Power Efficiency for Variation-Tolerant Multicore Processors

James Donald and Margaret Martonosi
Department of Electrical Engineering
Princeton University
Princeton, NJ
{jdonald, mrm}@princeton.edu

ABSTRACT

Challenges in multicore processor design include meeting demands for performance, power, and reliability. The progression towards deep submicron process technologies entails increasing challenges of process variability resulting in timing instabilities and leakage power variation. This work introduces an analytical approach for ensuring timing reliability while meeting the appropriate performance and power demands in spite of process variation. We validate our analytical model using Turandot to simulate an 8-core PowerPC$^{\text{TM}}$processor. We first examine a simplified case of our model on a platform running independent multiprogrammed workloads consisting of all 26 of the SPEC 2000 benchmarks. Our simple model accurately predicts the cutoff point with a mean error less than 0.5 W. Next, we extend our analysis to parallel programming by incorporating Amdahl's Law in our equations. We use this relation to establish limit properties of power-performance for scaling parallel applications, and validate our findings using 8 applications from the SPLASH-2 benchmark suite.

Categories and Subject Descriptors

C.1 [**Processor Architectures**]: Parallel Architectures; C.4 [**Performance of Systems**]: Performance Attributes

General Terms

Performance

Keywords

multicore, variation, power, parallel applications

1. INTRODUCTION

Process variation is an ever-increasing challenge in microprocessor design. Deep submicron technologies pose significant risks for wider spread in timing paths, as well as variations in leakage power. Within a few technology generations, it is expected that within-die variations will become more significant than die-to-die variations [4], and manifest in multicore chips as core-to-core variations [14]. Architects must design these chips with appropriate options to ensure reliability while still meeting appropriate performance and power requirements. This involves fallback modes at the circuit, architectural, and system level.

Some post-silicon circuit techniques can be applied to ensure valid timing, but these entail non-trivial costs in terms of dynamic and leakage power. Our starting parameter is P_{excess}, the amount of excess power on a core resulting from inherent leakage variation or as a side effect of circuit techniques to ensure proper timing. For maximum reliability, two techniques that ensure accurate timing on process-variant cores are adaptive body bias (ABB) and V_{DD} adjustment [34]. ABB refers to toggling an additional voltage between the base and source. This allows various timing paths to be sped up at the cost of possibly significant leakage power increase. V_{DD} adjustment generally does not increase the leakage as much, but has repercussions on both leakage and dynamic power. Furthermore, these effects on core power occur on top of inherent variations in leakage power, which can vary significantly across cores [14].

Our work takes into account these power discrepancies to formulate policies for post-silicon adaptivity to meet desired performance and power budgets. Modern platforms often have overall goals or configurable power modes that focus on maximizing the ratio of *performance/watt* rather than performance alone. In studying methods toward this goal, our approach is to turn off cores that are particularly expensive in terms of power consumption. For this it is necessary to establish the appropriate optimal tradeoff points. We seek to quantify these cutoffs depending on the level of variation and execution characteristics of the applications. By giving the system knowledge of power variation traits through diagnostics, these bounds can be known or calculated at system runtime and then used to properly tune performance and power to sustain desired user demands.

Since chip multiprocessors are becoming a widespread basis for platforms in the server, desktop, and mobile sectors, we tailor our analysis toward multicore designs. Modern devices now run a wide variety of applications, often concurrently, and must efficiently operate depending on their tasks at hand. We first derive an appropriate performance/power tradeoff point for the case of running 8 programs simultaneously through multiprogramming. We then extend our analysis to parallel programs, since multicore designs have become a major motivation factor toward seeing widespread use of parallel applications in all sectors.

We propose that multicore-based systems can adapt readily to meet power-performance requirements. Specifically, the core power ratings may be identified at the time of system integration, and can even be reconfigured through system diagnostics after power ratings change due to long-term depreciation effects. With knowledge of these variations, the system can choose how to efficiently allocate cores to particular tasks and put cores to sleep if potential additional performance is not power-efficient. Our specific contributions are as follows:

- We derive an analytical bound for estimating the amount of tolerable process variation for multicore policies seeking to maximize *performance/watt.* These excess power cutoff values, ranging from 1 to 6 W, are used to decide when to turn off extra power-consuming cores.
- We introduce PTCMP, a fast multicore simulation environment, and use this to validate our analysis using workloads formed from the SPEC 2000 suite.
- We extend our equations to the problem of parallel programming by incorporating Amdahl's Law, and use the new derived relation to establish limit properties for power-efficient parallel scaling.
- Using extended features of PTCMP we demonstrate these properties using 8 applications from the SPLASH-2 benchmark suite. The high parallel efficiency of `raytrace`, for example, allows it to increase in performance/power ra-

Global Design Parameters	
Process Technology	$35nm$
Target Supply Voltage	0.9 V (selectively adjusted for variation)
Clock Rate	2.4 GHz
Organization	8-core, shared L2 cache
Core Configuration	
Reservation Stations	Int queue (2x20), FP queue (2x5), Mem queue (2x20)
Functional Units	2 FXU, 2 FPU, 2 LSU, 1 BXU
Physical Registers	80 GPR, 72 FPR, 60 SPR, 32 CCR
Branch Predictor	16K-entry bimodal, gshare, selector
Memory Hierarchy	
L1 Dcache	32 KB, 2-way, 128 byte blocks, 1-cycle latency, 15-cycle snoop latency
L1 Icache	64 KB, 2-way, 128 byte blocks, 1-cycle latency
L2 cache	8 MB, 4-way LRU, 128 byte blocks, 9-cycle latency
Main Memory	80-cycle latency

Table 1: Design parameters for modeled 8-core CPU.

tio by running on as many as 7 cores although this maximum is easily offset by power variation beyond 0.28 W.

The next section describes our experiment and simulation methodology. Section 3 provides our analytical model and simulations for multiprogrammed workloads, while Section 4 extends our analysis and validation to parallel programs. Section 5 covers related work and Section 6 offers our conclusions.

2. EXPERIMENT METHODOLOGY

2.1 Architectural Model

We use an enhanced version of Turandot [26] and PowerTimer [5] to model performance and power of an 8-core PowerPC^{TM}processor. Our cycle-level simulator also incorporates HotSpot version 2.0 [13, 32] in order to model the temperature-dependence of leakage power from various components. Our process and architectural parameters are given in Table 1.

We assume ABB and adaptive V_{DD} can be applied at the granularity of individual cores, ensuring timing correctness while bringing some cores beyond their normally allowed power specification.

2.2 Simulation Setup

This work introduces *Parallel Turandot CMP* (PTCMP), our cycle-level simulator. Unlike its predecessor Turandot CMP [21], PTCMP is programmed with POSIX threads rather than process forking to achieve lightweight synchronization and parallel speedup. This infrastructure is an alternative to Zauber [22], which also avoids slowdown in the fork-based Turandot CMP, but by using a non-cycle-accurate approximation. Our method maintains all necessary cycle-level communication. PTCMP is able to test various combinations of CMP and SMT configurations without limits on the number of cores that have been vexing for some prior simulators. The maintained cycle-level communication not only aids with accurate modeling of shared cache contention, but is also necessary for other enhancements described below.

Like its predecessors, PTCMP also incorporates online calculations for power and temperature through integration with PowerTimer and HotSpot.

2.3 Benchmarks

We use all 26 benchmarks from the SPEC 2000 benchmark suite [12] to formulate several 8-program workloads. These programs are traced with Aria [26] in their appropriate SimPoint [30] intervals. The SPLASH-2 applications, on the other hand, are traced using the Amber tool on Mac OS X [2] from the beginning to end of their complete algorithm executions.

2.4 Modeling Thread Synchronization and Coherence

There are two main issues in our extensions to Turandot for parallel program simulation: synchronization and coherency. Fortunately, from an implementation perspective these can be dealt with independently. We implement modeled lock synchronization (not to be confused with the implementation's internal synchronization) and a MESI cache coherence protocol [28] to maintain memory consistency across each core's local cache with respect to the shared L2 cache.

We use Amber's thread synchronization tracing system in order to accurately track the status of pthread-based mutexes and condition variables. Our trace-driven simulator then models stalls for individual threads when such thread-communication dependencies are detected.

For shared memory coherence we implement a MESI [28] cache-coherence protocol to allow private copies of data in each core's local data cache. The data coherence is with respect to the shared L2 cache. We have correspondingly extended PowerTimer to account for the energy cost of cache snoop traffic.

2.5 Metrics

We are most interested in the metric of performance per watt. Since we use SimPoint-generated [30] subsections of the SPEC 2000 suite and some benchmarks may complete in different proportions depending on system properties, we use the weighted speedup metric [33] in order to measure performance. This effectively takes the sum of the programs' executions relative to their baseline single-threaded performance on our processor.

For our parallel program experiments, we use complete executions of 8 benchmarks from the SPLASH-2 suite. Because these are run to completion, in Section 4 we simply use the speedup ratio relative to the performance of a single node.

3. POWER-PERFORMANCE OF MULTIPROGRAMMED WORKLOADS

3.1 Analysis

We seek to maximize the throughput/energy ratio in spite of excess power on cores due to process variation, defined as P_{excess}. This excess power can arise due to inherent leakage variation but also as an after-effect of circuit techniques applied to ensure timing reliability. Specifically, cores that do not meet timing requirements at the time of manufacture can be receive ABB or V_{DD} adjustments, but this may cause these cores to go beyond its specified power limit [34].

ABB, when applied in forward mode, involves placing a positive bias between the body and source. Thus, these two techniques may be best used in combination to ensure timing requirements. This does not increase the dynamic switching power, but increases leakage power significantly more than V_{DD} adjustment [34]. Thus, these techniques may be best used in combination to ensure timing requirements.

For a given timing adjustment the supply voltage must be scaled up roughly linearly, resulting in an approximately linear increase in leakage power and quadratic increase in dynamic power. For cores which already meet their timing requirements with sufficient slack, we may also apply ABB in reverse (known as reverse body bias, RBB) or lower V_{DD} in order to save power. Even in a fortunate scenario where all cores have some timing slack, the degree to which ABB or V_{DD} adjustment can be applied will differ across cores. This combined with inherent leakage variation results in a set of cores on one die with possibly very different power characteristics.

For the purpose of managing these resultant power variations, the metric we aim to maximize is the ratio of *performance/watt*, a current focus for modern server applications [18] and one of the primary concerns for mobile platforms. We have also considered some more complex scenarios such as minimizing power for a fixed performance deadline or maximizing performance for a fixed power budget. These other analysis routes are interesting areas for future study, but for simplicity we have chosen to focus on maximizing the performance/power ratio.

Our approach is to find the appropriate cutoff point such that a system may decide to turn a power-hungry core off. There may often be a benefit to retaining an extra power-hungry core, since more running cores can help amortize power cost of dynamic and leakage power from shared resources such as the L2 cache.

We wish to see the appropriate cutoff point for when this core offers enough performance to make its wattage worthwhile, versus when we should put this core into sleep mode and make do with the remaining resources. Our criteria for when a core should be disabled can be stated in terms of an inequality relating the performance/power ratio of N cores to $N-1$ cores, as follows:

$$\frac{perf_N}{power_{N\&excess}} \leq \frac{perf_{N-1}}{power_{N-1}} \tag{1}$$

Here, $perf_N$ and $power_{N\&excess}$ represent the performance of power of the full processor with all N cores used, including the core with excess power. The corresponding $perf_{N-1}$ and $power_{N-1}$ represent those values when the Nth core (sorted from lowest to highest excess power) is turned off. We then expand the condition of Equation 1 as such:

$$\frac{N perf_1 \alpha_N}{N P_{core} + P_{excess} + P_{shar,N}} \leq \frac{(N-1) perf_1 \alpha_{N-1}}{(N-1) P_{core} + P_{shar,N-1}} \tag{2}$$

where N represents the number of active cores, P_{core} denotes average core power, P_{shar} denotes power shared among cores, such as power consumed by the L2 cache, and α represents a speed factor to take resource contention into account. Various elements such as P_{shar} are subscripted to indicate they have a specific value for different core configurations, while others such as P_{core} are taken as constant across different values of N. In fact we assume only a single value of P_{core}, since core power tends to vary significantly less than cache and interconnect power.

α is a factor typically less than 1. It represents the slowdown caused by contention on shared resources, a critical design element of CMPs. If there is little shared cache or memory contention, it becomes likely that $\alpha \approx 1$ [21]. This property does not hold for memory-intensive benchmarks, so we use the much weaker assumption that $\frac{\alpha_N}{\alpha_{N-1}} \approx 1$, which says that the incremental contention from an additional core is reasonably small for moderately sized N.

Solving for P_{excess} under these conditions, this results in:

$$P_{excess} \geq \left(\frac{N}{N-1}\right) P_{shar,N-1} - P_{shar,N} \tag{3}$$

This equation states our criterion in a relatively simple manner, by depending only on the power cost of shared resources but not the baseline core power nor contention factors. In essence, we plan to turn off any cores for which Equation 3 is true. This is one of our key insights that we utilize and validate.

If the condition of Equation 3 were to be checked and acted upon dynamically, this would require knowing the value of P_{excess}, which is calculated per core relative to the average across all cores, and P_{shar}. Direct power measurement, such as used in Intel's Foxton technology [27], could be done individually on all cores and the shared cache in order to provide the numerical input for these calculations.

While not readily apparent from Equation 3, a general characteristic of the cutoff point is that it increases roughly linearly with respect to the power cost of shared resources. This can be seen more clearly in the simplest case. When P_{shar} does not vary significantly with respect to N, the expression in Equation 3 simplifies down to:

$$P_{excess} \geq \frac{P_{shar}}{N-1} \tag{4}$$

integer-only	bzip2, crafty, eon, gcc, parser, perlbmk, vortex, vpr
FP-only	applu, equake, galgel, mesa, mgrid, sixtrack, swim, wupwise
memory-bound	ammp, applu, art, gap, lucas, mgrid, swim, twolf
CPU-bound	apsi, eon, equake, fma3d, gcc, gzip, mesa, sixtrack
mix1	ammp, crafty, facerec, galgel, mcf, parser, vpr, wupwise
mix2	applu, apsi, eon, gap, gcc, gzip, lucas, twolf

Table 2: Various 8-program workloads consisting of SPEC benchmarks for our experiments.

In this case, the amount of room for power variation increases linearly with P_{shar}. Similarly, with N in the denominator, increasing the number of active cores takes care of amortizing these costs and hence reduces room for large values of P_{excess}. In our validation experiments, we use Equation 3 to account for variation in shared cache power, but our results in Section 3.3 do confirm this insight of amortized shared power costs with varying N.

3.2 Test Workloads

We examine six characteristic workloads for the purpose of confirming our power-performance tradeoff analysis in the multiprogramming case. Our specified workloads are grouped as shown in Table 2. The first four workloads have benchmarks selected for the purposes as listed in the table. The remaining two workloads are formulated to include all remaining SPEC benchmarks.

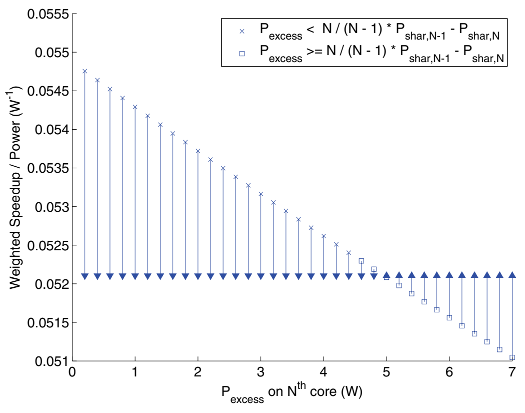
3.3 Results

Our validation uses the workloads listed in Table 2. However, when a lower power mode is entered by putting one core to sleep, one corresponding benchmark must be removed from the workload. In each case, we remove the median benchmark in terms of core power consumption, in order to ensure its value of P_{core} reasonably matches the average core power. For example in the mix1 workload, `ammp` is moderate in terms of power consumption and so is absent in the $N = 7$ case.

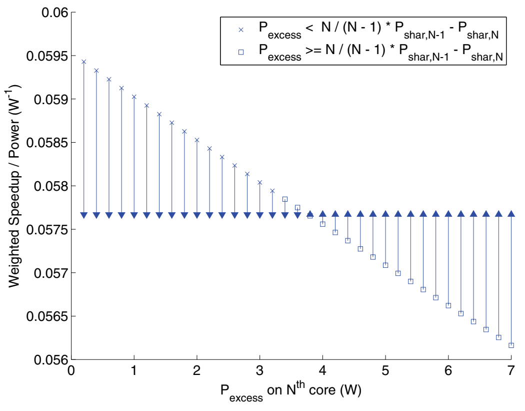
Modern process technologies create a roughly Gaussian distribution of product bins with nontrivial quantities of processors in bins for poor timing performance when using default voltage settings. Thus, our analysis needs to explore power excesses ranging from zero to as much as 50% in excess of the target core power. Figure 1 shows the performance/power ratio across various values of P_{excess} for $N = 6$ and $N = 8$, respectively. Each arrow represents the change in overall performance/power ratio due to turning off one core. Specifically, the lengths of the arrows represent the magnitude of gains or losses in the ratio due to dropping to a configuration of $N-1$ cores. The boxes and cross-marks distinguish whether the analytical criterion in Equation 3 is satisfied or not. In these two figures, all non-satisfying cases result in a decrease in performance/power while almost all criterion-satisfied cases result in an improvement. The Y axes shows a relevant subrange for which the ratio rises or falls in this region.

Notably, the observed and predicted safe ranges for P_{excess} are somewhat larger in the 6-program case than the 8-program one. This is an example of the difference in amortized shared power costs as we have described in Section 3.1. The smaller N configuration gives a larger safe range for P_{excess}, and we observe this in most cases of reduced N for all workloads. Furthermore, for a given 8-core processor the probability of having a dangerously large excess power remaining after shutting down 3 cores is unlikely as those cores were already likely to be the most power-hungry ones. The obvious drawback in the mode with less active cores, as shown in Figure 1(a), is that this configuration has a lower overall performance/power ratio. Thus, our most relevant cases lie with using all 8 cores.

For clarity we have performed our tests assuming only a single process-variant core, but our equations can also apply to variance across multiple cores. In the case of multiple variations, P_{core} would refer to the average power of the $N-1$ lower power cores, while P_{excess} still represents the difference



(a) mix1, $N = 6$



(b) mix1, $N = 8$

Figure 1: Performance/power ratio impacts due to sleeping an additional core when running the mix1 workload.

between the default power consumption of the most consuming core and that average.

We have summarized the cutoff points and our analytical criterion's margin of error for all workloads in Table 3. Error in this sense refers to the small range where our analytical criterion would make an incorrect system decision, such as turning off a core but resulting in reduced performance/power or not disabling a core when it should have. Equation 3 predicts the correct P_{excess} cutoff points with a mean squared error of less than 0.5 W. For perspective, the target design power for the entire processor is around 90 watts, where each core consumes about 8 watts. Total L2 cache and bus power ranges from 10 watts to 40 watts. At low access rates, the shared cache primarily consumes leakage power, while at high cache activity the total L2 power consumption is mainly reflected by the rate of cache accesses.

3.4 Underlying Themes

Our validation experiments have focused on only CPU power, but system designers may wish to include the full power cost of other resources including RAM, chipsets, memory-buffering add-ons, and other essentials. Equation 4 describes the general trend of how including all these elements generally serves to raise the cutoff point, due to better amortizing shared costs. This is one way to confirm general intuition regarding power efficiency of multicore designs.

On the flip side, Equation 4 also denotes an inverse relation between the P_{excess} cutoff and the number of cores. In the future, if multicore designs can feasibly can scale up to many more cores, this increases the chance that one core may become no longer worthwhile to use in the goal of maximizing performance/power. Even so, modern mobile platforms are designed with multiple power modes, and feasibly a highest power mode may seek to maximize performance and still have use for the core in some situations, while a lower power mode may aim to maximize performance/watt as has been the goal in our work.

workload ($N = 8$)	P_{excess} cutoff	model agreement
integer-only	1.25 W	−0.04 W
FP-only	3.88 W	+0.95 W
memory-bound	5.61 W	−0.09 W
CPU-bound	3.48 W	−0.01 W
mix1	3.77 W	+0.39 W
mix2	2.06 W	−0.29 W

Table 3: Summary of P_{excess} cutoffs and respective agreement with analytical model for multiprogrammed workloads (all $N = 8$).

4. POWER-PERFORMANCE OF PARALLEL APPLICATIONS

4.1 Analysis

Our analysis here uses much of the same methodology as in Section 3.1. A key difference is that our speed factor is no longer $\alpha_N N$, which is effectively linear, but rather dictated strongly by Amdahl's Law for parallel computation. The vast topic of parallel computation can certainly entail many complex versions of Amdahl's Law [11, 24, 31], but we choose the most basic form as sufficient for our analysis:

$$speedup_N = \frac{1}{s + \frac{1-s}{N}} \tag{5}$$

where s represents the fraction of sequential/serial computation that cannot be parallelized, and likewise $(1-s)$ represents that fraction that can be parallelized. The value of s is an important important application characteristic in deciding optimal performance tradeoffs. Using this, we perform an analysis similar to that used in Section 3.1. We evaluate Equation 1 and solve for P_{excess} as follows:

$$\frac{\frac{1}{s+\frac{1-s}{N}} perf_1}{NP_{core} + P_{excess} + P_{shar,N}} \leq \frac{\frac{1}{s+\frac{1-s}{N-1}} perf_1}{(N-1)P_{core} + P_{shar,N-1}} \tag{6}$$

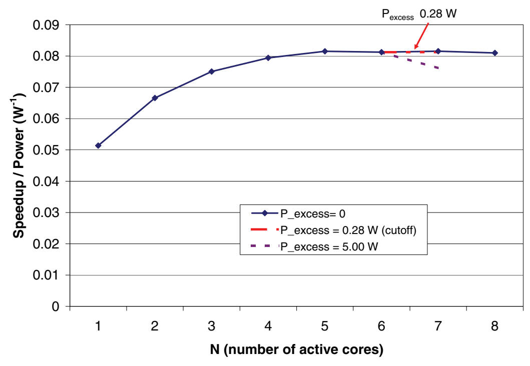
$$P_{excess} \geq \frac{s + \frac{1-s}{N-1}}{s + \frac{1-s}{N}} [(N-1)P_{core} + P_{shar,N-1}] - NP_{core} - P_{shar,N} \tag{7}$$

The above relation is more complex than the properties found in Section 3, but we can use it to study several properties of parallel programs. Because this criterion relies heavily on s, which is an empirical constant that varies not only from benchmark to benchmark but even within different configurations for a single benchmark, it cannot used to accurately predict cutoff points as we had done in Section 3. It does, however, have distinct limit properties that give us much insight to the general power behavior of parallel applications.

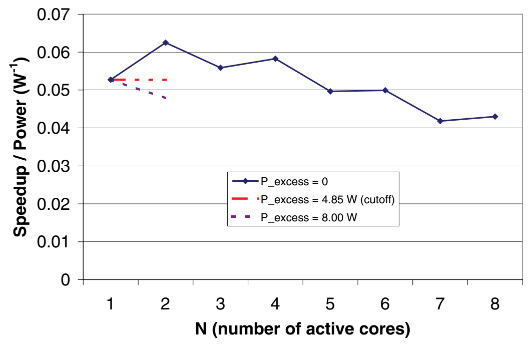
First, Equation 7 is actually a special case of its analog for the multiprogram analysis. When $s = 0$, this represents that the application is completely parallelizable with no penalty of dependencies or contention. Substituting $s = 0$ into the expression and simplifying yields exactly Equation 3.

On the other hand, the case of $s = 1$ represents the worst case of limited parallel speedup, where a program will not run any faster on multiple cores as compared to just one. If we substitute in $s = 1$, the expression evaluates to $P_{excess} \geq -P_{core}$, which is always true and confirms that in such a specific situation reducing the execution down to a single core will always increase the performance/power ratio. Thus, in order to seek a P_{excess} cutoff that is greater than zero, it helps if $s \ll 1$.

Next, always of interest in parallel programming problems are the limits of scaling up to large values of N. Taking the limit for large N with a nonzero value of s results in the fol-



(a) `raytrace`, best performance/power ratio at $N = 7$.



(b) `cholesky`, best performance/power ratio at $N = 2$.

Figure 2: Performance/power for two benchmarks across varying N and varying P_{excess} at the value of N providing the highest performance/power ratio for each benchmark.

lowing suboptimality condition:

$$P_{excess} \geq P_{shar,N-1} - P_{shar,N} - P_{core} \qquad (8)$$

Since typically $P_{shar,N} \geq P_{shar,N-1}$ this relation almost always also evaluates to a negative cutoff value for P_{excess}, meaning that the performance/power ratio only decreases after going beyond some finite N. Intuitively, we would thus expect a plot of this ratio vs increasing N to be a concave-down function that begins to decrease after leveling off. Furthermore, if the performance/power ratio grows only slowly before reaching this peak, we would expect a smaller allowable range for P_{excess}, as compared to the values found in Section 3 where the ratio would continue growing even up through $N = 8$.

4.2 Results

We use 8 of the 12 benchmarks from the SPLASH-2 benchmark suite [36]. Although we have actually conducted experiments with all 12 programs, three of the benchmarks—`ocean`, `fft` and `radix`—are algorithmically restricted from running a number of threads that is not a power of 2. To show clear tradeoffs across the number of cores we have focused only on other benchmarks which do provide this flexibility. Among the remaining 9 benchmarks, `volrend`'s runtimes are an order of magnitude longer than that of other programs, so we have focused on the remaining 8.

We use only the true execution phase of each SPLASH-2 benchmark run for our timing and power measurements. This phase begins after creation of all child threads and ends upon their completion, but does not include any long initialization phases beforehand nor the section of code at the end of each benchmark that generates a summary report.

Figure 2 gives examples of varying performance/power ratios with respect to N and P_{excess}. In each case, the main curve spanning all core counts assumes zero power excess on all cores. The two additional lines represent the change in

application	s	max N	P_{excess} cutoff
barnes	0.039	7	2.02 W
cholesky	0.254	2	4.85 W
fmm	0.066	5	1.41 W
lu	-0.009	8	1.34 W
radiosity	0.083	6	6.04 W
raytrace	0.044	7	0.28 W
water-nsquared	0.025	8	2.03 W
water-spatial	0.019	4	9.15 W

Table 4: Experimental results for SPLASH-2 benchmarks, showing most power-efficient N, cutoff for P_{excess} at that configuration, and each benchmark's corresponding s value.

power efficiency for possible values of P_{excess} at the otherwise optimal performance/watt point. In the first example (`raytrace`), we see a best configuration at $N = 7$, with only a small allowable range of power variation. In the second example (`cholesky`), we see good power efficiency occurring not beyond 2 cores. This is largely in part due to this algorithm's large serial portion [36].

The non-smooth patterns in the `cholesky` graph reveal other notable effects. In particular, core configurations that are not set as a power of 2 each take an additional performance reduction, consequently resulting in poorer performance/power. In fact, most of our benchmarks were found to show some degree of performance preference towards power-of-2 thread counts. This can be explained by non-ideal realities such as cache alignment. Such additional factors affecting performance have a rather direct effect on the performance/power ratio.

One difference between our results here as compared to Section 3 is that limitations are reached at a finite number of cores, as formulated by our analysis in Section 4.1. Our results for all parallel programs are summarized in Table 4. Individual s values used in our calculations for various benchmarks are calculated from the best fit according to speedups obtained through our simulations. However, we have also tested these programs on a real 8-way SMP system to confirm similar respective parallel speedup characteristics.

There are a few interesting cases shown. For one, `lu`'s characteristics best fit a negative value of s, meaning it received super-linear speedup with respect to N in some cases. This is unusual, although not impossible, as many complex effects such as improved cache hit ratios can combine for such a result.

Second, `radiosity` and `water-spatial` have unusually high allowable P_{excess} ranges for their optimal core configurations. The reason for this seems to be due to an interplay of effects that just happen to cause peak performance—possibly more due to negative effects on the adjacent configurations—at these choices of N for these two benchmarks.

Overall, these results confirm much of the intuitive limit behavior specified by our analytical formulation. However, unlike in Section 3, this formulation cannot accurately predict the numerical value of P_{excess} at any given finite point. These deviations from the analytical prediction are due to many application-specific non-ideal factors. A power-efficient multicore system design can take advantage of estimates based on the limit behaviors we have formulated, but an interesting topic for future study would be how a dynamic policy would adjust estimates to take into account application-specific special cases. Such a policy would not only utilize direct measurement of core power, as needed in the multiprogrammed case of Section 3, but also involve performance monitoring to track parallel efficiencies.

5. RELATED WORK

Much prior work has examined power and performance characteristics of multicore architectures when running multiprogrammed workloads [7, 10, 16, 17, 21, 22, 29] as well as parallel applications [3, 8, 15, 19, 20]. To the best of our knowledge, however, ours is the first to examine these problems in the context of process variation.

Although the majority of past research on variation tolerance has been at the circuit and device levels, recently a number of architectural approaches have been proposed. These include variation-tolerant register files [23], caches [1, 25], and pipeline organizations [9, 35]. Furthermore, Humenay et al. propose a model for variations in multicore architectures [14] while Chandra et al. provide a methodology for modeling variations during system-level power analysis [6].

6. CONCLUSIONS

Our work presents a foundation for power-performance optimization in the face of process variation challenges. We have formulated a simple analytical condition relating the shared power costs to predict an optimal cutoff point for turning off extra cores. Using PTCMP to model an 8-core processor, we have shown our model agrees on average within 0.5 W for the basic case of multiprogrammed workloads. P_{excess} cutoff values on our simulated processor range from 1 to 6 W depending on the workloads at hand.

Our analysis has been extended to parallel programming, a more complex problem that is relevant as software in the server, desktop, and mobile sectors all toward more common use of multithreaded applications. We have shown that our equations can be augmented to incorporate Amdahl's Law. Using the parameter s to represent each application's fraction of sequential execution, we have formulated a model to predict limit property trends across a range of parameters and demonstrated these properties on the SPLASH-2 benchmarks.

The purpose for finding these appropriate tradeoff points comes from a system design perspective. If a system is aware of its inter-component dynamic and leakage power excesses, it can make variation-aware decisions for allocating cores in a power-efficient manner. For future work, we intend to combine our on-off mechanism with dynamic voltage and frequency scaling and dynamic ABB to cover a more exhaustive power management design space. In an age when portable devices may execute different types of applications, such techniques are necessary to provide appropriate tradeoffs in performance and power in spite of different application characteristics and process variations.

7. ACKNOWLEDGEMENTS

We thank the Architecture/Performance Group at Apple and Jonathan Chang for their assistance with various tools. We also thank Chris Sadler and the anonymous reviewers for their helpful comments. This work is supported in part by grants from NSF, Intel, SRC, and the C2S2/GSRC joint microarchitecture thrust.

8. REFERENCES

[1] A. Agarwal et al. A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies. *IEEE Transactions on VLSI Systems*, 13(1), Jan. 2005.

[2] `amber(1)` manual page. BSD General Commands Manual, Dec. 2005.

[3] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law Through EPI Throttling. In *ISCA '05: Proc. of the 32nd Intl. Symp. on Computer Architecture*, June 2005.

[4] S. Borkar et al. Parameter Variations and Impact on Circuits and Microarchitecture. In *DAC '03: Proc. of the 40th Design Automation Conf.*, June 2003.

[5] D. Brooks et al. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *IEEE Micro*, 20(6):26–44, Nov/Dec. 2000.

[6] S. Chandra et al. Considering Process Variations During System-Level Power Analysis. In *ISLPED: Proc. of the Intl. Symp. on Low Power Electronics and Design*, Oct. 2006.

[7] J. Donald and M. Martonosi. Temperature-Aware Design Issues for SMT and CMP Architectures. In *WCED-5: Proc. of the 5th Wkshp. on Complexity-Effective Design*, June 2004.

[8] M. Ekman and P. Stenstrom. Performance and Power Impact of Issue-width in Chip Multiprocessor Cores. In *ICPP '03: Proc. of the 32nd Intl. Conf. on Parallel Processing*, Oct. 2003.

[9] D. Ernst et al. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *MICRO-36: Proc. of the Intl. Symp. on Microarchitecture*, Dec. 2003.

[10] S. Ghiasi and D. Grunwald. Design Choices for Thermal Control in Dual-Core Processors. In *WCED-5: Proc. of the 5th Wkshp. on Complexity-Effective Design*, June 2004.

[11] J. L. Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5):532–533, May 1988.

[12] J. L. Henning. SPEC CPU2000: Measuring CPU Performance in the New Millennium. *IEEE Computer*, 33(7):28–35, July 2000.

[13] W. Huang et al. Compact Thermal Modeling for Temperature-Aware Design. In *DAC-41: Proc. of 41st Design Automation Conf.*, June 2004.

[14] E. Humenay et al. Impact of Parameter Variations on Multicore Architectures. In *ASGI: Proc. of the First Wkshp. on Architectural Support for Gigascale Integration*, June 2006.

[15] I. Kadayif, M. Kandemir, and U. Sezer. An Integer Linear Programming Based Approach for Parallelizing Applications in On-Chip Multiprocessors. In *DAC '02: Proc. of the 39th Design Automation Conf.*, June 2002.

[16] S. Kaxiras et al. Comparing Power Consumption of an SMT and a CMP DSP for Mobile Phone Workloads. In *CASES '01: Proc. of the 2001 Intl. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, Nov. 2001.

[17] R. Kumar et al. Processor Power Reduction via Single-ISA Heterogeneous Multicore Architectures. *Computer Architecture Letters*, Apr. 2003.

[18] J. Laudon. Performance/Watt: The New Server Focus. In *dasCMP '05: Proc. of the Wkshp. on Design, Analysis, and Simulation of Chip Multiprocessors*, Nov. 2005.

[19] J. Li and J. F. Martínez. Power-performance implications of thread-level parallelism on chip multiprocessors. In *ISPASS: Proc. of the 2005 Intl. Symp. on Performance Analysis of Systems and Software*, Mar. 2005.

[20] J. Li and J. F. Martínez. Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors. In *HPCA '06: Proc. of the 12th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2006.

[21] Y. Li et al. Performance, Energy, and Thermal Considerations for SMT and CMP Architectures. In *HPCA '05: Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2005.

[22] Y. Li et al. CMP Design Space Exploration Subject to Physical Constraints. In *HPCA '06: Proc. of the 12th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2006.

[23] X. Liang and D. Brooks. Latency Adaptation of Multiported Register Files to Mitigate Variations. In *ASGI: Proc. of the First Wkshp. on Architectural Support for Gigascale Integration*, June 2006.

[24] Massively Parallel Technologies. http://www.massivelyparallel.com/, 2006.

[25] K. Meng and R. Joseph. Process Variation Aware Cache Leakage Management. In *ISLPED: Proc. of the Intl. Symp. on Low Power Electronics and Design*, Oct. 2006.

[26] M. Moudgill, J.-D. Wellman, and J. H. Moreno. Environment for PowerPC Microarchitecture Exploration. *IEEE Micro*, 19(3):15–25, May/June 1999.

[27] S. Naffziger. Dynamically Optimized Power Efficiency with Foxton Technology. In *Proc. of Hot Chips 17*, Aug. 2005.

[28] M. Papamarcos and J. H. Patel. A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories. In *ISCA '84: Proc. of the 11th Intl. Symp. on Computer Architecture*, June 1984.

[29] M. D. Powell, M. Gomaa, and T. N. Vijaykumar. Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System. In *ASPLOS-XI: Proc. of the 11th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2004.

[30] T. Sherwood et al. Automatically Characterizing Large Scale Program Behavior. In *ASPLOS-X: Proc. of the 10th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[31] Y. Shi. Reevaluating Amdahl's Law and Gustafson's Law. Computer Sciences Department, Temple University (MS:38-24), Oct. 1996.

[32] K. Skadron et al. Temperature-Aware Microarchitecture. In *ISCA '03: Proc. of the 30th Intl. Symp. on Computer Architecture*, Apr. 2003.

[33] A. Snavely and D. Tullsen. Symbiotic Jobscheduling for a Simultaneous Multithreading Architecture. In *ASPLOS IX: Proc. of the 8th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[34] J. Tschanz et al. Effectiveness of Adaptive Supply Voltage and Body Bias for Reducing Impact of Parameter Variations in Low Power and High Performance Microprocessors. *IEEE Journal of Solid-State Circuits*, 38(5), May 2003.

[35] X. Vera, O. Unsal, and A. González. X-Pipe: An Adaptive Reslient Microarchitecture for Parameter Variations. In *ASGI: Proc. of the First Wkshp. on Architectural Support for Gigascale Integration*, June 2006.

[36] S. C. Woo et al. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *ISCA '95: Proc. of the 22nd Intl. Symp. on Computer Architecture*, June 1995.