Reclaiming Performance and Energy Efficiency from Variability

Sebastian Herbert, Siddharth Garg, and Diana Marculescu
Department of Electrical and Computer Engineering
Carnegie Mellon University
{sherbert, sgarg1, dianam}@ece.cmu.edu

***Abstract*— This paper presents methods for addressing two sources of variability in the context of microprocessors - within-die process variability and dynamic thermal variability - and shows the improvements in performance and energy efficiency obtained by applying them to a globally asynchronous, locally synchronous (GALS) microprocessor design. The GALS design style partitions the core into several independently-clocked domains, which provides a natural granularity for variability to be addressed at. Process variability is addressed by observing that each domain has fewer critical paths than the processor as a whole, shifting their maximum frequency distributions towards higher speeds. Meanwhile, the detrimental effects of thermal variability are reduced by isolating the effects of thermal hotspots on delay to the domains they occur in. We simulate a subset of the SPEC2000 benchmarks, comparing the baseline GALS architecture against a version that accounts for the reduction in the number of critical paths per domain, a version that scales the clock speed of each domain based on the slack between its temperature and the maximum operating temperature of the core, and a final version that does both. These schemes achieve improvements of 2.0%, 9.5%, and 11.4% in execution time and 4.4%, 16.0%, and 20.0% in energy-delay^2, respectively.**

I. Introduction

Variability is becoming a key concern for microarchitects as technology scaling continues and more and more increasingly ill-defined transistors are placed on a single die. Both process and thermal variations result in a nonuniformity of transistor delays across a single die. As both the amount of variation and the number of critical paths increase, the clock speed must be set lower and lower to reduce the probability of a timing violation to an acceptably small level. However, the worst-case delay is very rarely exercised and as a result the overdesign that is necessary to deal with variability sacrifices large amounts of performance in the common case. Bowman et al. found that designs for the 50 nm technology node will lose an entire generation's worth of performance due to systematic within-die process variability alone [1].

A variability-aware architecture is able to recover some of this performance loss. We address two sources of variability. First, we attempt to reduce the magnification of the effects of process variability as a result of the existence of multiple critical paths in a microprocessor. Since every critical path in a synchronously-timed block must meet some delay constraint for the block as a whole to meet that constraint, the probability of meeting a given timing constraint t_{max} decreases with both the magnitude of variability and the number of critical paths. We leave this first factor alone and instead attack the second.

Second, we address dynamic thermal variability that manifests itself as hotspots across the surface of the microprocessor die. At typical operating temperatures transistor delay increases with temperature as a result of the effect of temperature on carrier mobility. Once again, an entirely synchronously timed block must be clocked such that the delay through its hottest part meets the timing constraint, even though cooler parts of the die could safely be run faster.

We choose the globally asynchronous, locally synchronous (GALS) [2], [3], [4] architecture as the design driver to which we apply our techniques. In a GALS microprocessor, the core is split into several domains which are clocked independently. Communication between the clock domains is accomplished through the use of synchronization circuitry. GALS design has been proposed as a method to reduce the amount of power dissipated by the clock network, as several smaller clock networks can be made to use less power than a single large one. GALS also reduces clock skew, since each local clock network can use shorter wires than a single global one.

Both of our techniques have the same basic approach: isolate the detrimental effects of some variation to the clock domain it occurs in, rather than allowing it to affect the entire processor. Addressing process variability, we observe that each domain in the GALS processor has fewer critical paths than the processor as a whole, which shifts the mean of the maximum frequency distribution for each domain towards higher speeds. Thus the domains in the GALS version can be clocked faster than the synchronous baseline to some degree. Looking at thermal variability, we see that in the synchronous case the entire core must be slowed down to accommodate the temperature-induced increase in delay through the hottest block. For the GALS case, the same is only true at the domain granularity. Thus the effects of a hotspot are isolated to the domain it is in and

do not require a global reduction in clock frequency.

The paper is organized as follows. Section II details our methods for addressing process variability while Section III does the same for thermal variability. Section IV presents our experimental methodology and Section V our results. Section VI concludes.

II. Addressing Process Variability

A. Approach

The impact of parameter variations has been extensively studied at the circuit and device levels. However, with the increasing impact of variability on design yield it has become essential to consider higher-level models for parameter variation. Bowman et al. introduced the generic critical path model with the aim of quantifying the impact of die-to-die and within-die variations on overall timing yield [1]. They showed that the impact of variability on combinational circuits can be captured using two parameters: the logic depth of the circuit n_{cp} and the number of independent critical paths in the circuit N_{cp}. They observed that WID variations tend to determine the mean of the worst-case delay distribution of a circuit, while D2D variability determines its variance. Their model was validated against microprocessors from two different technology nodes (250 nm and 130 nm) and achieved an error of under 3%.

We assume a balanced microprocessor design with equal n_{cp} across stages, so we focus on N_{cp} for the rest of this work. According to the generic critical path model, if $F_{WID-T_{cp,nom}}(t_{max})$ is the cumulative distribution function of a *single* path delay due to WID variations with corresponding probability density function $f_{WID-T_{cp,nom}}(t_{max})$, the PDF of the worst-case delay for a circuit with N_{cp} independent critical paths will be given by:

$$f_{WID}(t_{max}) = N_{cp} f_{WID-T_{cp,nom}}(t_{max}) \times (F_{WID-T_{cp,nom}}(t_{max}))^{N_{cp}-1} \quad (1)$$

In Figure 1 we plot the worst-case delay distributions for $N_{cp} = (1, 2, 10)$. As N_{cp} increases the standard deviation of the worst-case delay distribution decreases while its mean increases. We observe that each of the five clock domains in the GALS partitioning has fewer critical paths than the microprocessor as a whole. As a result, the mean of the FMAX distribution for each clock domain occurs at a higher frequency than the mean of the baseline FMAX distribution.

Unfortunately, determining the number of *independent* critical paths in a given circuit is not trivial. Correlations between critical path delays occur due to inherent spatial correlations in parameter variations and, more fundamentally, due to the overlap of critical paths that pass through one or more of the same gates. Instead of deriving a new analytical framework to deal with correlated critical paths, we redefine N_{cp} to be the *effective* number of independent critical paths that, when plugged into Equation 1, will yield a worst-case delay distribution that matches the statistics of the actual worst-case delay distribution of the circuit. We developed a new methodology to estimate the effective number of independent critical paths for the two kinds of circuits that occur most frequently in processor microarchitectures - combinational logic and array structures. This work improves on the assumptions about the distribution of independent critical paths that have been made in previous studies. For example, Marculescu and Talpes assumed 100 total independent critical paths in a microprocessor and distributed them among blocks proportionally to device count [5], while Humaney et al. assumed that logic stages have only a single critical path [6]. We currently use a variability model that accounts for random variation in the gate length, L_{eff}, modeling it as a normal random variable with mean μ_L and standard deviation σ_L.

B. Combinational Logic Variability Modeling

Determining the effective number of critical paths for combinational logic is fairly straightforward. Following the generic critical path model [1], we use the SIS environment [7] to map uncommitted logic to a technology library of two-input NAND gates with a maximum fan-out of three. Gate delays are assumed to be independent normal random variables with mean equal to the nominal delay of the gate d_{nom} and standard deviation $\frac{\sigma_L}{\mu_L} d_{nom}$. We use Monte Carlo sampling to obtain the worst-case delay distribution for a given circuit. Moment matching is then used to determine the value of N_{cp} that, when plugged into Equation 1, will yield a distribution similar to the one obtained via Monte Carlo sampling. Since Equation 1 has only one

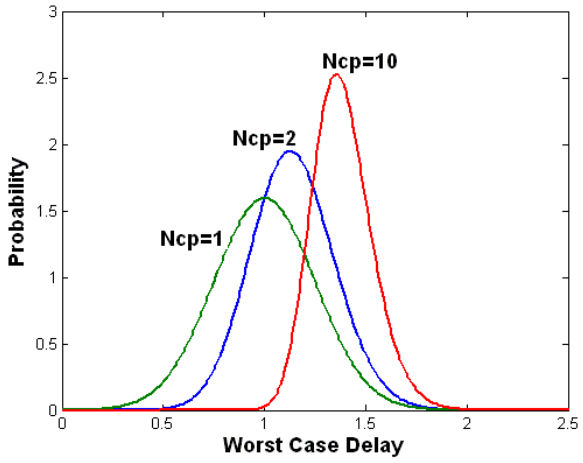


Fig. 1. Delay distributions for $N_{cp} = (1, 2, 10)$. The x-axis is normalized with respect to the nominal path delay of the circuit.

Circuit	Critical Paths	% Error in standard deviation
C432	4	35%
C499	11	22%
C880	4	33%
C2670	5	28%
C6288	1.2	7%

TABLE I

ESTIMATED EFFECTIVE NUMBER OF CRITICAL PATHS AND ERROR FOR ISCAS'85 BENCHMARK CIRCUITS.

unknown, we chose to match the first moments (means) of the two distributions. We evaluated our methodology over a range of circuits in the ISCAS'85 benchmark suite and observed that the obtained effective critical path numbers yield distributions that are reasonably close to the actual worst-case delay distributions, as seen in Table I. Note that the difference in the means of the two distributions will always be zero since we explicitly match them. Although the error in standard deviation can be as high as 35%, we expect the error to be much lower when considering the combined effect of D2D and WID variations. This is because, as mentioned before, the mean of the overall worst-case delay distribution is primarily determined by the mean of the WID distribution, which we match exactly, while the overall standard deviation is primarily determined by the standard deviation of the D2D distribution. These results can be used to directly assign critical path numbers to the functional units - the critical path numbers for ALUs and multipliers are taken from the C2670 and the C6288 circuits, respectively. We note that pipelining typically causes the number of critical paths in a circuit to be multiplied by the number of pipeline stages, as each critical path in the original implementation will now be critical in each of the pipeline stages as well. We therefore multiply the functional unit critical path numbers by their respective pipeline depths to estimate the impact of pipelining.

C. Array Structure Variability Modeling

Array structures are incompatible with the generic critical path model since they cannot be represented as combinational circuits composed of two-input NAND gates with a maximum fan-out of three. Since they constitute a large percentage of die area, it is essential to model the effect of WID variability on their access times accurately. One solution would be to simulate the impact of WID variability in a SPICE-level model of an SRAM array, but this would be prohibitively time-consuming. Instead we chose to enhance an existing high-level cache access time simulator, CACTI [8]. CACTI has been shown to accurately estimate access times to within 6% of HSPICE values while significantly reducing runtime.

To model the access time of an array, CACTI replaces the transistors and wires in a cache with an equivalent RC network. Figure 2 shows an inverter driving a load capacitance C_L and the equivalent RC model. Assuming

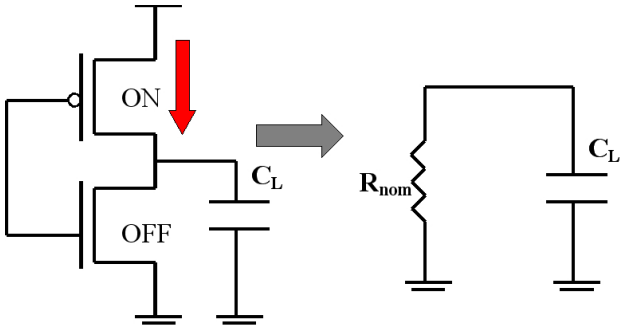


Fig. 2. An inverter with a rising output modeled as an RC network. R_{nom} corresponds to the equivalent nominal on-resistance of the PMOS transistor.

that we are interested in the delay of a rising edge at the output of the inverter, the resistance R_{nom} is the full on-resistance of the PMOS pull-up transistor. Since the on-resistance of a transistor is directly proportional to its effective gate length L_{eff}, which is modeled as a normal distribution $N(\mu_L, \sigma_L)$, we can represent R as a normal distribution $N(R_{nom}, \frac{\sigma_L}{\mu_L} R_{nom})$. To determine the inverter delay, CACTI uses the first-order time constant of the network t_f, which can be written as $t_f = R \cdot C_L$, and the Horowitz model [9]:

$$delay = t_f \cdot \sqrt{\alpha + \frac{\beta}{t_f}} \tag{2}$$

α and β are functions of the threshold voltage, supply voltage, and the input rise time, which we assume to be constant in our analysis. The delay is a weakly nonlinear (and therefore strongly linear) function of t_f, which in turn is a linear function of R. Each stage delay in the RC network can therefore be modeled as a normal random variable.

This analysis holds true for all stages in an array structure except the comparator and bitline stages, for which CACTI uses a second-order RC model. However, under the assumption that the input rise time is fast, we can approximate these stage delays as normal random variables as well.

Until now we have only considered variability in the on-resistance of the transistors due to gate length variations. Since the wire-delay contribution to overall delay is increasing as technology scales, we also model random variations in the wire dimensions. CACTI lumps the entire resistance and capacitance of a wire of length L into a single resistance $L * R_{wire}$ and a single capacitance $L * C_{wire}$, where R_{wire} and C_{wire} represent the resistance and capacitance of a wire of unit length. Variations in the wire dimensions translate into variations in the wire resistance and capacitance. We assume that R_{wire} and C_{wire} are *independent* normal random variables with standard deviation σ_{wire}. This assumption is not unreasonable, since the only physical parameter that affects both R_{wire} and

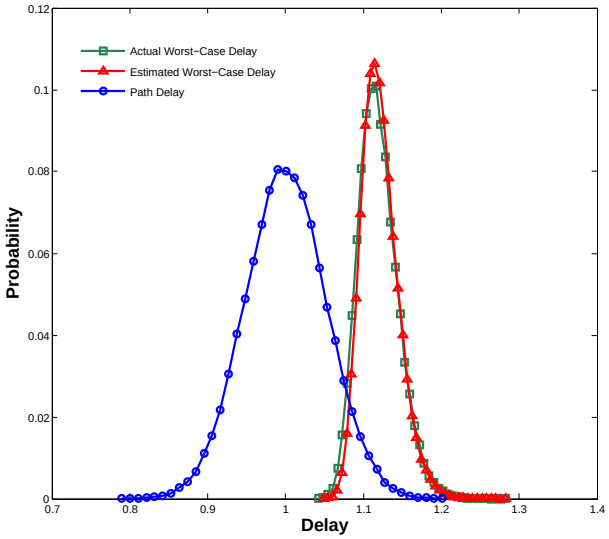


Fig. 3. Estimated versus actual worst-case delay distribution for a 1 KB direct-mapped cache with 32 B blocks. The x-axis is normalized to the nominal delay of the cache.

Array Size	Wordlines	Bitlines	Critical Paths
256 B	32	64	47
512 B	64	64	115
1024 B	128	64	225
2048 B	256	64	345

TABLE II

EFFECTIVE NUMBER OF CRITICAL PATHS FOR ARRAY STRUCTURES

C_{wire} is wire width, which typically has the least impact on wire delay variability [10]. Furthermore, we model variability across the length of a single wire and between wires. We assume that a wire of length L is composed of N segments, each of which has its own R_{wire} and C_{wire}. The standard deviation of the lumped resistance and capacitance of a wire of length L is therefore $\frac{\sigma_{wire}}{\sqrt{N}}$. The length of each segment is assumed to be the feature size of the technology in which the array structure is implemented, since this is the minimum distance across which a change in the physical parameters of a wire can be expected.

We implemented these variability models within CACTI and obtained the delay distributions of each stage along the critical path of an array access and the overall path delay distribution for the array. Monte Carlo sampling was used to obtain the worst-case delay distribution for the array from the observed stage delay distributions and the effective number of critical paths was then computed through moment matching. This is highly accurate - in most cases we observe that the estimated and actual worst-case delay distributions are nearly indistinguishable, as seen in Figure 3 for a 1 KB direct-mapped cache with a 32 B block size. In Table II we show some effective independent critical path numbers obtained with our model. As expected, larger caches have a greater number of effective critical paths. Furthermore, we can see that due to their regular structure caches typically have more critical paths than the combinational circuits evaluated previously. Humaney et al. reached a similar conclusion while comparing datapaths with memory arrays [6]. However, their analysis assumed that the number of critical paths in an array was equal to the number of bitlines. We provide an enhanced model that accounts for all sources of variability, including the wordlines, bitlines, decoders, and output drivers.

D. Application to the GALS Processor

We applied these critical path estimation methods to a microarchitecture similar in implementation to that in [5]. We assume a balanced microarchitecture with the same logic depth n_{cp} across stages. Table III details the effective number of independent critical paths in each domain. We then used these values of N_{cp} in Equation 1, which yields the probability density functions and cumulative distribution functions plotted in Figure 4.

We found the mean value of each distribution and then calculated the resulting speedups for the five GALS domains relative to the synchronous baseline as

$$speedup_{cp} = \frac{T_{cp,nom} + \mu_{\Delta t_{max,synchronous}}}{T_{cp,nom} + \mu_{\Delta t_{max,domain}}} \quad (3)$$

Results are shown in Table IV, assuming a path delay standard deviation of 5% for our 130 nm process. This is slightly higher than the 4.49% found for the 180 nm node using an analytical method by Agarwal et al. [11]. The fully sychronous baseline incurs an 18.5% higher mean delay as a result of having 6188 critical paths rather than only one. The GALS domains are only penalized by an average of 14.3%. We present simulation results obtained by applying these speedups in Section V.

Domain	Number of critical paths
Fetch/Decode	3070
Rename/Retire/Read	384
Integer	61
Floating Point	44
Memory	2629
Total	6188

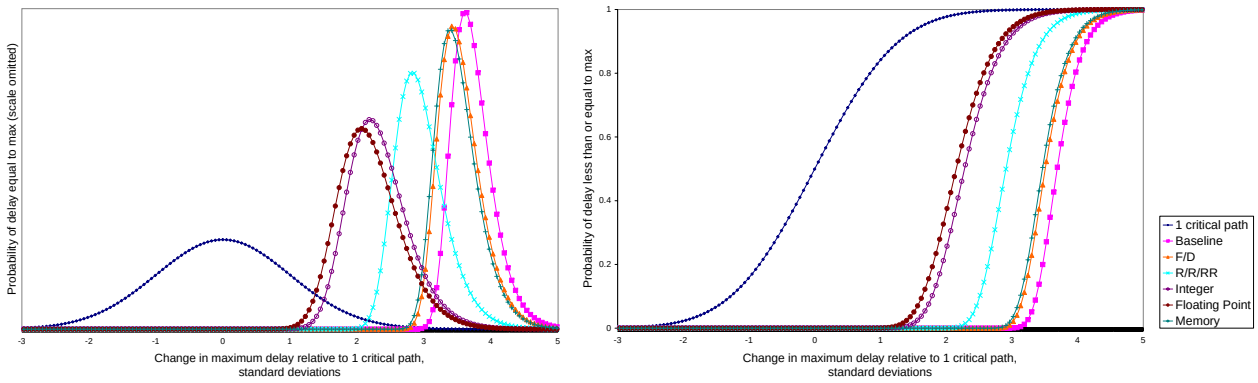
TABLE III

NUMBER OF CRITICAL PATHS IN EACH GALS DOMAIN

Domain	$T_{cp,nom} + \mu_{\Delta t_{max}}$	Speedup
Baseline	1.1845	1.000
Fetch	1.1754	1.0077
Rename/Retire/Register Read	1.1455	1.0340
Integer	1.1140	1.0633
Floating Point	1.1077	1.0694
Memory	1.1733	1.0095
Average over GALS domains	1.1432	1.0368

TABLE IV

SPEEDUPS OBTAINED WITH CRITICAL PATH INFORMATION



(a) PDF $f_{WID-\Delta T}$

(b) CDF $F_{WID-\Delta T}$

Fig. 4. PDFs and CDFs for ΔT_{WID}

III. Addressing Thermal Variability

At runtime there is dynamic variation in temperature across the die surface, which results in a further nonuniformity of transistor delays across the die. Some units, such as caches, tend to be cool while others, such as register files and ALUs, may run much hotter. Temperature affects delay in several different ways. We model two of these in particular. First, delay is inversely proportional to carrier mobility, μ:

$$d \propto \frac{1}{\mu} \tag{4}$$

We use the fitted model found in [12] to account for the effects of temperature on carrier mobility:

$$\mu = \frac{a}{T^b} \tag{5}$$

a and b are empirically determined constants. We use $a = 1.06 \times 10^9$ and $b = 2.49$ [12]. However, the value of a falls out of the equation we use for scaling frequency.

Temperature also affects delay indirectly through its effect on threshold voltage. Delay, supply voltage, and threshold voltage are related by the well-known alpha power law [13]:

$$d \propto \frac{V_{DD}}{(V_{DD} - V_{TH})^\alpha} \tag{6}$$

We use a value of 1.2 for α, the velocity saturation index, as given in [14]. The threshold voltage itself is dependent on temperature. We use the model from [15]:

$$V_{TH} = V_{TH,0} - k(T - T_0) \tag{7}$$

k is the threshold voltage temperature coefficient. We use k = 0.7 mV/K for our 130 nm process, as in [15]. Combining the effects of temperature on carrier mobility and threshold voltage, we obtain

$$d \propto \frac{T^b}{a} \frac{V_{DD}}{(V_{DD} - V_{TH,0} + k(T - T_0))^\alpha} \tag{8}$$

Frequency is inversely proportional to delay, so we introduce a proportionality constant C, lump the a term into it, and write

$$f = C \frac{1}{T^b} \frac{(V_{DD} - V_{TH,0} + k(T - T_0))^\alpha}{V_{DD}} \tag{9}$$

We simulate a processor similar to that in [16], so we choose C such that our processor will run at 3.0 GHz with V_{DD} = 1.3 V and V_{TH} = 0.212 V at a temperature of 100 °C. Essentially, the maximum operating temperature for our processor as specified by the manufacturer would be 100 °C. Normal operating temperatures will often be below this ceiling and we exploit this thermal slack by speeding up cooler domains.

IV. Experimental Setup

A. Baseline Simulator

We used a modified version of the SimpleScalar simulator [17] with the Wattch power estimation extensions [18] and HotSpot thermal simulation package [16]. We reorganized the microarchitecture to resemble an Alpha microprocessor, with the backend divided into integer, floating point, and memory clusters, each with their own instruction windows and issue logic, and separate instruction and data TLBs. We used the HotSpot floorplan from [16], which models an Alpha 21364-like core shrunken to 130 nm technology while keeping die area constant (the resulting excess area is used for additional L2 cache). The on-chip multiprocessor interface logic is likewise replaced with L2 cache. The processor parameters are summarized in Table V. We enhanced Wattch by splitting structures such as register files and map tables into integer and floating point units, where previously the two had been lumped together, allowing us to more accurately track the power consumption of the different clock domains.

B. Static Power Model

We added a static power model based on that proposed by Butts and Sohi [19] to complement Wattch's dynamic power model. We estimate the number of SRAM cells,

Parameter	Value
Frequency	3.0 GHz with a ∼3 MHz step size
Technology	130 nm node with V_{DD} = 1.3 V and V_{TH} = 0.212 V
L1-I/D cache configuration	64 KB, 64 B blocks, 2-way set associative, 2-cycle hit time, LRU replacement
L2 cache configuration	4 MB, 128 B blocks, 8-way set associative, 12-cycle hit time, LRU replacement
TLB configuration (both TLBs)	128 entries, fully associative, 30-cycle miss latency, LRU replacement
Pipeline configuration	16 stages deep, 4 instructions wide
Window sizes	32 integer, 16 floating point, 16 memory
Load/Store queue size	64
Integer functional units	4 ALUs, 1 MUL/DIV
Floating point functional units	2 adders, 1 MUL/DIV
Memory access ports	2
Memory access time	120 ns for a random access, 4 ns for subsequent consecutive accesses
Branch predictor	gshare, 12 bits of history, 4096-entry table

TABLE V
PROCESSOR PARAMETERS

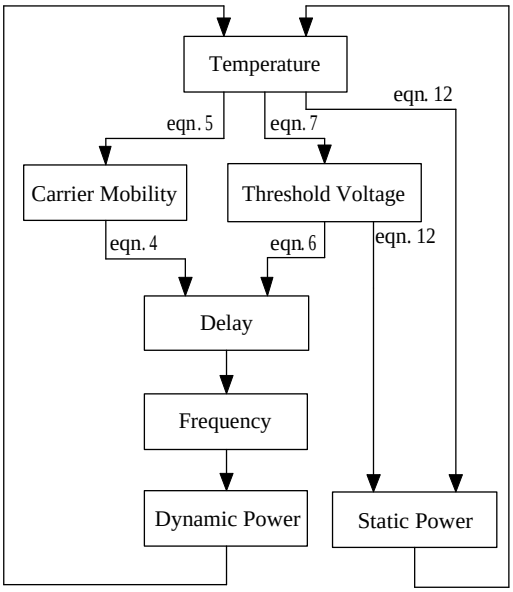


Fig. 5. Dependencies modeled

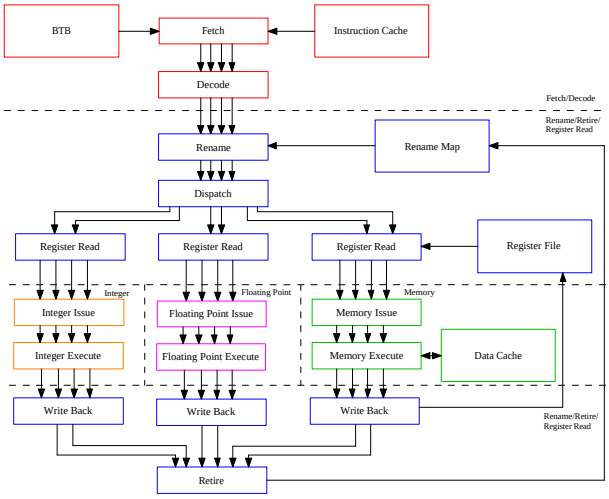


Fig. 6. Microarchitecture with GALS partitioning

CAM cells, D flip-flops, D latches, and random logic gates in each block that is tracked by Wattch using the same methodology as [5]. When power statistics are updated every cycle, we use the same access counters used by Wattch to determine if a block was unused and, if so, compute the leakage power dissipated. We also model the effect of temperature on leakage power. The leakage current model of [19] accounts for the dependence of leakage current on both temperature and threshold voltage:

$$I_{Dsub} = k \cdot e^{\frac{-q \cdot V_{TH}}{a \cdot k_B \cdot T}} \tag{10}$$

T is the absolute temperature, q and k_B are physical constants, and a and k are device parameters. We lump together q, k_B, and a into a single constant term c:

$$I_{Dsub} = k \cdot e^{\frac{c \cdot V_{TH}}{T}} \tag{11}$$

Since threshold voltage is directly dependent upon temperature as well, as given by equation 7, leakage current actually depends on temperature both directly and indirectly. Thus our final model for scaling leakage current based on temperature is:

$$I_{Dsub} = k \cdot e^{\frac{c(V_{TH,0} - k(T - T_0))}{T}} \tag{12}$$

We extracted the values of the constants k and c from Figure 5 in [19] and calculated a baseline per-device leakage current of 13.14 nA at 100 °C. We call HotSpot to update chip temperatures every 20,000 cycles. We then compute a leakage scaling factor for each block (at the same granularity used by Wattch) and use it to scale the leakage power computed every cycle until the next temperature update. Figure 5 shows all of the dependencies that we take into account in our various simulations and references the equations that govern them.

C. GALS Simulator

We also created a GALS simulator. It is split into five clock domains: fetch/decode, rename/retire/register read, integer, floating point, and memory. Figure 6 shows the microarchitecture with clock domain boundaries superimposed. Each domain has a power model for its clock signal

that is based on the number of pipeline registers within the domain. Inter-domain communication is accomplished through the use of asynchronous FIFO queues [20], as these offer improved throughput over many other synchronization schemes under nominal FIFO operation. As long as the FIFO is neither empty nor full, the cell to be used for the next read differs from that to be used for the next write, allowing the two operations to proceed in parallel. As a result, an item can be enqueued every producer clock cycle and an item can be dequeued every consumer clock cycle as long as the FIFO does not become full or empty.

We created several versions of the GALS simulator. The first is the baseline version (GALS-B), which splits the core into multiple clock domains but runs each one at the same speed as the synchronous baseline (SYNCH). The second speeds up each domain as a result of the individual domains having fewer critical paths than the microprocessor as a whole. The speedups are taken from Table IV and this version is called GALS-CP. In the interests of reducing simulation time, we simulate only the mean speedups. These represent the average benefit that a GALS processor would display over an equivalent synchronous processor over the fabrication of a large number of dies. The third version, GALS-T, assigns each domain a baseline frequency that is equal to the synchronous baseline's but then scales each domain's frequency for its temperature according to Equation 9 after every chip temperature update (every 20,000 cycles). A final version, GALS-CP-T, uses the speeds from GALS-CP as the baseline domain speeds and then applies our thermally-aware frequency scaling. Both GALS-T and GALS-CP-T perform dynamic frequency scaling, for which we assume an aggressive Intel Xscale-style DFS system as in [21] and [22], with a single frequency step being approximately 3 MHz.

D. Benchmarks Simulated

The feedback effects from the dependencies in Figure 5 had the effect of requiring multiple simulation runs for each benchmark and configuration, feeding the output steady-state temperatures of one run back in as the initial temperatures of the next in search of a consistent operating point. We iterated until temperature, power, and performance values converged and then recorded the results of three additional runs as our reported statistics.

Benchmark	Arguments
164.gzip	input.source
175.vpr	net.in arch.in place.in -nodisp -place_only -init_t 5 -exit_t 0.005 -alpha_t 0.9412 -inner_num 2
197.parser	2.1.dict < ref.in
177.mesa	-frames 1000 -meshfile mesa.in -ppmfile mesa.ppm
183.equake	< inp.in
188.ammp	< bptipg.amp

TABLE VI
BENCHMARKS USED

The large number of simulation runs required per benchmark prevented us from simulating the entire suite of SPEC2000 benchmarks [23] due to time constraints. We were limited to using six of the benchmarks: the *164.gzip*, *175.vpr*, and *197.parser* integer benchmarks and the *177.mesa*, *183.equake*, and *188.ammp* floating point benchmarks. Table VI shows the command-line parameters used for each.

E. Simulation Variability

We addressed time variability by simulating three points within each benchmark, starting at 500, 750, and 1000 million instructions and gathering statistics for 50 million more. The one exception was *188.ammp*, which finished too early. Instead, the cycle-gathering phases were started at 200, 300, and 400 million instructions. Because the GALS microprocessor is globally asynchronous, space variability is also an issue (for example, the exact order in which domains tick could have a significant effect on branch prediction performance as the arrival time of prediction feedback will be altered). Our simulator randomly assigns phases to the domain clocks, which introduces slight perturbations into the ordering of events and so averages out possible extreme cases over the three runs per simulation point per benchmark. Both types of variability were thus addressed using the approaches suggested by [24].

V. RESULTS

A. Overview

The GALS configurations are compared on execution time, average power, total energy, and energy-delay^2 in Figure 8. Note that the scale for the energy-delay^2 graph differs from that used on the first three due to the poor performance of *164.gzip* on the GALS architecture. Tables VII and VIII show the average speedups obtained for each benchmark

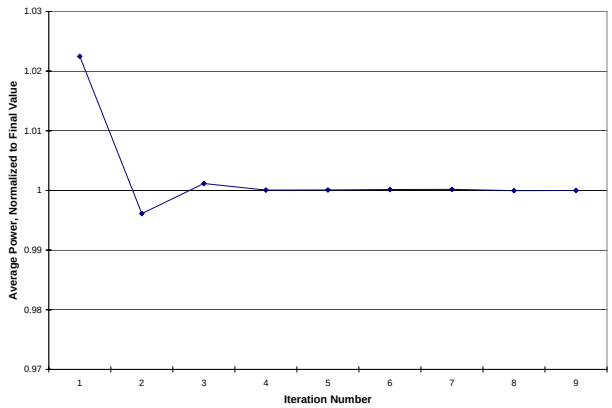
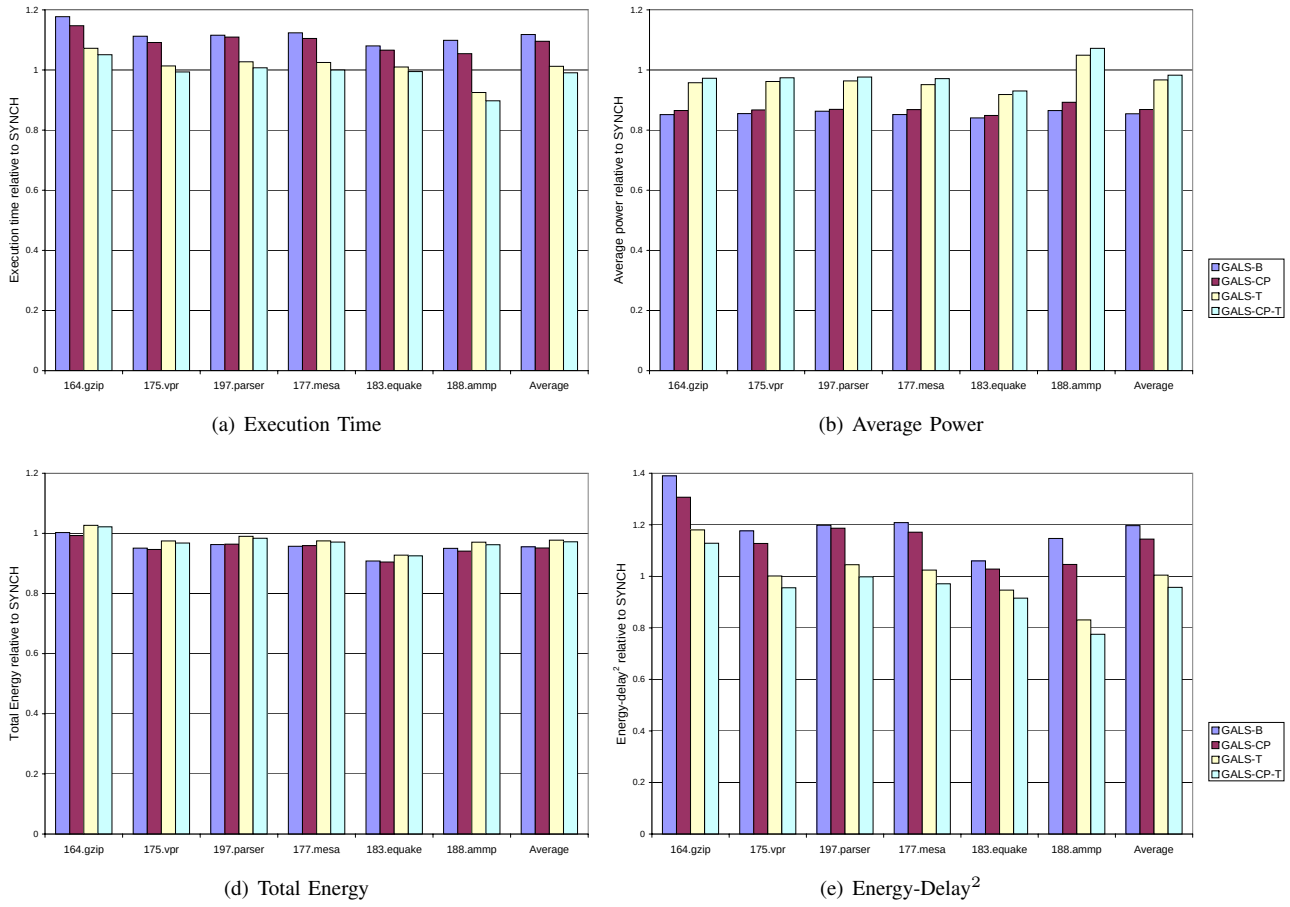


Fig. 7. Average Power vs. Iteration for *175.vpr* under GALS-T (starting at 500 million instructions)



(a) Execution Time

(b) Average Power

(d) Total Energy

(e) Energy-Delay^2

Fig. 8. Simulation results, all relative to synchronous baseline

under GALS-T and GALS-CP-T. These speedups are self-consistent - that is, starting a run with these speedups and the corresponding temperatures will yield the same temperatures as steady-state temperatures in the result. Figure 7 illustrates the convergence achieved with a sample plot of average power versus iteration number for *175.vpr*. The other benchmarks displayed similar behavior.

B. GALS-B

Moving from a fully synchronous design to a GALS one (GALS-B) incurs an average 11.8% penalty in execution time, somewhat higher than observed in previous GALS studies [3], [4]. This is primarily an effect of our larger caches, which reduce the probability that some delay due to GALS will be hidden behind a stall that occurred even in the synchronous case, increasing the observed performance penalty. Five out of the six benchmarks display a similar performance degradation of roughly 10%, while *164.gzip* does more poorly in adapting to the GALS architecture and runs over 17% slower. Due to the use of small local clock networks and the stretching of execution time, the GALS processor draws 14.6% less power per cycle, resulting in a consumption of 4.5% less energy than the synchronous baseline over the execution of the same instructions. Energy-delay^2 is increased by 19.7% in making the move to the baseline GALS architecture, making it uncompetitive in most applications.

C. GALS-CP

Despite the average per-domain speedup in GALS-CP being 3.68%, execution time decreases by only 2.0% because of the mismatch between speedups. The fetch and memory domains are barely sped up at all as a result of the large number of critical paths in the level 1 caches. This decreases the average number of dispatched instructions per clock tick for each back-end domain because 1) instructions are entering their window at a relatively reduced rate due to the low instruction cache speedup and 2) load-dependent instructions must wait relatively longer for operands due to the low data cache speedup. As a result of the faster clocking of domains, the average power drawn per cycle increases very slightly when enabling the -CP speedups (by about 1.6%). However, the faster execution leads to essentially no change in energy usage (actually an extremely marginal decrease of 0.4%) and an overall energy-delay^2 reduction of 4.4%.

We observe that GALS-CP suffers from the domain parti-

	F/D	R/R	Int	FP	Mem	Avg
164.gzip	1.15	1.11	1.10	1.23	1.16	1.15
175.vpr	1.14	1.12	1.11	1.23	1.18	1.16
197.parser	1.14	1.13	1.11	1.23	1.18	1.16
177.mesa	1.17	1.13	1.11	1.20	1.15	1.15
183.equake	1.16	1.12	1.10	1.23	1.16	1.15
188.ammp	1.24	1.23	1.22	1.25	1.24	1.24
Average	1.17	1.14	1.12	1.23	1.18	1.17

TABLE VII
PER-DOMAIN SPEEDUPS OBTAINED WITH GALS-T

	F/D	R/R	Int	FP	Mem	Avg
164.gzip	1.15	1.14	1.16	1.31	1.17	1.19
175.vpr	1.14	1.16	1.18	1.31	1.18	1.19
197.parser	1.15	1.17	1.18	1.31	1.18	1.20
177.mesa	1.17	1.16	1.17	1.28	1.15	1.18
183.equake	1.17	1.16	1.16	1.31	1.17	1.19
188.ammp	1.25	1.27	1.29	1.33	1.25	1.28
Average	1.17	1.18	1.19	1.31	1.18	1.21

TABLE VIII
PER-DOMAIN SPEEDUPS OBTAINED WITH GALS-CP-T

tioning used, which is performed based on the actual functionality of blocks without taking into account the number of critical paths that they contain. A better partitioning would use some metric that relates the number of critical paths in a block to its criticality to performance. However, "criticality to performance" can be difficult to quantify, since the critical path through the core will be different for different types of applications. Moreover, there is overhead associated with every domain and domain boundary crossing. Combining domains can reduce the required number of domain boundary crossings as well as design complexity but will also reduce the power savings introduced by the GALS clocking scheme (since we are merging some small clock networks to create a single larger one). Furthermore, it reduces the flexibility of the GALS architecture and might impact other schemes such as our thermally-aware frequency scaling or dynamic voltage/frequency scaling. On the other hand, splitting a clock domain into multiple smaller domains requires the opposite set of tradeoffs to be evaluated.

D. GALS-T

GALS-T offers significantly better performance than GALS-B or GALS-CP, with an average execution time reduction of 9.5%. For *188.ammp*, which runs cool due to its low IPC, thermally-aware frequency scaling actually results in better performance than the synchronous baseline. On the other hand, *164.gzip* still lags in performance. Overall, GALS-T begins to approach the performance of the synchronous baseline, with an average execution time penalty of only 1.0%. Since the speedups in GALS-T are greater than those in GALS-CP, as seen in Table VII, a more significant average power penalty is observed when applying this scheme to the baseline GALS architecture at 13.2%. This includes increases in dynamic power due to the higher clock frequencies as well as increases in leakage power due to the elevated temperature arising from higher power density. However, as a result of the faster execution, the total energy penalty is much smaller at an average of 2.3%, resulting in a 16.0% reduction in energy-delay^2 relative to GALS-B and a 1.0% reduction relative to the synchronous baseline.

GALS-T suffers somewhat from naïvely speeding up domains whether this improves performance or not. The most egregious example is the speeding up of the floating point domain by 23% in the integer benchmarks. This may even adversely affect performance because each clock tick dissipates some power, regardless of whether there are any instructions in the domain or not. This results in higher local temperatures, which may spill over into a neighboring domain which is critical to performance and cause it to be clocked at a lower speeds. One possible solution is to use some control scheme similar to those used for DVFS to decide whether a domain should actually be sped up. Since Equation 9, which we use for scaling frequency with temperature, also includes the dependence of frequency on supply voltage, one could even combine the two. An integrated control system would be required to prevent the two schemes from pulling clock frequency in opposite directions. This is an area that will require further research. We have done some preliminary experiments with GALS-T and DVFS which suggest that such a scheme may be fairly complicated. Like GALS-CP, GALS-T could also benefit from a more intelligent domain partitioning. Since each domain's speed is limited by its hottest block, it might make sense to group blocks into domains based on whether they tend to run cool, hot, or in-between. However, while there are some functional blocks which can be identified as generally being hotspots (e.g. the integer register file and scheduling logic) the temperature at which some other blocks run is highly workload-dependent (e.g. the entire floating point unit).

E. GALS-CP-T

The results for GALS-CP-T show that the two schemes are very largely additive. An 11.4% reduction in execution time is achieved at the cost of 15.0% higher average power. The total energy penalty is 1.7%. This is lower than that of GALS-T, demonstrating once again that applying the -CP speedups reduces total energy consumption. The final energy-delay^2 improvement relative to GALS-B is 20.0%. An initial fear when combining GALS-CP and GALS-T was that the higher baseline speeds as a result of the -CP speedups would result in a sufficient increase in temperature to reduce the -T speedups by an equal amount, resulting in a scheme that offered no better performance than GALS-T and was more complex. However, our results show that the speedups applied by GALS-CP and GALS-T are largely independent.

Perhaps most importantly, we see that GALS-CP-T has caught up to the synchronous baseline on the execution time metric, displaying an average reduction of 0.9%. Of course, this is within the likely margin of error of our simulations and so does not provide conclusive evidence. However, we observe that once we exclude the two extreme cases in terms of relative execution time (*164.gzip* and *188.ammp*), the remaining benchmarks are all essentially even with the synchronous baseline (on average 0.001% faster). GALS-CP-T displays a larger advantage over the synchronous baseline on our other three metrics, with reductions of 1.7%, 2.8%, and 4.6% in average power, total energy, and energy-delay^2, respectively.

VI. Conclusion

Variability is one of the major concerns that microprocessor designers will have to face as technology scaling continues. It is potentially easier for a GALS design to address variability as a result of the processor being partitioned into smaller clock domains, which allows the negative effects of variability to be localized to the domain they occur in. We evaluated two applications of this approach, one to address process variability and one to address thermal variability. Both schemes improved performance and energy-delay^2 relative to the baseline GALS architecture, while applying them at the same time led to essentially the same performance as the synchronous baseline with somewhat better energy-efficiency.

Interestingly, the 130 nm technology node this work was conducted at seems to be the break-even point of GALS-CP-T. As technology continues to scale, these schemes will become more attractive as the magnitude of process variations grows (increasing $\sigma_{WID-T_{cp,nom}}$) and thermal variations worsen (higher difference in power densities and thus temperatures across the chip). Given the fact that current commercial microprocessor designs are at the 65 nm node, two generations ahead of the one used for this work, we can expect that the point were a significant benefit would be offered by a variability-aware GALS architecture will soon come if it has not, in fact, come already.

References

[1] K. Bowman, S. Duvall, and J. Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 2, Feb 2002.

[2] A. Iyer and D. Marculescu, "Power and performance evaluation of globally asynchronous locally synchronous processors," in *ISCA '03: Proceedings of the 29th annual International Symposium on Computer Architecture*, 2002, pp. 158–168.

[3] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, 2002, p. 29.

[4] E. Talpes and D. Marculescu, "Toward a multiple clock/voltage island design style for power-aware processors," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 13, no. 5, pp. 591–603, 2005.

[5] D. Marculescu and E. Talpes, "Variability and energy awareness: a microarchitecture-level perspective," in *DAC '05: Proceedings of the 42nd annual Design Automation Conference*, 2005, pp. 11–16.

[6] E. Humaney, D. Tarjan, and K. Skadron, "Impact of parameter variations on multi-core chips," in *ASGI '06: Proceedings of the 2006 Workshop on Architectural Support for Gigascale Integration*, 2006.

[7] E. Sentovich, "SIS: A system for sequential circuit synthesis," University of California, Berkeley, Tech. Rep., 1992.

[8] S. Wilton and N. Jouppi, "CACTI: an enhanced cache access and cycle time model," *IEEE Journal of Solid State Circuits*, vol. 31, no. 5, pp. 677–688, 1996.

[9] M. A. Horowitz, "Timing models for MOS circuits," Stanford University, Tech. Rep., 1983.

[10] M. Orshansky, C. Spanos, and C. Hu, "Circuit performance variability decomposition," in *Proceedings of the 4th International Workshop on Statistical Metrology*, 1999.

[11] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gal, and R. Panda, "Statistical delay computation considering spatial correlations," in *ASP-DAC '03: Proceedings of the 2003 Asia and South Pacific Design Automatic Conference*, Jan 2003, pp. 271–276.

[12] J. Batista, A. Mandelis, and D. Shaughnessy, "Temperature dependence of carrier mobility in Si wafers measured by infrared photocarrier radiometry," *Applied Physics Letters*, vol. 82, no. 23, pp. 4077–4079, June 2003.

[13] T. Sakurai and A. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 584–594, Apr 1990.

[14] K. Chen and C. Hu, "Performance and Vdd scaling in deep submicrometer CMOS," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 10, pp. 1586–1589, Oct 1998.

[15] A. Basu, S.-C. Lin, V. Wason, A. Mehrotra, and K. Banerjee, "Simultaneous optimization of supply and threshold voltages for low-power and high-performance circuits in the leakage dominant era," in *DAC '05: Proceedings of the 41st annual Design Automation Conference*, 2004, pp. 884–887.

[16] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *ISCA '03: Proceedings of the 30th annual International Symposium on Computer Architecture*, 2003.

[17] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," University of Wisconsin - Madison Computer Sciences Department, Tech. Rep. 1342, June 1997.

[18] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *ISCA '03: Proceedings of the 27th annual International Symposium on Computer Architecture*, June 2000.

[19] J. A. Butts and G. S. Sohi, "A static power model for architects," in *MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, 2000, pp. 191–201.

[20] T. Chelcea and S. Nowick, "Robust interfaces for mixed systems with application to latency-insensitive protocols," presented at DAC '01: The 38th annual Design Automation Conference, Jun 2001.

[21] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors," in *ASPLOS-XI: Proceedings of the 11th international conference on Architectural Support for Programming Languages and Operating Systems*, 2004, pp. 248–259.

[22] ——, "Voltage and frequency control with adaptive reaction time in multiple-clock-domain processors," in *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 2005, pp. 178–189.

[23] "SPEC CPU2000." [Online]. Available: http://www.spec.org/cpu2000/

[24] A. R. Alameldeen and D. A. Wood, "Variability in architectural simulations of multi-threaded workloads," in *HPCA '03: Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, 2003.