requirements. SMT designers must determine whether this increase in processor resources (and thus per-cycle power) can be well balanced by the reduction of other resources (less speculation) and the increase in performance attained across the multiple threads.

Compiler support

Compilers can assist the microarchitecture in reducing the power consumption of programs. As explained in the "Compiler support" box, a number of compiler techniques developed for performance-oriented optimizations can be exploited (usually with minor modifications) to achieve power reduction. Reducing the number of memory accesses, reducing the amount of switching activity in the CPU, and increasing opportunities for clock gating will help here.

Energy-efficient cache architectures

We've seen several proposals for power-efficient solutions to the cache hierarchy design.[2-5,17] The simplest of these is the filter cache idea first proposed by Kin et al. (referenced in the "Adaptive microarchitectures" box). More recently, Albonesi (see same box) investigated the power-performance trade-offs that can be exploited by dynamically changing the cache sizes and clock frequencies during program execution.

Our tutorial-style contribution may help enlighten the issues and trends in this new area of power-aware microarchitectures. Depending on whether the design priority is high performance or low power, and what form of design scaling is used for energy reduction, designers should employ different metrics to compare a given class of competing processors.

As explained by Borkar,[21] the static (leakage) device loss will become a much more significant component of total power dissipation in future technologies. This will impose many more difficult challenges in identifying energy-saving opportunities in future designs.

The need for robust power-performance modeling at the microarchitecture level will continue to grow with tomorrow's workload and performance requirements. Such models will enable designers to make the right choices in defining the future generation of energy-efficient microprocessors. MICRO

References

1. M.K. Gowan, L.L. Biro, and D.B. Jackson, "Power Considerations in the Design of the Alpha 21264 Microprocessor," *Proc. IEEE/ACM Design Automation Conf.*, 1998, ACM, New York, pp. 726-731.
2. 1998 ISCA Workshop on Power-Driven Microarchitecture papers; http://www.cs.colorado.edu/~grunwald/LowPowerWorkshop.
3. 2000 ISCA Workshop Complexity-Effective Design papers; http://www.ece.rochester.edu/~albonesi/ced00.html.
4. D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA),* IEEE Computer Society Press, Los Alamitos, Calif., 2000, pp. 83-94.
5. N. Vijaykrishnan et al., "Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower," *Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA)*, 2000, pp. 95-106.
6. V. Zyuban, "Inherently Lower-Power High Performance Super Scalar Architectures," PhD thesis, Dept. of Computer Science and Engineering, Univ. of Notre Dame, Ind., 2000.
7. V. Zyuban and P. Kogge, "Optimization of High-Performance Superscalar Architectures for Energy Efficiency," *Proc. IEEE Symp. Low Power Electronics and Design*, ACM, New York, 2000.
8. Cool Chips Tutorial talks presented at MICRO-32, the 32nd Ann. IEEE Int'l Symp. Microarchitecture, Dec. 1999, http://www.eecs.umich.edu/~tnm/cool.html.
9. R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *IEEE J. Solid-State Circuits*, Vol. 31, No. 9, Sept. 1996, pp. 1277-1284.
10. M.J. Flynn et al., "Deep-Submicron Microprocessor Design Issues," *IEEE Micro*, Vol. 19, No. 4, July/Aug. 1999, pp. 11-22.
11. D. Burger and T.M. Austin, "The SimpleScalar Toolset, Version 2.0," *Computer Architecture News*, Vol. 25, No. 3, Jun. 1997, pp. 13-25.
12. M. Moudgill, J-D Wellman, and J.H. Moreno, "Environment for PowerPC Microarchitecture Exploration," *IEEE Micro*, Vol. 19, No. 3, May/June 1999, pp. 15-25.
13. M. Moudgill, P. Bose, and J. Moreno, "Validation of Turandot, a Fast Processor Model for Microarchitecture Exploration," *Proc. IEEE Int'l Performance, Computing and Communication Conf.*, IEEE Press, Piscataway, N.J., 1999, pp. 451-457.
14. V. Tiwari et al., "Reducing Power in High-Performance Microprocessors," *Proc. IEEE/ACM* Design *Automation Conf.*, ACM, New York, 1998, pp. 732-737.
15. K. Diefendorff, "Power4 Focuses on Memory Bandwidth," *Microprocessor Report*, Oct. 6, 1999, pp. 11-17.
16. M. Pant et al., "An Architectural Solution for the Inductive Noise Problem Due to Clock-Gating," *Proc. Int'l Symp. Low-Power Electronics and Design,* ACM, New York, 1999.
17. U. Ko, P.T. Balsara, and A.K. Nanda, "Energy Optimization of Multilevel Cache Architectures for RISC and CISC Processors," *IEEE Trans. VLSI Systems,* Vol. 6, No. 2, 1998, pp. 299-308.
18. Theme issue, "The Future of Processors," *Computer*, Vol. 30, No. 9, Sept. 1997, pp. 37-93.
19. G. Sohi, S.E. Breach and T.N. Vijaykumar, "Multiscalar Processors," *Proc. 22nd Ann. Int'l Symp. Computer Architecture*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 414-425.
20. D.M. Tullsen, S.J. Eggers, and H.M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," *Proc. 22nd Ann. Int'l Symp. Computer Architecture,* IEEE CS Press, 1995, pp. 392-403.
21. S. Borkar, "Design Challenges of Technology Scaling," *IEEE Micro,* Vol. 19, No. 4, July/Aug., 1999, pp. 23-29.

Several authors at IBM T.J. Watson Research Center in Yorktown Heights, New York, collaborated on this tutorial. **David Brooks,** a PhD student at Princeton, worked as a co-op student while at IBM Watson.

Pradip Bose is a research staff member, working on power-aware architectures. He is a senior member of the IEEE. **Stanley Schuster** is a research staff member, working in the area of low-power, high-speed digital circuits. He is a Fellow of the IEEE. **Hans Jacobson**, a PhD student at the University of Utah, worked as an intern at IBM Watson. **Prabhakar Kudva**, a research staff member, works on synthesis and physical design with the Logic Synthesis Group. **Alper Buyuktosunoglu,** a PhD student at the University of Rochester, New York, worked as an intern at IBM Watson. **John-David Wellman** is a research staff member, working on high-performance processors and modeling. **Victor Zyuban** is a research staff member, working on low-power embedded and DSP processors. **Manish Gupta** is a research staff member and manager of the High Performance Programming Environments Group. **Peter Cook** is a research staff member and manager of High Performance Systems within the VLSI Design and Architecture Department.

Direct comments about this article to Pradip Bose, IBM T.J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598; pbose@us.ibm.com.