RC21876 (98432) 7 November 2000 **Computer Science**

IBM Research Report

Power-Aware Microarchitecture: Design and Modeling Challenges for the Next Generation Microprocessors

David M. Brooks, Pradip Bose, Stanley E. Shuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, Peter W. Cook

IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

Power-Aware Microarchitecture: Design and Modeling Challenges for the Next Generation Microprocessors

David M. Brooks[1], Pradip Bose, Stanley E. Schuster, Hans Jacobson[2], Prabhakar N. Kudva, Alper Buyuktosunoglu[3], John-David Wellman, Victor Zyuban, Manish Gupta and Peter W. Cook

IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

ABSTRACT

Power dissipation limits have emerged as a major constraint in the design of microprocessors. This is true not only at the low end, where cost and battery life are the primary drivers, but also now at the midrange and high-end system (server) level. Thus, the ability to estimate power consumption at the high-level, during the early-stage definition and tradeoff studies is a key new methodology enhancement sought by design and performance architects. In this paper, we first review the fundamentals in terms of power estimation and power-performance tradeoffs at the microarchitecture level. We then discuss the opportunities of saving power that can be exposed via microarchitecture-level modeling. In particular, the potential savings that can be effected through straightforward clock-gating techniques is cited as an example. We also describe some future ideas and trends in power-efficient processor design. Examples of how microarchitectural observations can be used towards power-saving circuit design optimizations are described. The design and modeling challenges highlighted in this paper are in the context of work in progress within IBM Research. This research is in support of future, high-end processor development within IBM.

1. Introduction

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 (execution time) performance as the primary design issue. Battery life and system cost constraints are what drive the design team to consider power over performance in such a scenario. Increasingly, however, power is a key design issue in the workstation and server market as well (see [21]). In this high-end arena, the increasing microarchitectural complexities, clock frequencies and die sizes are pushing the chip-level (and hence, system-level) power consumption to such levels that traditionally air-cooled multiprocessor server boxes may soon need to budget for liquid-cooling or refrigeration hardware. Such a need is likely to cause a break point (with a step *upwards*) in the ever-decreasing price-performance ratio curve. As such, a design team which considers power consumption and dissipation limits early in the design cycle and is able to adopt an inherently lower-power microarchitectural family 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, there has been a surge of interest in estimating power at the microarchitecture definition stage; and, specific work on power-efficient microarchitecture design has been reported [1-7]. In

[1] David Brooks is a coop graduate student from Princeton University, currently working at IBM.

[2] Hans Jacobson is a graduate student at Univ. of Utah; he contributed to this IBM project during a recent summer internship.

[3] Alper Buyuktosunoglu is a graduate student at Univ. of Rochester, NY; he contributed to this IBM project during a recent summer internship.

this paper, we first dwell on some of the fundamental issues and tradeoffs in power vs. performance from a microarchitect's viewpoint. This includes a discussion of "appropriate" metrics for evaluation of power-performance efficiency at this level. Next, we describe how energy models can be integrated into traditional performance simulators. The resulting trace- or execution-driven power-performance simulators are key new tools for defining tomorrow's energy-efficient microprocessors. We present some example tradeoff analysis experiments, using our own *PowerTimer* toolkit. These experiments were done in the context of future PowerPC™ processor design space exploration studies. Later in the paper, we take a brief look at emerging new paradigms in server-class microprocessor design. We examine the expected performance and power-performance characteristics of these approaches. The analysis is intuitive, based on our experiences with current generation superscalar processors.

Paper Outline:

In Section 2, we review the basics of power and performance at the microarchitecture level. Power-performance numbers and estimation methods typically used are reviewed. The problem of finding a representative set of benchmarks and metrics to compare power-performance efficiency across available processors is highlighted; and, numbers reported for commercial processors are discussed. In Section 3, microarchitectural power estimation models are described with emphasis on a modeling technique developed at IBM Research. Section 4 then describes some of the power-performance optimizations that were discovered as a result of using the estimation tools. In this context, Sidebars A and B describe some of the circuit-level optimizations that could result in significant power savings by taking advantage of microarchitectural observations. Compiler support is a crucial aspect of microarchitecture-level techniques for power-performance optimization. Sidebar C provides a snapshot of the basic infrastructure in place to aid in our research in this area. Section 5 provides the recent trends in power-efficient microarchitecture design. Sidebar D illustrates the issues of power and power-performance efficiency, by using a simple loop-based example.

2. Power-Performance Fundamentals at the Microarchitecture Level

Let us first discuss the basics of power dissipation in a processor chip.

Power Basics:

At the elementary transistor gate (e.g. an inverter) level, total power dissipation can be formulated as the sum of three major components: switching loss, leakage and short-circuit loss (see [5, 8, 9, 13]).

$$PW_{device} = (1/2)C.V_{dd}.V_{swing}.a.f + I_{leakage} . V_{dd} + I_{sc} . V_{dd} \quad \text{..... (2.1)}$$

where, C is the output capacitance, V_{dd} is the supply voltage, f is the chip clock frequency and a is the activity factor ($0 < a <= 1$) which determines the device switching frequency; V_{swing} is the maximum voltage swing across the output capacitor, which in general is less than V_{dd}; $I_{leakage}$ is the leakage current and I_{sc} is the short-circuit current. In the literature, V_{swing} is often approximated to be equal to V_{dd} (or simply V for short) making the switching loss ~ $(1/2)C.V^2$. a.f. Also, as discussed in [8], for current ranges of V_{dd} (say 1 V to 3 V) switching loss: $(1/2)CV^2af$ remains the dominant component. So, as a first-order approximation, for the whole chip, we may formulate the power dissipation to be:

$PW_{chip} = (1/2)([\text{summation over i}]: C_i \cdot V_i^2 \cdot a_i \cdot f_i)$ ..... (2.2)

where, C_i, V_i, a_i and f_i are unit- or block-specific average values in the most general case; the summation is taken over all blocks or units i, at the microarchitecture level (e.g. icache, dcache, integer unit, floating point unit, load-store unit, register files and buses [if not included in individual units], etc.). Also, for the voltage range considered, the operating frequency is roughly proportional to the supply voltage; and the capacitance C remains roughly the same if we keep the same design but scale the voltage. If a single voltage and clock frequency are used for the whole chip, the above reduces to:

$PW_{chip} = V^3 . ([\text{sum over i}]: K^v_i \cdot a_i) = f^3 . ([\text{sum over i}]: K^f_i \cdot a_i)$ ..... (2.3b)

If we consider the worst-case activity factor for each unit i, i.e. if $a_i = 1$ for all i, then,

$PW_{chip} = K_v \cdot V^3 = K_f \cdot f^3$ ..... (2.3c)

where K_v and K_f are design-specific constants.

The last equation, 2.3c is what leads to the so-called "cube-root" rule [8]: i.e., if we take an existing design optimized for frequency and the voltage is lowered, the frequency is reduced by the cube-root of the original power; equivalently, the power reduction ratio is proportional to the cube of the voltage reduction ratio. This implies the single-most efficient method for reducing power dissipation for a processor designed to operate at high frequency: namely, reduce the voltage (and hence the frequency). It is believed that this is the primary mechanism of power control in the Transmeta chip [10]. There is a limit, however, of how low 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. The execution time can be written as:

$T = PL * CPI * CT = PL * CPI * (1/f)$ ..... (2.4)

where 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 the clock frequency f. Since performance increases with decreasing T, one may formulate performance PF as:

$PF_{chip} = K_{pf} \cdot f \sim= K_{pv} \cdot V$ ..... (2.5)

where, the K's are constants for a given microarchitecture-compiler implementation. The K_{pf} value stands for the average number of machine instructions that are executed per cycle on the machine being measured. Performance, PF_{chip}, in this case is measured in units like (millions of) instructions per second, or **mips.**

Selecting a suite of publicly available benchmark programs which everybody accepts as being "representative" of real-world workloads is difficult to begin with. Adopting a non-controversial weighted mix is also not easy. For the commonly used SPEC benchmark suite (see http://www.specbench.org) the SPECmarks rating (for each class: e.g. integer or floating point) 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. This method has the advantage that different machines can be ranked unambiguously from a performance viewpoint, if one believes in the particular benchmark suite. That is, the ranking can be shown to be independent of the reference machine used in such a formulation.

Power-Performance Efficiency Metrics:

The most common (and perhaps obvious) metric to characterize the power-performance efficiency of a microprocessor is a simple ratio, like **mips/watt.** This attempts to quantify the efficiency by projecting the performance achieved or gained (measured in millions of instructions per second) for every watt of power consumed. Clearly, the higher the number, the "better" the machine is. While this seems a reasonable 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 meets maximum power budget constraints) even if it operates at a much lower mips/watt efficiency compared to one that operates at better efficiency but at a lower performance level. As such, $(mips)^2$/watt or even $(mips)^3$/watt 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, one may want to put an even greater weight on the power aspect than the simplest mips/watt metric; i.e. one may just be interested in minimizing the watts for a given workload run, irrespective of the execution time performance, provided the latter does not 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 which can complete up to 4 instructions per cycle, the theoretical peak performance is 4000 mips). If the average completion rate for a given workload mix is p instructions per cycle, then the average mips would equal 1000 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, there are fundamental questions which persist about how to boil down "performance" into a single ("average") rating that is meaningful in comparing a set of machines. Since power consumption varies, depending on the program being executed, the issue of benchmarking is also relevant in assigning an average power rating. In measuring power and performance together for a given program execution, one may use a fused metric like power-delay product (PDP) or energy-delay product (EDP) [13]. In general, the PDP-based formulations are more appropriate for low-power, portable systems, where battery-life is the primary index of energy efficiency. The mips/watt metric is an inverse PDP formulation, where delay refers to average execution time per instruction. The power-delay product, being dimensionally equal to energy, is the natural metric for such systems. For higher end systems (e.g. workstations) the EDP-based formulations are deemed to be more appropriate, since the extra delay factor ensures a greater emphasis on performance. The $(mips)^2$/watt 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 $(mips)^3$/watt, which is an inverse ED^2P formulation. Alternatively, one may use $(cpi)^3$.watt as a direct ED^2P metric, applicable on a "per instruction" basis.

The energy*(delay)^2 metric, or perf^3/power formula is analogous to the cube-root rule [8] which follows from constant voltage scaling arguments (see previous discussion, equations 2.1 through 2.3). Clearly, to formulate a voltage-invariant power-performance characterization metric, we need to think in terms of perf^3/(power). When we are dealing with the SPEC benchmarks, one may therefore evaluate efficiency as (SPECrating)^x/watt, or (SPEC)^x/watt for short; where the exponent value x (= 1, 2, or 3) may depend on the class of processors being compared.

Figure 1 shows the power-performance efficiency data[1] for a range of commercial processors of approximately the same generation. In each chart, the latest available processor is plotted on the left and the oldest one on the right. We have used SPEC/watt, SPEC^2/watt and SPEC^3/watt as the alternative metrics, where SPEC stands for the processor's SPEC95 rating (see definition principles, earlier in this section). For each category, like SPEC^2/watt, the worst performer is normalized to 1, and the 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/watt metric seems to be fairest, with the very latest Intel Pentium-III and AMD Athlon offerings (at 1 GHz) at the top for integer workloads; and, the older HP-PA 8600 (552 MHz) and IBM Power3 (450 Mhz) still dominating in the floating point class. For "power-first" processors targeted towards integer workloads (like Intel's mobile Celeron-333) spec/watt seems to be the fairest.

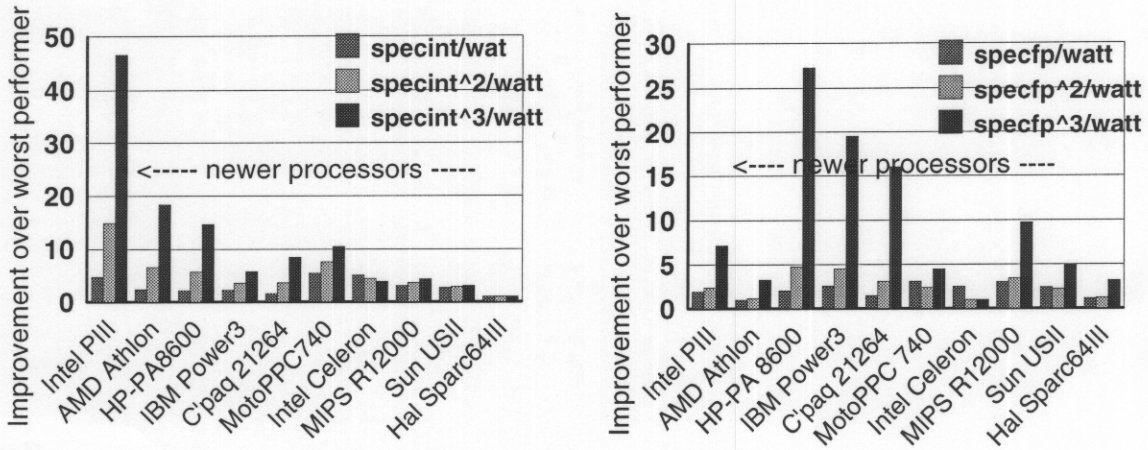


Figure 1. Performance-power efficiencies across commercial processor products

[1] Sources of data include: http://www.bwrc.eecs.berkeley.edu/CIC/ , http://www.specbench.org , Microprocessor Report, August 2000, and individual vendor web pages

Table 1 and 2 below show the explicit ranking of the processors considered in Figure 1, from a power-performance efficiency viewpoint based on specint and specfp benchmarks. The only intent here is to illustrate the point that we tried to make earlier: that depending on the intended market (e.g. general purpose: server-class, workstation or low-power mobile, etc.) and application class (e.g. integer-intensive or floating point intensive) different efficiency metrics may be suitable. Note that we have 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.* As an example, the 33 W maximum power rating for the Intel PIII-1000 processor that we computed from the maximum current and nominal voltage ratings specified for this part in the vendor's web page[1], is higher than that reported in the Microprocessor Report source cited before. Actually, this points to the need of standardization of methods used in reporting maximum and average power ratings for future processors. Coupled with the standard ways of reporting performance, it should be possible, in future for customers to compare power-performance efficiencies across competing products in a given market segment.

Table 1.

	SPECint/watt	**SPECint^2/watt**	**SPECint^3/watt**
1	Moto PPC7400 (450 MHz)	Intel PIII-1000	Intel PIII-1000
2	Intel Celeron (333 MHz)	Moto PPC7400-450	AMD Athlon-1000
3	Intel PIII (1000 MHz)	AMD Athlon-1000	HP-PA8600-552
4	MIPS R12000 (300 MHz)	HP-PA8600-552	Moto PPC7400-450
5	Sun USII (450 MHz)	Intel Celeron-333	Alpha 21264-700
6	AMD Athlon (1000 MHz)	Alpha 21264-700	IBM Power3-450
7	IBM Power3 (450 MHz)	MIPS R12000-300	MIPS R12000-300
8	HP-PA8600 (552 MHz)	IBM Power3-450	Intel Celeron-333
9	Alpha 21264 (700 MHz)	Sun USII-450	Sun USII-450
10	Hal Sparc64-III	Hal Sparc64-III	Hal Sparc64-III

Table 2.

	SPECfp/watt	**SPECfp^2/watt**	**SPECfp^3/watt**
1	Moto PPC7400-450	HP-PA8600-552	HP-PA8600-552
2	MIPS R12000-300	IBM Power3-450	IBM Power3-450
3	IBM Power3-450	MIPS R12000-300	Alpha 21264-700
4	Intel Celeron-333	Alpha 21264-700	MIPS R12000-300
5	Sun USII-450	Intel PIII-1000	Intel PIII-1000
6	HP-PA8600-552	Moto PPC7400-450	Sun USII-450
7	Intel PIII-1000	Sun USII-450	Moto PPC7400-450
8	Alpha 21264-700	Hal Sparc64-III	AMD Athlon-1000
9	Hal Sparc64-III	AMD Athlon-1000	Hal Sparc64-III
10	AMD Athlon-1000	Intel Celeron-333	Intel Celeron-333

[1] http://www.intel.com/design/pentiumiii/datashts/245264.htm

3. Microarchitecture-level Power Estimation Methods

In this section, we provide a brief review of recent work in the area of power estimation at the microarchitecture level. 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 is very similar to the recently published methods used in Princeton University's *Wattch* [3], Penn State's *SimplePower* [4] and George Cai's model at Intel (see [7]).

The core of such models is a classical trace- or execution-driven, cycle-by-cycle performance simulator. In fact, the power-performance models described in [3, 4, 5] and by Cai et al. in [7] are all built upon Burger and Austin's widely-used (publicly available, parameterized) *SimpleScalar* performance simulator [11]. In our case, we are building our toolset around existing, research and production-level simulators (see [18, 19]) used in the various stages of definition and design of high-end PowerPC processors. The key difference, as far as the power projections are concerned, is determined by the nature and detail of the energy models that are used in conjunction with the workload driven cycle simulator.

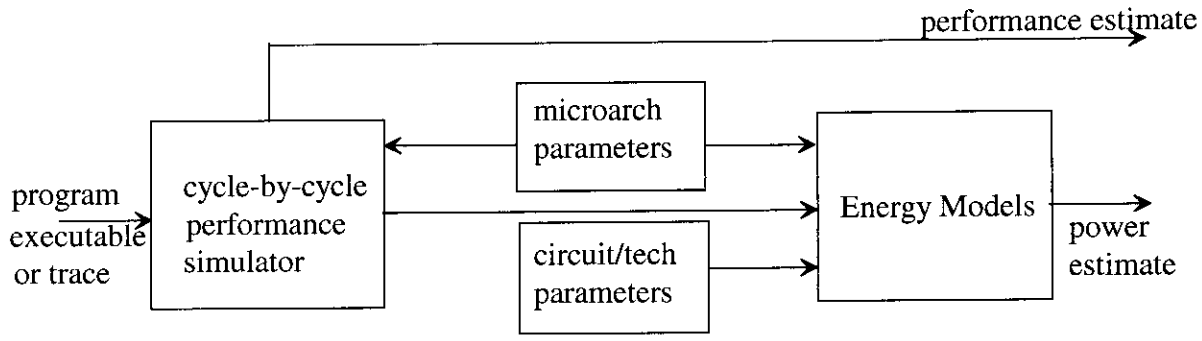


Figure 2. High-Level Block Diagram of PowerTimer

During every cycle of the simulated processor operation, the microarchitecture-level units or blocks that are activated (or busy) 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. These cycle-by-cycle resource usage statistics, available easily from a trace- or execution-driven performance simulator, can essentially be used to estimate the unit-level activity factors a_i referred to earlier in Section 2 (see equations 2.2). If accurate energy models exist for each modeled resource, then, on a given cycle, if unit i is accessed or used, the corresponding energy consumed can be estimated and added to the net energy spent overall and for that unit. So, at the end of the simulation, the total energy spent on a unit basis as well as for the whole processor can be estimated. 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, it is clear that average and peak power dissipation profiles for the whole chip can be estimated, in the context of the input workload. Of course, this kind of analysis implicitly assumes that *clock-gating techniques*

at the circuit-level (e.g., see [22]) can be used to selectively turn off dynamic power consumption in inactive functional units and on-chip storage/communication resources. Figure 3 shows the unit-level utilization 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, like the Power4 processor [12]. The data shown is across a representative sampled trace repository covering a range of benchmark workloads (selections from SPEC95 and tpcc) commonly used in such designs. The functional units tracked in this graph are the floating point unit (FPU), the integer (fixed point) unit (FXU), the branch unit (BRU), the load-store unit (LSU), the branch unit (BRU) and the "logic on condition register" unit (CRU). This simulation data gives us an indication of the potential power savings that can be achieved, by selectively "gating off" non-utilized segments of the processor during the execution of a given workload mix. For example, the FPU is almost totally inactive for the workloads gcc, go, ijpeg, li, vortex and tpcc-db2. The maximum unit utilization is only about 50 % (FXU: m88ksim and FXU: vortex).

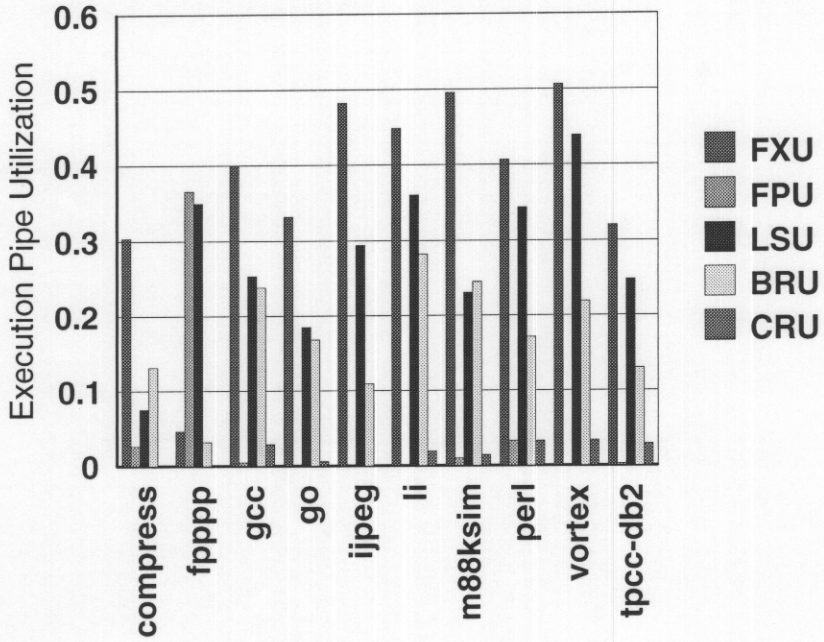


Figure 3. Unit-level utilization data for a current generation server-class superscalar processor

The potential illustrated above is especially significant, if we consider the distribution of power within a modern processor chip. In an aggressive superscalar processor that does not employ any form of clock-gating, a large fraction of the total power (up to 70 %) can be consumed by the clock tree, drivers and the clocked latches in the design. Even with the registers and cache structures accounted for separately, the clock-related power for a recent high performance Intel CPU was reported to be almost half the total chip power [22]. Thus, the use of gated clocks to disable latches (pipeline stages) and entire units when possible, is potentially a major source of power-saving. Depending on the exact hardware algorithm used to gate off

clocks, different characteristics of power-performance optimization can be depicted. For example, in theory, if the clock can be disabled for a given unit (perhaps even at the individual pipeline stage granularity) on every cycle that it is not used, one would 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 like:
(a) the need to include additional on-chip decoupling capacitance to deal with large current swings (L. [di/dt] effects); (b) more complicated timing analysis to deal with additional clock-skew and the possibility of additional gate-delays on critical timing paths, (c) more effort to test and verify the microprocessor.

Despite these additional challenges in the design, some commercial high-end processors have begun to implement clock gating at various levels of granularity: e.g. the Intel Pentium series and Compaq Alpha 21264. At IBM, in our server-class microprocessors, the need for pervasive use of clock gating 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. For example, simpler designs may implement gated clocks in localized "hot-spots" or in league with dynamic feedback data: e.g. if it is detected that the power-hungry floating point unit has been inactive over the last (say) 1000 cycles, then the execution pipes within that unit may be gated off in stages, a pipe at a time to minimize current stages (see [17]). Also, for a given resource, like a register file, it may be possible to conserve power by enabling only the segment(s) in use, depending on the number of ports that are active at a given time. 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. This and other overhead power consumed by the circuitry added to implement gated-clock designs, must be accounted for carefully in simulations to measure the overall advantage. In their Wattch paper [3], Brooks et al. report some of the observed power-performance variations, with different mechanisms of clock-gating control. Other schemes of dynamic adaptation (e.g. see sidebar B) of on-chip resources to conserve power are also of interest in the ongoing power-aware microarchitecture research at IBM.

Asynchronous circuit techniques such as IPCMOS provide the benefits of fine-grained clock gating as a natural and integrated aspect of the design (see sidebar A). Such techniques have the further advantage that current surges are automatically minimized because each stage's clock is generated locally and asynchronously with respect to other pipeline stages. Thus, techniques like IPCMOS have the potential of exploiting the unutilized execution pipe resources (depicted in Figure 1) to the fullest, by burning clocked latch power strictly on demand at every stage of a pipeline, without the inductive noise problem.

The usefulness of microarchitectural power estimators hinges upon the accuracy of the underlying energy models. The energy models for given functional unit blocks (e.g. the integer or floating point execution data path), storage structures (e.g. cache arrays, register files or buffer space) or communication bus structures (e.g. the instruction dispatch bus or the result bus) can be formulated using: (a) circuit-level or register-transfer-level (RTL) simulation of the corresponding structures, using circuit and technology parameters germane to the particular design; or, (b) analytical models or equations which formulate the energy characteristics in terms of the parameters of design of a particular unit or block. In [3] and [5], analytical capacitance equations describing the individual unit and resource structure types are formulated. In [3], for

example, the categories of structures that are modeled for power are: (i) array structures: which include caches, register files and branch prediction tables; (ii) content-addressable memory (CAM) structures, including those used in issue queue logic, translation lookaside buffers (TLBs), etc; (iii) combinational logic blocks and wires, including the various functional units and buses and (iv) clocking structures including the corresponding buffers, wires and capacitive loads.

The methodology for forming energy models in [3] was validated by comparing the energy models for array structures with schematic and layout level circuit extractions for array structures within a commercial Intel microprocessor. They were found to be accurate within 10%. This is similar to the accuracy reported by analytical cache delay models which use a similar methodology. On the other hand, since the core simulation was done at the microarchitectural abstraction, the speed was 1000X or faster compared to existing layout-level power estimation tools.

If detailed power and area measurements for a given chip are possible, it is possible to build reasonably accurate energy models based on power density profiles across the various units. Such models can be used to project expected power behavior for follow-on processor design points by using standard technology scaling factors. Such power-density based energy models have been used in conjunction with trace-driven simulators to do power analysis and tradeoff studies for some Intel processors [7]. In the work that we are pursuing at IBM Research, the energy models developed are of various types: (a) power density based, for design families with available power and area measurements from implemented chips; (b) analytical, in terms of microarchitecture-level design parameters: like issue width, number of physical registers, pipeline stage lengths, misprediction penalties, cache geometry parameters, queue/buffer lengths, etc. The analytical energy behaviors are formulated based on simple area determinants and the constants are validated using circuit-level simulation experiments using representative test vectors. Data-dependent transition variations are not considered in our initial energy model formulations; but will be factored into the next version of the simulator. In our work, we consider validation to be an integrated part of the methodology. Our focus is on building models that are high on *relative* accuracy, i.e. the goal is to enable designers to explore early-stage microarchitecture options which are inherently superior from a power-efficiency standpoint. *Absolute* accuracy of early-stage power projection for a future processor is not as important, so long as tight upper bounds can be established early.

4. Example Power-Performance Tradeoff Result

In this section, we first provide a high-level description of the processor model assumed in our simulation toolkit. Then, we present some example experimental results with analysis and discussion. The results were obtained using our current version of PowerTimer, which works with pre-silicon performance and energy models developed for future, high-end PowerPC processors.

4.1. Base Microarchitecture Model:

Figure 4 shows the high-level organization of our modeled processor. The details of the model make it equivalent in complexity to a modern, out-of-order, high-end microprocessor (e.g. [12]). For the purposes of this paper, we assume a generic, parameterized, out-of-order

superscalar processor model adopted in a research simulator called *Turandot* [18,19]. (Figure 4 is essentially reproduced from [19]).

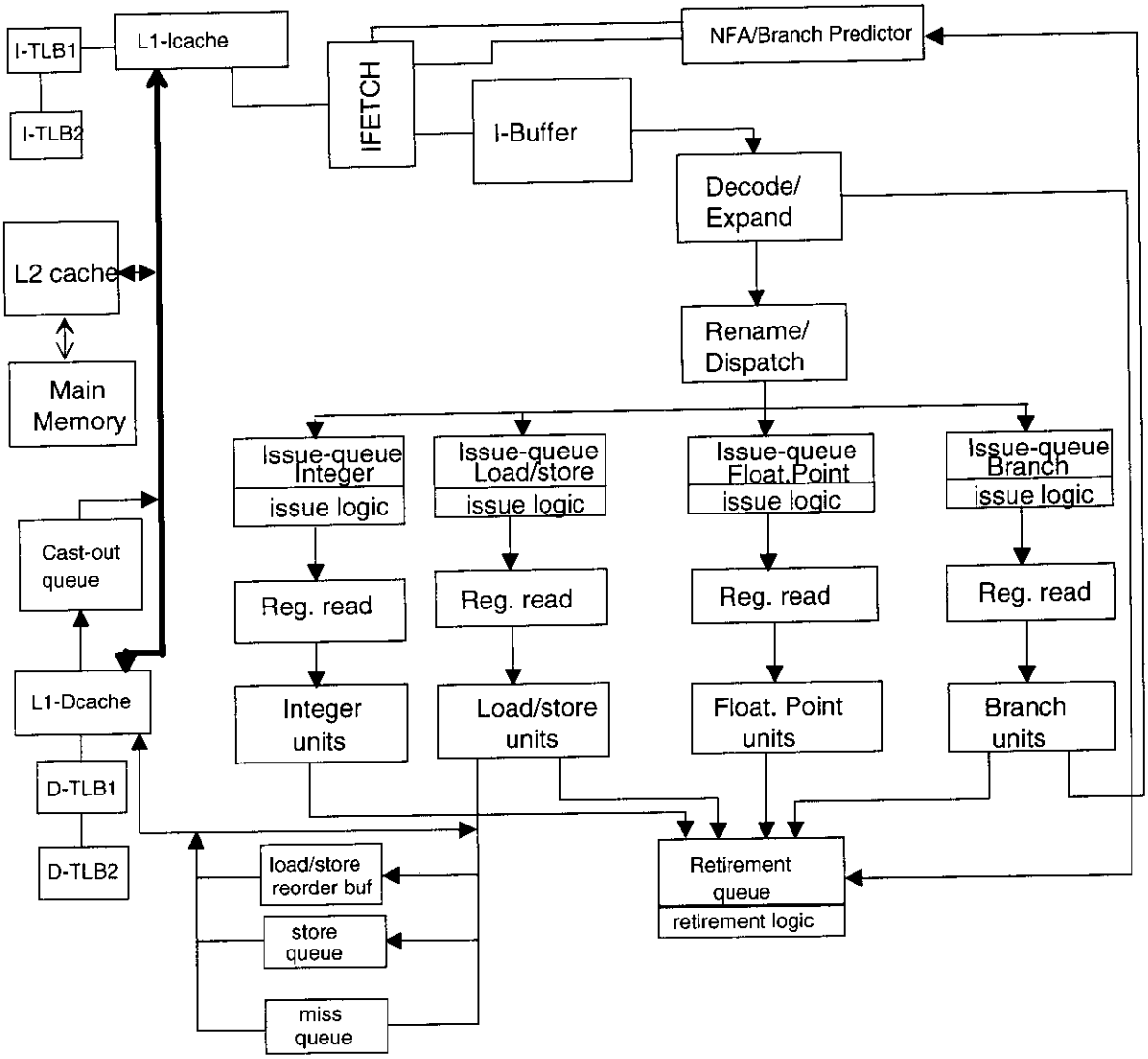


Figure 4. Processor organization modeled by the Turandot simulator

As described in [18, 19], this research simulator was calibrated against a pre-RTL, detailed, latch-accurate processor model (referred to as R-model in [19]). The R-model served as a validation reference in the real processor development project and was developed by the actual design team. The R-model is a custom simulator, written in C++ (with mixed VHDL "interconnect code"). There is a virtual 1-to-1 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 is 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 1-2 orders of magnitude faster

than R-model. It supports a much greater number and range of parameter values (see [19] for details). In the initial version of this paper, we report power-performance results using the same version of R-model which was used in [19]. That is, we decided to use our developed energy models first in conjunction with the R-model: this ensured accurate measurement of the resource utilization statistics within the machine. To circumvent the simulator speed limitations, we used a parallel workstation cluster (farm); also, we post-processed the performance simulation output and fed the average resource utilization 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 have been possible to get instantaneous, cycle-by-cycle energy consumption profiles through such a method, it would not have changed the average power numbers for entire program runs.

4.2. Data Cache Size and the Effect of Scaling Techniques

In this section we evaluate the relationship between performance, power, and L1 data cache size. We vary the cache size by increasing the number of cache lines per set while leaving the linesize and cache associativity constant. Figure 5a and 5b show the results of increasing the cache size from the baseline architecture (points labeled 1x on the x-axes). Figure 5a illustrates the variation of *relative*[1] CPI with the level-1 data cache size. Figure 5b shows the variation when we consider the metric $(CPI)^3$ * power. From Figure 5b, it is clear that the small CPI benefits of increasing the data cache are outweighed by the increases in power dissipation due to larger caches. In Figure 5b, we show the power-performance metric ($(cpi)^3$.power) with two different scaling techniques. The first technique assumes that power scales linearly with the cache size. As the number of lines is doubled, the power of the cache is also doubled. The second scaling technique is based on data from [20] which studied energy optimizations within multi-level cache architectures. In [20], data is presented for cache power dissipation for conventional caches with sizes ranging from 1KB to 64KB. In the second scaling technique, which we call "non-lin", the cache power is scaled with the ratios presented in [20]. The increase in cache power by doubling cache size using this technique is roughly 1.46x, as opposed to the

[1] CPI and $(CPI)^3$ * power values in Figs. 4 and 5 are relative to the baseline (1x) CPI values for each workload.

2x with the simple linear scaling method. Obviously the choice of scaling technique can greatly impact the results; however, it is clear that with either scaling choice, conventional cache organizations[1] will not scale in a power-efficient manner. (Note that the curves shown in Figure 5(b) assume a given, fixed circuit/technology generation; they are intended to show the effect of adding more cache to an existing design).

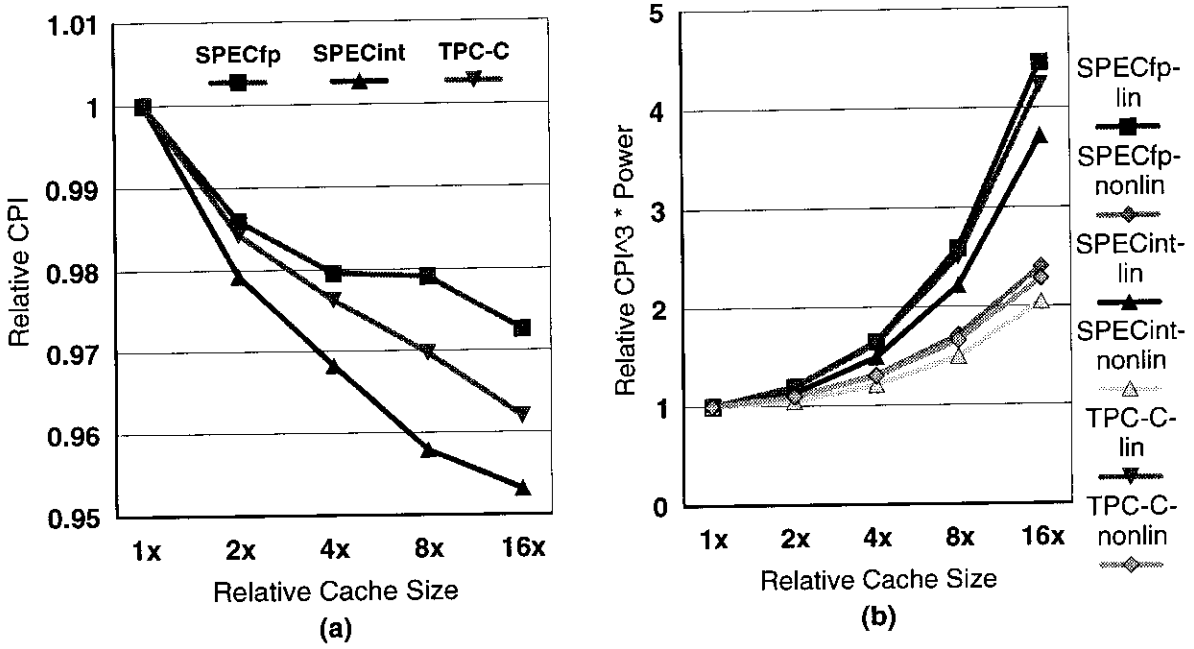


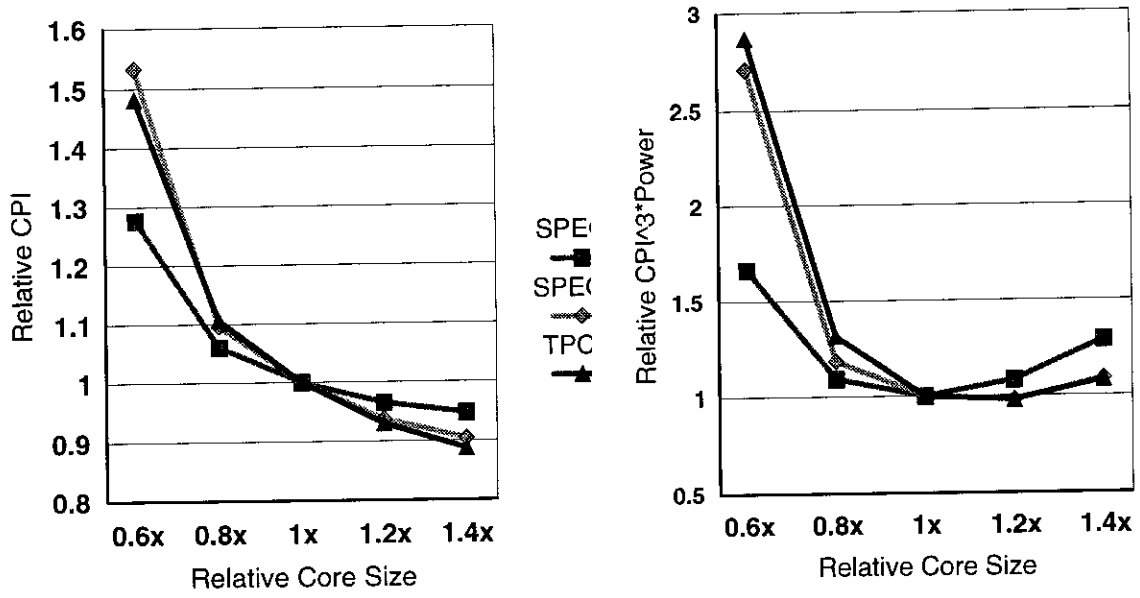
Figure 5. Variation of Performance and Power-Performance with Cache Size

4.3. Ganged Sizing

Out-of-order superscalar processors of the class considered rely on queues and buffers to efficiently decouple instruction execution to increase performance. The depth of the pipeline and the sizes of the resources required to support decoupled execution (queues, rename registers, completion table) combine to determine the performance of the machine. Because of this decoupled execution style, increasing the size of one resource without regard to the other resources in the machine may quickly create a performance bottleneck. Thus, in this section we consider the effects of varying multiple parameters rather than just a single parameter in our modeled processor. Figure 6a and 6b show the effects of varying all of the resource sizes within the processor core. This includes 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 is 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 which are 4x larger). The configuration had a power dissipation of 52%-55% higher than the baseline core. Figure 6b,

[1] i.e., cache designs, that do not use partial array shutdowns to save power.

shows that the most power efficient core microarchitecture is somewhere between the 1x and 1.2x cores.



(a) (b)

Figure 6. Variation of Performance and Power-Performance with Core Size (ganged parms)

5. Power-efficient Microarchitecture Design Ideas and Trends

Once we have the means to evaluate power-efficiency during early-stage microarchitecture evaluations, we have the ability to propose and characterize processor organizations which are inherently energy-efficient. Also, if we use the right efficiency metric in the right context (see earlier discussion in Section 2), we are able to compare alternate design points and order them using a single power-performance efficiency number. In this section, we first examine the power-performance efficiency trend exhibited by the current regime of superscalar processors. That is, we wish to investigate the worth of continuing on the path of wider issue, speculative, out-of-order processors. Later, we look at some alternate paradigms of the future, which have the promise of improved power-performance scalability over time. Sidebar D 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, like SMT may help correct the decline in power-performance efficiency measures.

Single core, wide issue superscalar processor chip paradigm:

One school of thought envisages a continued progression along the path of ever-wider, aggressively speculative superscalar paradigm (e.g. see Patt et al. in [16]). Researchers continue to innovate in the quest of increasing the exploitation of single-thread instruction-level parallelism (ILP). Value prediction advances (see Lipasti et al. in [16]) promise to break the limits imposed by data dependency limits. Trace processors (see Smith et al. in [16]) ease the fetch bandwidth bottleneck, which can otherwise impede scalability. Nonetheless, in spite of

such advances, increasing the superscalar width beyond a certain limit seems to yield ever-diminishing gains in performance, while continuously reducing the performance-power efficiency metric (e.g. $SPEC^3$/watt). Thus, studies show that for the superscalar paradigm, going for ever-widening issue widths and speculative hardware in the core processor ceases to be viable beyond a certain complexity point (see illustrative example in Sidebar D). Such a point certainly seems to be upon us in the processor design industry. In addition to power issues, ever-complicated, ever-bigger cores add to the verification and yield problems: which all add up to higher cost and delays in time-to-market. Advanced methods in low-power circuit techniques (see sidebar A) and adaptive microarchitectures (see sidebar B) can help us extend the superscalar paradigm for a few more years. Other derivative designs, like multicluster processors and SMT (see below) can be considered to be more definitive paradigm shifts.

Multicluster superscalar processors:

Zyuban in his Ph.D work [5], has studied the class of *multi-cluster* superscalar processors as a means to extend the power-efficient growth of the basic superscalar paradigm. In this sub-section, we provide a very brief overview of the main results and conclusions of Zyuban's work (see also [6]). Elements of this work will be used in the modeling and design work in progress within the power-aware microarchitecture project at IBM.

As implied in the preceding discussion, the desire to extract more and more ILP using the superscalar approach requires the growth of most of the centralized structures. Among them are: the 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. Detailed analysis of circuit-level implementation of these structures was performed by Zyuban. This analysis shows that in most of the cases, the energy dissipated per instruction grows in a super-linear fashion with respect to the issue width. None of the known circuit techniques solves this energy growth problem. Given that the IPC performance grows sub-linearly with issue width (with asymptotic saturation), it is clear why the "classical" superscalar path will lead to increasingly power-inefficient designs. One way to address the energy growth problem at the micro-architectural level is to replace a classical superscalar CPU with a set of clusters, so that all key energy consumers are split between 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 makes each access path simpler (i.e. lower latency) and lower power.

The current generation of server-class processors (e.g. the Compaq Alpha 21264 and IBM's Power4) certainly has elements of multi-clustering, especially in terms of duplicated register files and distributed issue queues, etc. In his work, Zyuban proposed and modeled a specific multicluster organization. Simulation results [5] showed that in order to compensate for the overhead of inter-cluster communication and get a significant improvement in power-performance efficiency (over the classical superscalar processor), each cluster must be a powerful out-of-order superscalar machine by itself. This simulation-based study was able to determine the optimal number of clusters and their configurations, for a specified efficiency metric (e.g. the energy-delay product).

The multi-cluster organization does yield IPC performance that is inferior to a classical superscalar with centralized resources (assuming equal net issue width and total resource sizes). The latency overhead of inter-cluster communication is the main reason behind the IPC shortfall. Another reason is that centralized resources are always better utilized than distributed ones. However, Zyuban's simulation data shows that the multi-cluster organization is potentially more energy-efficient for wide issue processors with an advantage that grows with the issue width [6]. Given the same power budget, the multi-cluster 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) processor family. The promise here is (or was) that much of the hardware complexity could be moved to the software (compiler). The latter would do global program analysis to look for available parallelism and present explicitly parallel execution packets to the hardware, which could be relatively simple in that it would avoid much of the dynamic unraveling of parallelism that a modern superscalar does. The reader is referred to the preceding issue of IEEE Micro, which is devoted to Itanium, for inferences regarding performance, power and efficiency metrics for this class of microarchitectures.

Chip multiprocessing:

Server product groups like IBM's PowerPC division have relied on chip multiprocessing as the scalable paradigm for the future. The Power4 design [12] is the first example of this trend. Here, the promise is to be able to build multiple processor cores on the same die or package to deliver scalable solutions at the system level. Each building block core is a limited (e.g. 4-way) issue-width superscalar.

Multiple processor cores on a single die can be architected to 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. in [16]), one may consider building a multiscalar processor [15], which can spawn off speculative tasks (derived from a sequential binary) to execute concurrently on multiple cores. Inter-task 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 hazard, the offending task(s) must be squashed and restarted. Multiscalar-like paradigms have the promise of providing scalable, power-efficient designs, if the compiler-aided task partitioning 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, which promises to provide a significant boost in performance with a small increase in hardware complexity is that of simultaneous multithreading or SMT [14]. 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 of the processor. Because the issue and execution resources can be filled by instructions from multiple independent threads in the same cycle, the per-cycle utilization of the processor

resources can be significantly improved, leading to much better processor throughput. The Compaq 21x64 family has already announced that its future designs will embrace the SMT paradigm.

For occasions when only a single thread is executing on an SMT processor, the processor behaves almost exactly like a traditional superscalar machine, and thus the same power reduction techniques are likely to be applicable. When an SMT processor is simultaneously executing multiple threads, however, the per-cycle utilization of the processor resources should be noticeably increased, 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, does provide additional options for power-aware design.

The SMT processor does generally provide 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) where there is a greater emphasis on the overall (throughput) performance than on low-power (e.g. where a $[(cpi)^3 * watt]$ metric is utilized). Furthermore, note that a processor that is specifically designed with SMT in mind can provide even greater power performance efficiency gains.

Because the SMT processor can take advantage of multiple threads of execution, 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 utilization when speculation would have been utilized 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 non-speculative instructions. This inherently implies a greater power efficiency per thread; i.e. the power expended in the execution of *useful* instructions weighs better against *mis-speculated* instructions on a "per-thread" basis. This also implies a somewhat simpler branch unit design (e.g. fewer resources devoted to branch speculation) which can further aid in the development by reducing design complexity and verification effort.

One of the major drawbacks of an SMT system is that it requires the processor to keep independent copies of the multiple thread states. In the case of a simple SMT processor, this implies a physical register for each of the architectural registers of every thread, so a 4-way SMT processor will necessarily require four times as many physical registers as each thread has architected registers (plus any required for register renaming). In most cases, for example, this implies the processor would have to have at least 128 physical general-purpose registers for an instruction set that supports 32 architected registers, and additional registers beyond these 128 if the processor will include register renaming. There is a further requirement that each thread in an SMT processor have a separate program counter and whatever associated resources as a necessary to properly capture the relevant state of a thread. Clearly, this increase in hardware does imply some increase in the per-cycle power requirements for the processor. The question to be answered for each SMT design is whether this increase in processor resources (and thus per-cycle power) can be well balanced by the reduction of other resources (e.g. less speculation) and the increase in performance attained across the multiple threads.

Compiler Support:

Compilers can be used to assist the microarchitecture in reducing the power consumption of programs. As explained in Sidebar C, a number of compiler techniques, which have been developed for performance-oriented optimizations, can be exploited (usually, with minor

modifications) to achieve power reduction by effects like reducing the number of memory accesses, reducing the amount of switching activity in the CPU, and increasing opportunities for clock-gating.

Energy-efficient cache architectures:

In this article, our focus has been on the core microarchitecture. In terms of power-efficient solutions to the cache hierarchy design, there have been several proposals (see [1-4, 20]). The simplest of these is the filter cache idea, first proposed by Kin et al. (see reference in Sidebar B). In more recent work, Albonesi (see sidebar B) has investigated the power-performance tradeoffs that can be exploited by *dynamically* changing the cache sizes and clock frequencies during program execution.

6. Conclusion

In this paper, we address the issue of power estimation and power-efficient designs at the microarchitecture level. Recently, there has been a surge of interest in this domain as complexity and power dissipation have emerged as key constraints even in the design of server-class, general purpose processors. The goal of this paper is to serve as a tutorial-style contribution to help understand the issues and trends in this new area. We start with a description of the fundamental issues in microarchitecture-level estimation of performance and power. We examine the metrics that have been proposed in the literature to quantify power-performance efficiency and point out the preferred metrics for various classes of processors. We emphasize the point that depending on whether the priority is high performance or low power, and what form of design-scaling is used for energy reduction, one should perhaps use different metrics to compare a given class of competing processors. We then review the basic, simulation-based approaches that have been used by R&D groups to study power-performance tradeoffs at the microarchitecture level. Wherever relevant, we use examples from our own modeling experience within IBM to illustrate specific issues. We also provide a brief discussion of emerging new paradigms of the future which are promising from the point of sustaining the power-efficient performance growth curve for server-class processor chips of the future.

The need for robust power-performance modeling at the microarchitecture-level will continue to grow with tomorrow's workloads and performance requirements. Such models will enable designers to make the right choices in defining the future generation of energy-efficient microprocessors. As explained by Borkar [9], 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 the task of identifying energy-saving opportunities in future designs.

Acknowledgements

The authors are grateful to other members of the research workgroup on power-efficient server design. Special thanks are due to Peter Hofstee of IBM Austin Research Laboratory for his insight and input on voltage-invariant scaling and its impact on power-performance metrics. Finally, the support and input received from our various university research partners are gratefully acknowledged.

REFERENCES

1. Papers presented at: 1998 ISCA Workshop on Power-Driven Microarchitecture; http://www.cs.colorado.edu/~grunwald/LowPowerWorkshop/agenda.html
2. Papers presented at: 2000 ISCA Workshop on Complexity-Effective Design; http://www.ece.rochester.edu/~albonesi/ced00.html
3. D. Brooks, V. Tiwari and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *Proc. 27th. Ann. Int'l. Symp. on Computer Architecture (ISCA),* pp. 83-94, June 2000.
4. N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, W. Ye, "Energy-driven integrated hardware-software optimizations using SimplePower," *Proc. 27th. Ann. Int'l. Symp. on Computer Architecture (ISCA),* pp. 95-106, June 2000; also: M. J. Irwin and V. Narayanan, "Low power design: from soup to nuts," Tutorial Notes, presented at *27th. Ann. Int'l. Symp. on Computer Architecture (ISCA),* June 2000.
5. V. Zyuban, "Inherently lower-power high performance super scalar architectures," Ph.D thesis, Univ. of Notre Dame, Dept. of Computer Science and Engineering; January 2000.
6. V. Zyuban and P. Kogge, "Optimization of high-performance superscalar architectures for energy efficiency," *IEEE Symposium on Low Power Electronics and Design,* August 2000.
7. Talks presented at the Cool Chips Tutorial, *IEEE Int'l. Symp. on Microarchitecture, MICRO-32,* Haifa, December 1999.
8. M. J. Flynn et al., "Deep-submicron microprocessor design issues," *IEEE Micro,* vol. 19, no. 4, pp. 11-22, July/August 1999.
9. S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23-29, July/August 1999.
10. D. R. Ditzel, "Transmeta's Crusoe: a low power x86-compatible microprocessor built with software," Transmeta talk given at IBM T. J. Watson Research Center, June 2000 (see also: http://www.transmeta.com).
11. D. Burger and T. M. Austin, "The SimpleScalar Toolset, Version 2.0," *Computer Architecture News,* vol. 25, no. 3, June 1997, pp. 13-25.
12. K. Diefendorff, "Power4 focuses on memory bandwidth," *Microprocessor Report,* pp. 11-17, October 6, 1999.
13. R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journ. of Solid-State Circuits*, vol. 31, no. 9, pp. 1277-1284, Sept. 1996.
14. D. M. Tullsen, S. J. Eggers and H. M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism," *Proc. 22nd Ann. Int'l. Symp. on Computer Architecture*, pp. 392-403, June 1995.
15. G. Sohi, S. E. Breach and T. N. Vijaykumar, "Multiscalar Processors," *Proc. 22nd. Ann. Int'l. Symp. on Computer Architecture*, pp. 414-425, June 1995.
16. *IEEE Computer*, vol. 30, no. 9, Sept. 1997; theme issue on: "The Future of Processors."
17. M. Pant, P. Pant, D. Wills and V. Tiwari, "An architectural solution for the inductive noise problem due to clock-gating," *Proc. Int'l. Symp. on Low-Power Electronics and Design*, August 1999.
18. M. Moudgill, J-D Wellman, J. H. Moreno, "Environment for PowerPC microarchitecture exploration," *IEEE Micro*, vol. 19, no. 3, pp. 15-25, May/June 1999.
19. M. Moudgill, P. Bose, J. Moreno, "Validation of Turandot, a fast processor model for microarchitecture exploration," *Proc. IEEE Int'l. Performance, Computing and Communication Conference,* IEEE Press, Piscataway, N. J., Feb. 1999, pp. 451-457.
20. U. Ko, P. T. Balsara and A. K. Nanda, "Energy optimization of multilevel cache architectures for RISC and CISC processors," *IEEE Trans. on VLSI Systems,* vol. 6, no. 2, pp. 299-308, June 1998.
21. M. K. Gowan, L. L. Biro, D. B. Jackson, "Power considerations in the design of the Alpha 21264 microprocessor.," Proc. IEEE/ACM Design Automation Conf., pp. 726-731, 1998.
22. V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, F. Baez, "Reducing power in high-performance microprocessors," *Proc. IEEE/ACM Design Automation Conf.*, pp. 732-737, 1998.

SIDEBAR A

Low Power, High Performance Circuit Techniques

Stanley Schuster, Peter Cook, Hans Jacobson, Prabhakar Kudva

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.1 (reproduced from [1]) shows how a typical block is 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 is an operation to perform.

Measured results on an experimental chip demonstrate robust operation for IPCMOS at 3.3GHz under typical conditions and 4.5GHz under best case conditions in a 0.18 micron 1.5V CMOS technology. Since the locally generated clocks for each stage are active only when the data to a given stage is valid, power is conserved when the logic blocks are idling. Furthermore, with the simplified clock environment it is possible to design a very simple single stage latch that can capture and launch data simultaneously without the danger of a race. 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 which 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 which implements the asynchronous interlocking between stages. (The reader is referred to [1] for details). A significant power reduction results when there is 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 CAM/RAM structures:

Fast memory structures consume significant power due to their operating mechanism. While some of this power is consumed in latches and control logic associated with the memory, the majority of the power is consumed in the memory circuit structure itself. A row of memory cells forms a memory word. All memory cells in a column is connected to a common wire called a bitten. When a memory word is read, the gate of all read transistors of the memory cells in one row must be driven to a high value. The read transistors then discharge the bit-planes if a zero is being read, else the bitten remains high. The capacitances on the wordlines and bitlines are quite high because of all the transistors connected to them. Significant power is therefore consumed when accessing the memory. To save power, techniques at both the microarchitectural and circuit

level are becoming important in order to lower the capacitance that is actively being driven on bitlines and wordlines. One effective way of reducing power consumption on memory reads is to divide the memory wordlines into separate banks. Banked memory structures are typically used to improve access time. However, banking can also be used to reduce power consumption. Since only the sub-bitline in one bank needs to be discharged and precharged on a read, rather than the whole bitline, energy can be saved.

Content addressable memory (CAM) [4] is another type of memory structure with a high power consumption (see Figure A.2). A CAM usually can be read and written just like a normal memory. It has an additional functionality however, in that it provides a match search of its contents. A search is performed by simultaneously comparing all memory words against an externally supplied word. A set of matchlines then indicate what memory words matched the external word. To make the process of matching fast, precharged logic is often used to implement the match logic. The logic for a precharged wired XNOR match function is illustrated in Figure 2(b). Both the matchlines, which produce the match results, and the taglines, which drive the external word to be compared against, are high capacitance wires due to the many transistors connected to them. The way the precharged wired XNOR logic is structured, the matchline needs to be precharged to a high value and then be discharged to ground if there was a mismatch. If there was a match then the precharged high value remains and indicates a match. Since in most processor applications, the rate of mismatches is significantly higher than that of matches, a lot of power is wasted as the matchline precharges and discharges on virtually every cycle. Another source of the high power consumption of CAMS is the need to drive and clear the high capacitance taglines every cycle. The taglines need to be cleared before the matchline node can be precharged, to avoid short circuit power consumption. Due to the very high power consumption of CAMs, much more than RAMs, CAM structures must generally be kept fairly small to deal with the power dissipation problem. New CAM structures that can offer significantly lower power while offering performance comparable to wired XNOR CAMs may be needed for future processors. Such new circuit structures have recently been studied as part of our research in this area. CAM structures may be used to implement register files with renaming support; and, also for issue queue structures (see sidebar B) that require support for operand and instruction identifier matching.

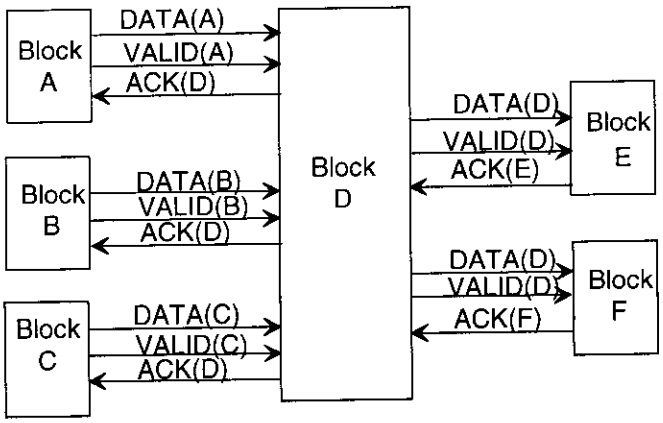


Figure A.1. Interlocking at block level

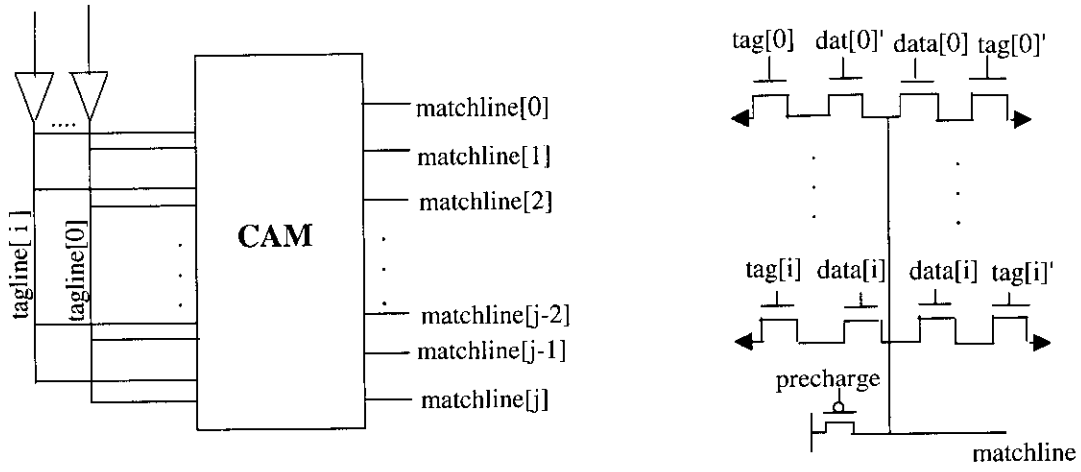


Figure A.2. CAM structure and match logic

References:

[1] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. Immediato, K. Jenkins, "Asynchronous interlocked pipelined CMOS circuits operating at 3.3-4.5 Ghz," *Proc. 2000 IEEE Int'l. Solid-State Circuits Conference (ISSCC),* paper WA 17.3, pp. 292-293.
[2] I. Sutherland, "Micropipelines," *Communications of the ACM,* vol 32, no. 6, June 1989.
[3] Mead and Conway, Introduction To VLSI Systems Addison-Wesley Publishing Company Inc., 1980, Chapter 7.
[4] N. Weste and K. Eshraghian, Principles of VLSI Design, Addison-Wesley Publishing Company Inc., 1988.

SIDEBAR B

Adaptive Microarchitectures

Alper Buyuktosunoglu (and the other co-authors of this paper)

Current generation microarchitectures are usually non-adaptive: i.e. they use fixed-size resources and a fixed functionality across all program runs. The choices are made to achieve best overall performance over a range of applications. However, an individual application whose requirements are not well-matched to this particular hardware organization may exhibit poor performance. Even a single application run may exhibit enough variability that causes uneven use of the chip resources during different phases. The result may often be low unit utilization (e.g.

see Figure 3 in the main article), unnecessary wastage of power and higher access latencies (than required) for storage resources. Albonesi et al. [1, 2] in their CAP (complexity-adaptive processors) project at the University of Rochester, have studied the power and performance advantages in dynamically adapting the on-chip resources in tune with changing workload characteristics. For example, the cache sizes can be adapted on demand; a smaller cache size can burn less power while allowing for faster access. Thus, power and performance attributes of a (program, machine) pair can both be enhanced via dynamic adaptation. Incidentally, a *static* solution which attempts to exploit the fact that most of the cache references can be trapped by a much smaller "filter cache" [6] 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 Albonesi et al.'s group at University of Rochester, in implementing aspects of such dynamic adaptation in a prototype research processor. In particular, we have finished the high-level design (with simulation-based analysis at the circuit and microarchitecture level) of an adaptive issue queue [3] that feeds an IPCMOS-based (see Sidebar A) execute unit. The out-of-order issue queue structure is a major contributor to the overall power consumption of a modern superscalar core. In our research, we have experimented with a power-efficient, adaptive issue queue in which the latter is dynamically powered down and up (in chunks). The resizing is controlled by the dynamic workload behavior. The activity of the issue queue is continuously monitored by added counter-based circuitry. This added logic is on the side, and does not impact the cycle time of the processor. Over a defined "cycle window" (that can either be fixed or variable by design) if the issue queue block farthest from the execute pipe is deemed to be idle or notably under-utilized, it is quickly disabled (powered down) using circuit-level controls. The disabling takes effect after any valid instructions in that issue block have issued for execution. Similarly, the decision logic generates control signals to increase the issue queue size, when it determines that the smaller size is a performance bottleneck for the issue-execute sub-system. Circuit-level simulations have shown that the issue queue energy can essentially be reduced linearly, with decrease in dynamic size. The transistor and power overhead of the added counter-based monitoring and decision control logic is less than 2 %. Details of this design with measurements will be available soon in a research report and external publication [3].

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. [4] 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 is determined that the processor is likely to be 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 mis-speculated executions, thereby saving power. Manne et al. have shown that such power reductions can be effected with minimal loss of performance. Brooks et al. [5] 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. This 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," *Proc. ISCA-25 Workshop on Power-Driven Microarchitecture*, June 1998; see also the author's paper in the regular ISCA-25 proceedings.
2. R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu and S. Dwarkadas, "Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures," *Proc. 33rd Int'l. Symp. on Microarchitecture,* Micro-33, December 2000 (to appear).
3. A. Buyuktosunoglu et al., "An adaptive issue queue for reduced power at high performance," under clearance for publication, September 2000.
4. S. Manne, A. Klauser and D. Grunwald, "Pipeline gating: speculation control for energy reduction," *Proc. 25th. Ann. Int'l. Symp. on Computer Architecture (ISCA-25),* pp. 132-141, June/July 1998.
5. D. Brooks and M. Martonosi, "Dynamic thermal management for high performance microprocessors," *Proc. 7th Int'l. Symp. on High Performance Computer Architecture* HPCA-7, Monterrey, Mexico, January 2001 (to appear).
6. J. Kin, M. Gupta, W. Mangione-Smith, "The filter cache: an energy-efficient memory structure," *Proc. IEEE Int'l. Symp. on Microarchitecture*, pp. 184-193, December 1997.

SIDEBAR C
Compiler Support

Manish Gupta

A high-level organization of the IBM XL family of compilers is shown in Figure C.1. 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 which performs classical data flow optimizations like 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 using 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-like 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. Power-aware instruction scheduling can be used to increase opportunities for clock-gating of CPU components, by trying to group

instructions with similar usage of resources, as long as the impact on performance is kept to a reasonable limit.

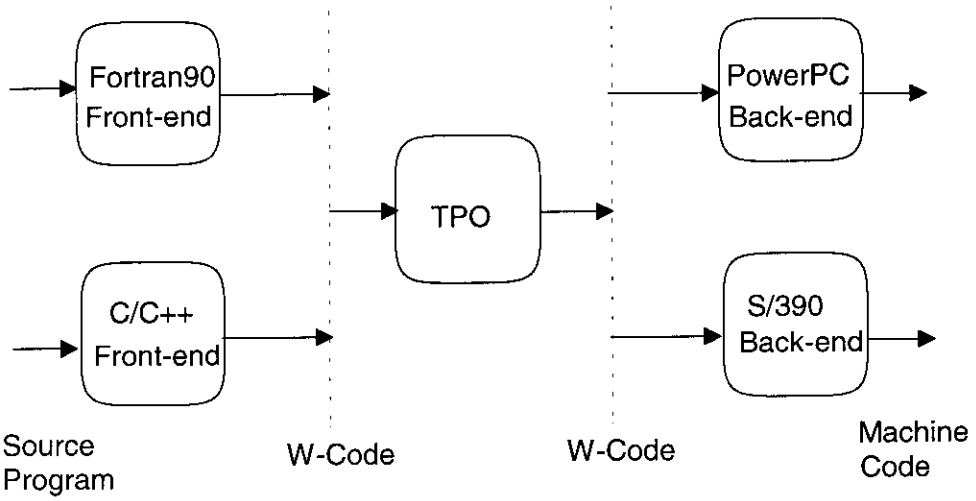


Figure C.1. Architecture of IBM XL compilers

References:

[1] U. Banerjee. Loop transformations for restructuring compilers. Kluwer Academic Publishers, Boston, MA, 1997.

[2] C. Kulkarni, K. Danckaert, F. Catthoor, M. Gupta. Interaction between data-parallel compilation and data transfer and storage cost for multimedia applications. Proceedings of EuroPar'99, Toulouse, France, October 1999.

[3] M. Kandemir, N. Vijaykrishnan, M.J. Irwin, H.S. Kim. Experimental evaluation of energy behavior of iteration space tiling. Proceedings of the 13th Workshop on Languages and Compilers for Parallel Computing, Yorktown Heights, New York, August 2000.

SIDEBAR D

Illustrating the differences between "classical super scalar", SMT and the multiscalar form of CMP; and pointing out the energy efficiency issues.

Let us consider the following floating-point loop kernel, shown in Table D-1.

Table D-1. Example Loop Test Case.

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

The loop body consists of 7 instructions, the final one being a conditional branch that causes control to loop back to the top of the loop body. The instructions are labeled A through G. (The labels T through Z are used to tag the corresponding instructions for a parallel thread - when we consider SMT and CMP). The lfdu/stfdu instructions are load/store instructions with update, where the base address register (e.g. r1, r2 or r3) is updated after execution by holding the newly computed address.

The base machine (refer to Figure 4 in the main article) is assumed to be a 4-wide superscalar, 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 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 above, we essentially focus on the *LSU-FPU sub-engine* of the whole processor.

Let us assume the following high-level parameters (latency and bandwidth) characterizing the base super scalar machine, of width W = 4.

* Instruction fetch bandwidth, fetch_bw = 2*W = 8 instructions/cycle.
* Dispatch/decode/rename bandwidth, disp_bw = W = 4 instructions/cycle; dispatch stalls beyond the first branch scanned in the instruction fetch buffer.
* Issue_bandwidth from LSQ (reservation station), lsu_bw = W/2 = 2 instructions/cycle
* Issue_bandwidth from FPQ, fpu_bw = W/2 = 2 instructions/cycle.
* Completion bandwidth, compl_bw = W = 4 instructions/cycle
* Back-to-back dependent floating point operation issue delay, fp_delay = 1 cycle.
* The best-case load latency, from fetch to writeback is: 5 cycles
* The best-case store latency, from fetch to writing in the pending store queue is: 4 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 writeback is: 7 cycles (when the issue queue, FPQ is bypassed, because it is empty).

Loads and floating point operations are eligible for completion (retirement) the cycle after writeback into rename buffers. For simplicity of analysis let us assume that the processor uses *in-order issue* from the issue queues (LSQ and FPQ). In our simulation model, the 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 are varied 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].

A simple analytical energy model is assumed, where the power consumed is a function of the following parameters: W, ls_units, fp_units, l_ports and s_ports. In particular, the power, PW, in pseudo-watts is computed as: $PW = W^{0.5} + ls\_units + fp\_units + l\_ports + s\_ports$. Figure D-1 shows the performance and performance/power ratio variation with superscalar width, W. The MIPS values are computed from the cpi values, assuming a clock frequency of 1 Ghz.

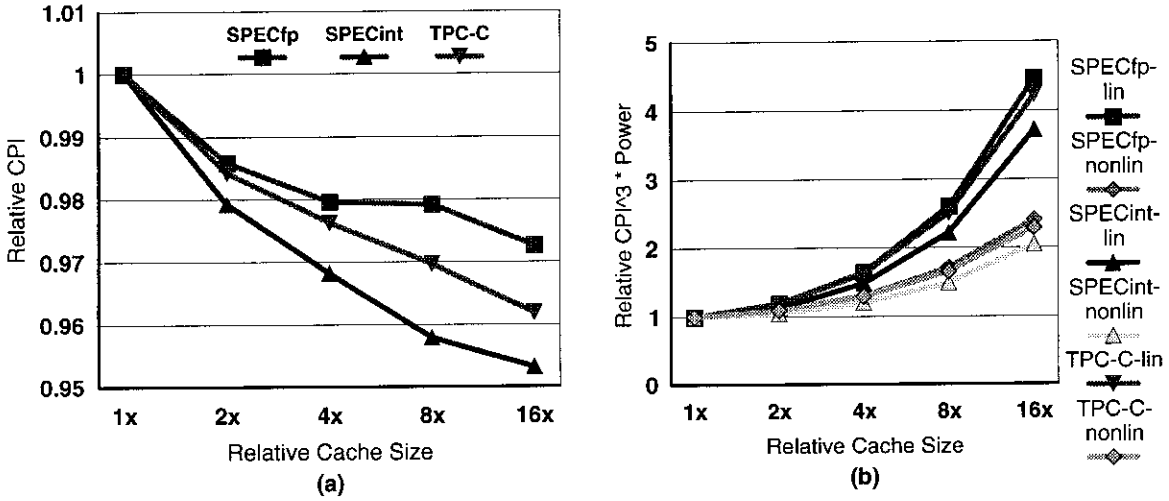


Figure D-1. Loop performance and performance/power variation with issue width

The graph shows that a maximum issue width of W = 4 could be used to achieve the best (idealized) CPI performance. However, as shown in Figure 4(b), from a power-performance efficiency viewpoint (measured as a performance over power ratio in this example), the best-case design is achieved for W = 3. Depending on the sophistication and accuracy of the energy model (i.e. how power varies with microarchitectural complexity), and the exact choice of the power-performance efficiency metric, the inflexion point in the curve in Figure D-1 (b) will change; however, it should be obvious that 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 being considered (as apparent from Figure D-1 (a)).

Note, by the way that the resource sizes are assumed to be large enough, so that they are effectively infinite for the purposes of our running example above. Some of the actual sizes assumed for the base case (W=4) are:

Completion (reorder) buffer size, cbuf_size = 32; Load-store queue size, lsq_size = 6; Floating point queue size, fpq_size = 8; Pending store queue size, psq_size = 16;

As indicated in section 5 of the main article, the microarchitectural trends beyond the current superscalar regime, are effectively targeted towards the goal of extending the processing efficiency factors. That is, the complexity growth must ideally scale at a slower rate than the growth in performance. 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 non-decreasing function of time is a way of achieving complexity-effective designs (see [2] in the main paper reference list).

Let us examine one of the paradigms discussed in Section 5: namely SMT, to understand how this may affect our notion of power-performance efficiency. Table D-2 shows a snapshot of the *steady-state*, cycle-by-cycle, resource utilization profile for our example loop trace executing on the baseline, W=4 superscalar machine (with fp_delay = 1). In the figure, we show 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 utilization 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 D-1, that the steady-state cpi for this case is 0.28.

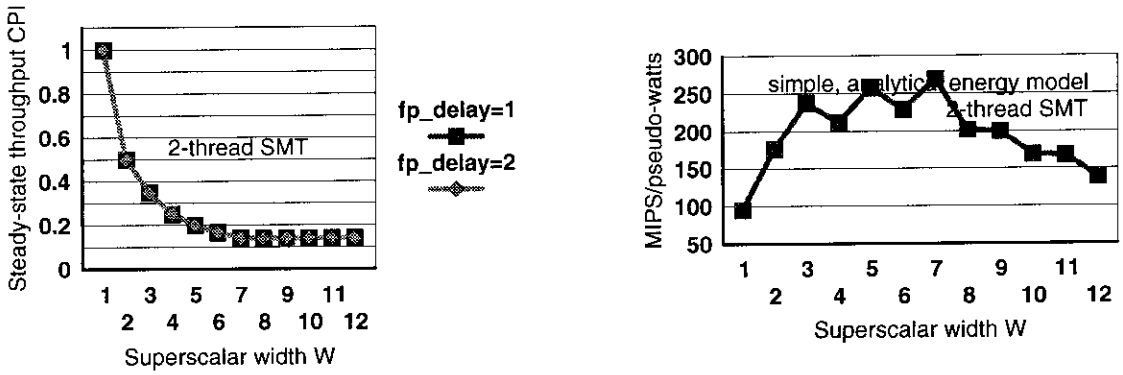
Table D-2. Steady-state, cycle-by-cycle resource utilization profile (W=4 superscalar)

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

The data in Table D-2 shows that the steady-state utilization of the CBUF is 53 %, the LSU0/FPU0 pipe is fully utilized (100 %) and the average utilization of the LSU1/FPU1 pipe is 50 % [(0.6 + 0.4)/2 = 0.5, for FPU1]. The LSQ and FPQ are totally *unutilized* (0 %), because of the queue bypass facility and the relative lack of dependence-stalls. The average read/write port utilization 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 %). Since, due to fundamental ILP limits, the cpi will not decrease beyond W=4, while queue/buffer sizes and added pipe utilization factors will go on decreasing as W is increased, it is clear why power-performance efficiency will be on a downward trend (see Figure D-1). (Of course, here we assume maximum processor power numbers, without any clock gating or dynamic adaptation to bring down power).

With SMT, assume that we can fetch from two threads (simultaneously, if the icache is dual-ported, or in alternate cycles if the icache remains single-ported). Suppose two copies of the same loop program (Table D-1) 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 utilization factors, and the net throughput performance to go up, without a significant increase in the maximum clocked power. This is because, W is not 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 a bit without a performance hit. For example, the fp_delay parameter can be increased, to reduce core complexity, without any performance degradation. Figure D-2 shows the expected performance and power-performance variation with W for the 2-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. (The fraction added is assumed to be linear in the number of threads, in an n-thread SMT).



(a) (b)

Figure D-2. Performance and power-performance variation with W for 2-thread SMT

Figure D-2 shows that under the assumed model, the power-performance efficiency scales better with W, compared with the base superscalar (Figure D-1).

In a multiscalar-like CMP machine, different iterations of a *single* loop program could be initiated as separate tasks or threads on different core processors on the same chip. Thus, the threads A-B-C-D-E-F-G and T-U-V-W-X-Y-Z, derived from the same user program would be issued in sequence by a global task sequencer to two cores, in a 2-way multiscalar CMP. 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. In this paradigm also, *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, it is possible to achieve better overall scalability of power-performance efficiency.

AUTHOR BIOS

David Brooks is currently pursuing a Ph.D degree in Electrical Engineering at Princeton University. He is spending time as a coop student at IBM Watson during the second half of 2000. His research interests include architecture-level power-performance modeling and the definition of power-efficient, high performance microarchitectures. He received his BSEE from the University of Southern California and his MA degree in Electrical Engineering from Princeton.

Pradip Bose is a Research Staff Member at IBM T. J. Watson Research Center. He received his M.S. and Ph.D degrees in Electrical & Computer Engineering from University of Illinois, Urbana-Champaign in 1981 and 1983 respectively. His research interests include computer architecture, power-aware design, systems modeling and validation. He is a senior member of IEEE.

Stanley Schuster is a Research Staff Member at IBM Watson, working in the area of low power, high speed digital circuits. He received his B.S. and M.S. Degrees in Electrical Engineering from New York University in 1962 and 1969 respectively. He joined IBM in 1965 and to date, he has 38 issued patents and over 40 published papers. He is a Fellow of the IEEE and a member of the IBM Academy of Technology.

Hans Jacobson received his M.Sc. degree from Lulea University, Sweden in 1996 and is currently pursuing a doctoral degree in computer science at University of Utah, USA. Among his research interests are asynchronous circuits and systems, and SMT microarchitectures for high-speed processors. He spent the summer of 2000 as an intern at IBM Watson.

Prabhakar Kudva received his Ph.D degree in Computer Science from the University of Utah. Since January 1995, he has been with the logic synthesis group at IBM T. J. Watson Research Center, where he currently works on synthesis and physical design. His work has contributed to the development of a recent server-class System/390 microprocessor; and for this he received an IBM Research Division Award.

Alper Buyuktosunoglu is a Ph.D student at the University of Rochester, NY. He spent the summer of 2000 as an intern at IBM T. J. Watson Research Center. He received his B.S. In Electrical and Electronics Engineering from Middle East Technical University, Ankara, Turkey in 1998 and his M.Sc in Electrical and Computer Engineering from University of Rochester, NY in 1999. His research interests are in complexity-adaptive processors and power-aware design at the microarchitecture and circuit levels.

John-David Wellman is a research staff member at the IBM T. J. Watson Research Center in Yorktown Heights, New York. He received a BSE, MSE and PhD in computer science and

engineering from the University of Michigan. He is a member of the IEEE. His research interests are in computer architecture, performance modeling and verification.

Victor Zyuban received his BS and MS degrees from the Moscow Institute of Physics and Technology, and a Ph.D degree from Notre Dame University. He is currently a Research Staff Member at IBM Watson. He is working on a project that involves the design of low-power DSP and embedded processors. His research interests are in low power circuit design, microarchitecture and methodologies for power-aware design.

Manish Gupta is a Research Staff Member and Manager of the High Performance Programming Environments group at IBM T. J. Watson Research Center. He is currently leading a research effort to investigate new compilation models and optimizations for Java on servers and embedded systems. His research interests include optimizing compilers, high performance computer architectures and object-oriented design.

Peter Cook is a Research Staff Member and Manager of High Performance Systems within the VLSI Design and Architecture Department at IBM Watson. Dr. Cook was educated at the University of Cincinnati (EE 1962) and did his graduate work at Carnegie-Mellon University (MS 1968, PhD 1971). He joined the staff of the IBM Thomas J. Watson Research Center in late 1965. He has been involved in various circuit and tools related aspects of VLSI design for essentially all of his career, including CMOS circuit design, and tools for layout, clock network generation, and techniques for post fabrication personalization of semiconductor chips.