Efficiency Trends and Limits from Comprehensive Microarchitectural Adaptivity

Benjamin C. Lee and David Brooks

School of Engineering and Applied Sciences, Harvard University

{bclee,dbrooks}@eecs.harvard.edu

Abstract

Increasing demand for power-efficient, high-performance computing requires tuning applications and/or the underlying hardware to improve the mapping between workload heterogeneity and computational resources. To assess the potential benefits of hardware tuning, we propose a framework that leverages synergistic interactions between recent advances in (a) sampling, (b) predictive modeling, and (c) optimization heuristics. This framework enables qualitatively new capabilities in analyzing the performance and power characteristics of adaptive microarchitectures. For the first time, we are able to simultaneously consider high temporal and comprehensive spatial adaptivity. In particular, we optimize efficiency for many, short adaptive intervals and identify the best configuration of 15 parameters, which define a space of 240B points.

With frequent sub-application reconfiguration and a fully reconfigurable hardware substrate, adaptive microarchitectures achieve $bips^3/w$ efficiency gains of up to 5.3x (median 2.4x) relative to their static counterparts already optimized for a given application. This 5.3x efficiency gain is derived from a 1.6x performance gain and 0.8x power reduction. Although several applications achieve a significant fraction of their potential efficiency with as few as three adaptive parameters, the three most significant parameters differ across applications. These differences motivate a hardware substrate capable of comprehensive adaptivity to meet these diverse application requirements.

Categories and Subject Descriptors B.8.2 [*Performance Analysis and Design Aids*]; I.6.5 [*Model Development*]: Modeling Methodologies

General Terms Design, Experimentation, Measurement, Performance

Keywords Reconfigurablity, Adaptivity, Microarchitecture, Simulation, Statistics, Inference, Regression, Performance, Power, Efficiency

1. Introduction

Adaptive microarchitectures arise from a design paradigm that promises greater performance and power efficiency by dynamically allocating computational resources to meet the requirements of a workload more effectively. Adaptivity enables power efficient, high-performance microarchitectures by provisioning (or over-provisioning) computational resources to maximize overall performance while increasing the locality of associated power costs to periods of computation that actually utilize these resources. As transistor density increases and microprocessor resources become more abundant, designers will have greater flexibility to implement adaptive structures provided that these structures deliver the promised efficiency. The feasibility of this paradigm will be decided by assessing benefits and costs, but a comprehensive assessment of potential benefits has long eluded designers and researchers due to the challenging dimensionality of the analysis.

An alternative to static general-purpose design, the adaptive computing paradigm increases the flexibility of the microarchitecture in two dimensions. In one dimension, the degree of *temporal adaptivity* is the frequency at which resources reconfigure to optimize efficiency. Broadly, this dimension might range from application-level adaptivity to interval-level (*e.g.*, sub-application) adaptivity. Greater temporal adaptivity improves microarchitectural responsiveness to underlying workload heterogeneity but places greater burdens on the adaptive control algorithm. In the other dimension, the degree of *spatial adaptivity* is the microarchitectural scope of reconfigurations. The number and adaptive range of reconfigurable design parameters are units of measurement for this dimension. More comprehensive spatial adaptivity leverages synergies and interactions between parameter to maximize efficiency gains at the cost of greater complexity.

Nested within these two adaptivity dimensions is a highly expensive optimization problem. Microarchitectural adaptivity achieves the greatest potential efficiencies with high temporal and spatial adaptivity. This scenario translates into frequent, short intervals optimized over an adaptive space of many design parameters. Analyzing this scenario is a computationally daunting procedure and is exacerbated by the limitations of detailed simulation. Despite its computational costs, such an optimization is critical to an accurate assessment of potential efficiencies and would provide half the data needed for a rigorous cost-benefit analysis. Significant efficiency gains, if found, would motivate more thorough cost and complexity analyses of microarchitectural adaptivity.

The computational complexity of this problem has hindered advances in microarchitectural adaptivity as prior work often constrained adaptivity in the temporal (*e.g.*, applications[24], working sets[7], subroutines[16], and multimedia frames[17]) and/or spatial (*e.g.*, two or three parameters among depth [8, 31], width[17], queues [1, 27], and caches [24, 4]) dimensions. Analyses with constraints to temporal adaptivity do not fully illustrate the potential efficiency gains of dynamic structural reconfiguration. Without an analysis of comprehensive spatial adaptivity, prior studies may not account for interactions between parameters as structures adapt, resulting in migrating bottlenecks and limited efficiency gains. Fur-

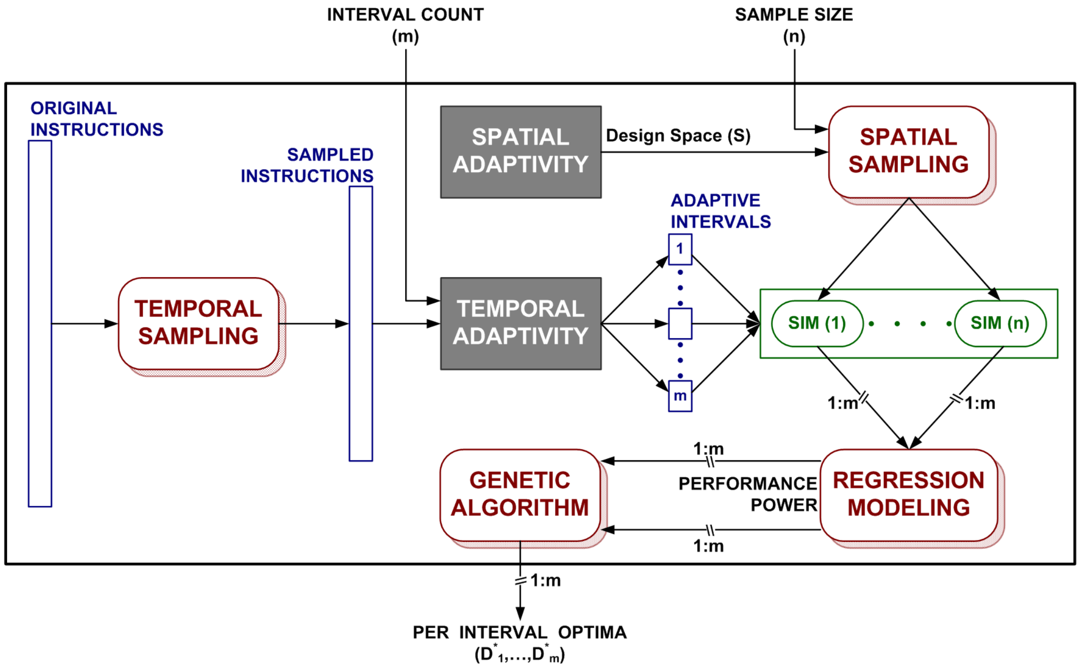


Figure 1. Integrated framework for analyses of microarchitectural adaptivity.

thermore, only a comprehensive study will reveal the most significant parameters for adaptivity in the presence of such interactions. Any further work in adaptive microarchitectures requires solutions to these fundamental limitations.

Recent advances in applying statistical inference and optimization heuristics have enabled qualitatively new capabilities in microarchitectural analysis. In particular, three fundamental advances have dramatically lowered the cost of early-stage performance and power estimation: (a) sophisticated sampling of instruction traces and design spaces, (b) predictive modeling to reduce the reliance on detailed simulation, and (c) heuristics for combinatorial optimization. Each technique in isolation is insufficient for analyzing microarchitectural adaptivity. Applied collectively, however, they produce synergistic and complementary contributions to computational efficiency. This efficiency enables, for the first time, a comprehensive assessment of potential benefits from high temporal and comprehensive spatial adaptivity. In particular, the following summarizes the contributions of this work:

1. **Analysis Framework:** We leverage recent advances in analysis and optimization to construct a coherent framework for adaptive microarchitectures based on random spatial sampling, spline-based regression for prediction, and genetic algorithms for optimization (Section 2). We apply this framework to assess the $bips^3/w$ efficiency from high temporal and comprehensive spatial adaptivity. In particular, we optimize 1,125 adaptive intervals each with 80,000 instructions over an adaptive space of 240B configurations defined by combinations of 15 parameters.

2. **Temporal Adaptivity:** We apply the framework to assess the effects of varying degrees of temporal adaptivity under comprehensive spatial adaptivity (Section 4). We demonstrate significant efficiency gains of up to 5.3x (median 2.4x) from high temporal adaptivity (interval-level optimization, 0.08M-instruction intervals) relative to low-temporal adaptivity (application-level optimization). This 5.3x benefit is derived from a 1.6x performance gain and 0.8x power reduction. Furthermore, we observe interaction between adaptive parameters where the magnitude of each parameter's change increases with the number of simultaneously changed parameters during an interval transition. Such interactions will likely complicate the design of adaptive control algorithms.

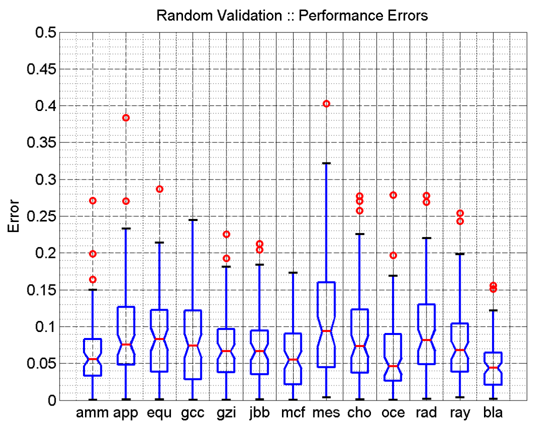
3. **Spatial Adaptivity:** We apply the framework to assess the effects of varying degrees of spatial adaptivity under high temporal adaptivity (Section 5). We examine reduced spatial adaptivity by comparing the efficiencies of adapting a reduced subset of parameters against the efficiencies of adapting all fifteen. Three parameters are often sufficient to achieve, on average, 77.3 percent of fifteen-parameter adaptive efficiency. However, the three most significant parameters differ across applications and a comprehensive adaptive hardware substrate encompassing a majority of design parameters would be needed to fully realize these efficiency benefits.

Thus, we establish a rigorous foundation for assessing the benefits of comprehensive microarchitectural adaptivity. The significant efficiency gains we identify motivate a complementary, equally rigorous analysis of the associated costs and complexities.

2. Statistical Inference and Optimization

A convergence of recent advances in statistical inference and optimization heuristics enable new capabilities in microarchitectural analysis. These techniques control exponentially increasing design space sizes, enable computationally efficient prediction for metrics of interest, and identify global solutions to previously intractable combinatorial optimization problems. None of the individual techniques in isolation are sufficient, but a coherent framework that combines the strengths of each technique exposes synergies that enable qualitatively more comprehensive analyses of microarchitectural adaptivity. This framework not only provides old answers to prior questions more quickly, it also provides new answers to much larger and previously intractable questions.

Figure 1 presents the flow of data through our analysis framework. We take an original instruction trace and pass it through some



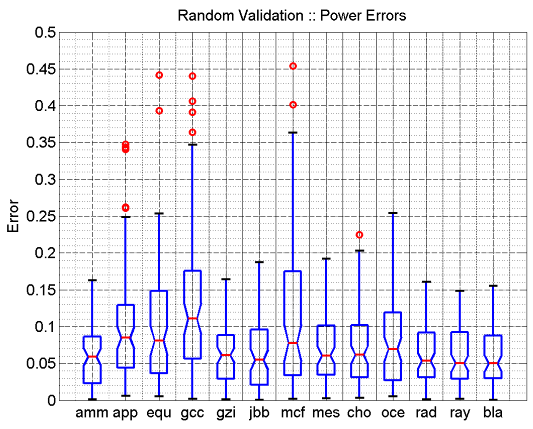


Figure 2. Distribution of prediction errors for 100 random validation designs.

technique for temporal sampling, which produces a reduced trace of representative instructions. This trace is broken into m intervals where m depends on the degree of temporal adaptivity specified by the user. In parallel, we define the space of adaptive parameters S and simulate n sparsely sampled designs from this space for each of the m adaptive instruction intervals. These simulations are used to construct per interval regression models for performance and power. Finally, we identify the best configurations for each interval by applying the genetic algorithm on the regression models.

2.1 Temporal and Spatial Sampling

Prior studies of microarchitectural adaptivity considered only a few parameters due to the computational costs of detailed, cycle-accurate simulation. Such costs are prohibitively high when optimizing per application, per subroutine, or per interval metrics over p parameters each with m possible values. For example, a study of three parameters each with five possible values requires 125 simulations per adaptive interval and a fourth parameter would increase these costs to 625 simulations per interval.

These computational costs are mitigated by *temporal sampling* where representative instruction traces were sampled from the time domain prior to simulation [29, 33]. Although these techniques are highly effective in reducing per simulation costs, they do not impact the number of required simulations for design space optimization. This limitation is problematic since the number of simulations increase exponentially as the design space becomes more comprehensive. Thus, prior studies examined a significantly reduced subset of the design space to accommodate this limitation.

To address this fundamental challenge, we further perform *spatial sampling* by sparsely simulating points selected uniformly at random from a large comprehensive design space. Spatial sampling effectively decouples design space size from the number of simulations required to identify a trend within the space, thus controlling exponential increases in space size [21]. This approach provides observations from the full range of parameter values. An arbitrarily large number of values may be included, thereby achieving high spatial resolution.

2.2 Regression Modeling

Techniques in statistical inference reveal performance and power trends from sparsely simulated samples, enabling adaptivity studies for much larger, comprehensive design spaces. In particular, we apply the approach proposed by Lee and Brooks for spline-based regression models [21]. These models predict a performance or power response as a function of design parameter values. Within this framework, interactions between predictors are captured by products terms specified in the models' functional form using domain-specific knowledge. For example, pipeline depth interacts with cache sizes since depth determines pipeline sensitivity to cache misses. Non-linearity is captured by cubic spline (*i.e.*, piecewise polynomial) transformations on the predictors. Model construction is computationally efficient and may be reduced to a series of cubic transformations followed by a linear solve (highly optimized matrix operations). This efficiency allows the construction of models for every adaptive interval of instructions.

Model evaluation, expressed as matrix multiplication, is also highly efficient. Hundreds or thousands of predictions per second are possible. Figure 2 illustrates model accuracy when validated against simulation for randomly selected validation points, demonstrating median errors of 6.5 and 6.3 percent for performance and power prediction, respectively.[1] These models have been applied to and validated for practical design studies, demonstrating accuracy sufficient for early stage design optimization [22]. This prior work also found outlier errors tend to occur near design space boundaries where models are extrapolating instead of interpolating. Since joint performance and power optimization typically identifies optima well within space boundaries, outliers are unlikely to significantly affect our analysis. The error distributions are presented for models predicting overall application performance and power. However, these errors are comparable to those of sub-application models constructed for intervals of arbitrary length.

2.3 Genetic Algorithms

We estimate upper bounds on the efficiency gains of microarchitectural adaptivity by quantifying gains under best-case, oracle-driven scenarios. In practice, this requires identifying efficiency-maximizing designs for each interval. Although regression models are orders of magnitude faster than detailed microprocessor simulation, using exhaustive search to optimize these models across a design space of nearly 240 billion points remains intractable. We must therefore combine regression models with scalable heuristics, such as genetic algorithms, for global combinatorial optimization.

[1] Boxplots display location (median) and dispersion (interquartile range), identify possible outliers, and indicate the symmetry or skewness of the distribution. Boxplots are constructed by (1) horizontal lines at median and at upper, lower quartiles, (2) vertical lines drawn up/down from upper/lower quartile to most extreme data point within 1.5 IQR of upper/lower quartile where IQR is the interquartile range between first and third quartile, and (3) circles to denote outliers.

Genetic algorithms mimic the process of natural selection in which a candidate solution to the optimization problem is treated as an organism and a candidate's optimality is treated as the organism's fitness [12]. Breeding among highly fit organisms increases the likelihood of passing desirable attributes to future generations. Breeding among less fit organisms and the possibility of mutation ensures population diversity. As the population evolves from one generation to the next, the population of candidate solutions improves and the likelihood of observing the global optimum within the population increases. We describe the genetic algorithm in the context of microprocessor optimization and discuss tunable elements of this algorithm including (1) population size and number of generations, (2) parent selection, (3) genetic crossover, and (4) mutation rates.

Population Size and Generation Count. In the microarchitectural context, each organism is a candidate design represented by a vector of $p = 15$ design parameter values. The algorithm is initialized to a random population of 100 candidates and the system is evolved for 100 generations. If computationally feasible, larger populations are favored since they provide a more diverse genetic pool from which to generate offspring, thereby diversifying the search and discouraging premature convergence to sub-optima. The algorithm should terminate when population diversity is low and the algorithm has converged. We have empirically found 100 generations to strike an effective balance between diversity and convergence for our design space.

Parent Selection. Both parents are selected by fitness rank where fitness is quantified by $bips^3/w$ efficiency and computed using the derived regression models for performance and power. Alternative selection schemes might include selecting one or both parents uniformly at random. These alternatives allow for the possibility of passing weak design attributes from random parents to subsequent generations, thereby slowing convergence and allowing a more diverse search of the space. We evaluated these alternatives and did not find any empirical advantage to random parent selection for this design space.

Genetic Crossover. Once two parents are selected, a variety of genetic operators may be applied to obtain an offspring. In the microarchitectural context, a new candidate design is obtained by constructing a vector of design parameter values from some combination of values from the parents' vectors. The simplest crossover method uses a random position in the p-element vector. Offspring values to the left of this position come from one parent and values to the right of this position come from the other parent. Alternatively, we considered random crossover in which each offspring value is taken from either parent uniformly at random. In practice, we find this latter approach more effective in preserving population diversity through greater genetic mixing.

Mutation. Mutations randomly alter an offspring's genetic code to increase population diversity and provide a mechanism for escaping local optima. We implement an aggressive mutation scheme in which each value of the offspring's design vector can independently mutate up or down by one step (as defined by ranges of Table 2) with 5 percent probability. This particular mutation rate was empirically found to be effective when sweeping a range of possible values. If this rate is too low, many potentially good innovations will be missed. If this rate is too high, the algorithm's ability to preserve desirable attributes will be degraded.

Parents are repeatedly selected to produce mutated offspring until the previous generation is replaced. The algorithm proceeds until the pre-determined generation limit is reached and the best design in the last generation is returned. It is intractable to validate results from genetic algorithms against those from exhaustive search. However, we find the same optima are produced with high frequency when repeatedly invoking the genetic algorithm with random starting populations, giving us confidence in the algorithm's ability to converge consistently toward superior designs.

2.4 Synergies for Adaptivity Analysis

The framework of Figure 1 leverages the synergistic interactions between temporal/spatial sampling, regression modeling, and genetic algorithms. Although each technique reduces computational costs, their combination is required to enable new analysis in microarchitectural adaptivity. Temporal and spatial sampling are complementary techniques and both are required to reduce per simulation costs and the number of required simulations, respectively. However, spatial sampling must leverage predictive models constructed from sparsely simulated points to reveal broader trends in the space. Spline-based regression models are particularly well suited for a framework targeting adaptivity analyses. Since regression model construction is typically expressed as a linear solve and therefore computationally efficient, assessing high temporal adaptivity by constructing one model per interval becomes tractable.

Iterative optimization heuristics, such as genetic algorithms, require hundreds or thousands of predictions. Performance and power must be estimated for every population member across every generation. While implementing genetic algorithms with detailed simulation is possible [10], regression models are particularly effective inputs to the algorithm. These models are able to predict an entire population's performance and power characteristics via two matrix multiplications. This efficiency makes tractable the optimization of models for each of the many, short adaptive intervals. Thus, the strengths of each component in our framework are complementary and suited for modeling microarchitectural adaptivity.

We apply this framework to quantify the efficiency of high temporal and comprehensive spatial adaptivity as follows:

- **Temporal Adaptivity:** We define the degree of temporal adaptivity as the frequency at which the microarchitecture re-optimizes its computational resources. To consider the potential benefits of high temporal adaptivity, we partition workloads into intervals, formulate separate regression models for each interval, and optimize each model independently to identify efficiency maximizing designs. The result of this optimization is equivalent to the efficiency of a microarchitecture that adapts its configuration to maximize $bips^3/w$ for every interval. We quantify the potential efficiency gains when adapting the microarchitecture at varying interval sizes, identify the source of these gains in performance and power analyses, and assess the diversity of adaptive configurations.
- **Spatial Adaptivity:** We define the degree of spatial adaptivity as the number of microarchitectural resources available for adaptivity. To consider the potential benefits of high spatial adaptivity, we define the comprehensive design space of Table 2 and evaluate efficiency when each parameter in this space is adapted to maximize $bips^3/w$. We then consider adapting fewer parameters and assess the resulting limitations to efficiency. Lastly, we assess the impact of orthogonal adaptivity from dynamic voltage and frequency scaling (DVFS).

These assessments of adaptivity assume (1) $bips^3/w$ maximizing configurations are provided by an oracle at the beginning of each interval and (2) microarchitectural reconfiguration is performed with zero delay or power cost. Although these assumptions produce an optimistic estimate, they are appropriate for quantifying potential efficiency bounds on adaptive microarchitectures. Estimates of such bounds are necessary first steps toward a rigorous cost-benefit analysis of microarchitectural adaptivity.

		amm	app	equ	gcc	gzi	jbb	mcf	mes	cho	oce	rad	ray	bla
S_1	depth	9	9	12	15	33	9	18	36	30	27	30	24	15
S_2	width	2	8	2	4	2	2	2	2	8	8	2	8	2
S_3	bp	8	8	1	4	8	8	8	2	8	8	8	8	4
S_4	lsq	31	36	31	31	11	26	11	11	41	41	26	56	21
S_5	reg	80	130	70	130	130	130	130	130	130	130	130	130	130
S_6	resv	11	13	15	6	6	6	6	11	11	12	15	6	6
S_7	i1Size(KB)	16	16	16	64	32	16	16	16	16	16	16	32	32
S_8	i1Assoc	1	1	1	1	1	1	1	1	1	1	1	8	1
S_9	d1Size(KB)	8	16	8	64	32	8	64	64	64	64	64	8	8
S_{10}	d1Assoc	2	1	4	1	1	8	1	1	1	1	1	1	1
S_{11}	d1Lat	1	1	1	1	1	2	1	1	1	2	1	2	2
S_{12}	l2Size(MB)	0.5	0.25	0.25	1	2	1	2	2	2	2	2	0.25	1
S_{13}	l2Assoc	4	1	8	8	1	1	8	2	1	2	1	2	8
S_{14}	l2Lat	8	8	14	14	8	8	8	8	8	8	8	8	8
S_{15}	memLat	90	85	70	70	115	115	70	115	115	70	115	115	90

Table 1. Static baseline configurations that maximize each application's efficiency.

	Set	Parameters	Measure	Range	$\|S_i\|$
S_1	Depth	depth	FO4	9::3::36	10
S_2	Width	width	issue b/w	2,4,8	3
S_3	Branch Predictor	BTB associativity BTB size, log_2	sets entries	1,2,4,8 12::1::15	4
S_4	Load/Store	load/store queue	entries	9::5::54	10
S_5	Physical Registers	general purpose (GP) floating-point (FP) special purpose (SP)	count count count	40::10::130 40::8::112 42::6::96	10
S_6	Reservation Stations	branch fixed-point/memory floating-point	entries entries entries	6::1::15 10::2::28 5::1::14	10
S_7	I-L1 Cache	i-L1 cache size	KB	16::2x::256	5
S_8		i-L1 cache assoc.	sets	1,2,4,8	4
S_9	D-L1 Cache	d-L1 cache size	KB	8::2x::128	5
S_{10}		d-L1 cache assoc.	sets	1,2,4,8	4
S_{11}		load/store latency	cycles	1::1::5	5
S_{12}	L2 Cache	L2 cache size	MB	0.25::2x::4	5
S_{13}		L2 cache assoc.	sets	1,2,4,8	4
S_{14}		L2 cache latency	cycles	8::2::16	5
S_{15}	Memory	memory latency	cycles	70::5::115	10

Table 2. Design space parameters where i::j::k denotes a set of values from i to k in steps of j.

SPEC CPU 2000	
ammp	Simulates molecular dynamics
applu	Solves parabolic/elliptic partial differential equations (PDE's)
equake	Simulates seismic wave propagation
gcc	Compiles C programs
gzip	Performs compression
mcf	Performs combinatorial optimization
mesa	Provides 3-D graphics library support
SPEC JBB 2000	
jbb	3-tier Java business server
SPLASH	
cholesky	Factorizes sparse matrix using blocked Cholesky method
ocean	Simulates ocean using Gauss-Seidel multigrid solver
radiosity	Computes equilibrium distribution of light
raytrace	Renders three-dimensional images
BIOPERF	
blast	Searches database for protein/nucleotide sequencing

Table 3. Benchmarks

3. Experimental Methodology

3.1 Simulation Framework

We use Turandot, a generic and parameterized, out-of-order, superscalar processor simulator [25]. Turandot is enhanced with PowerTimer to obtain power estimates based on circuit-level power analyses and resource utilization statistics [5]. The modeled baseline architecture is similar to the POWER4/POWER5. The simulator has been validated against both a POWER4 RTL model and a hardware implementation. This simulator implements pipeline depth performance and power models based on prior work [35]. Power scales superlinearly as pipeline width increases using scaling factors derived for an architecture with clustered functional units [34]. Cache power and latencies scale with array size according to CACTI [30]. We do not currently model the power overhead of adaptivity, but these overheads have been found to be small in prior work [2]. We do not leverage any particular feature of the simulator in our models and believe our framework may be generally applied to other simulation frameworks with similar effect.

Table 2 identifies fifteen groups of parameters varied simultaneously. Parameters within a group are varied together to avoid fundamental design imbalances. The range of values considered for each parameter group is specified by a set of values, $S_1, \ldots, S_{15}$. The Cartesian product of these sets, $S = \prod_{i=1}^{15} S_i$, defines the entire design space of 240 billion points. We report experimental results from models formulated with 500 samples obtained uniformly at random from the design space S.

We use R, an open-source software environment for statistical computing, to script and automate statistical analyses [28]. Within this environment, we use the Hmisc and Design packages implemented by Harrell [14].

3.2 Benchmarks

We report experimental results based on PowerPC traces of the benchmarks in Table 3 [15, 32, 3]. The traces used in this study were sampled from the full reference input set to obtain 100 million instructions per benchmark program [19]. Systematic validation was performed to compare the sampled traces against the full traces to ensure accurate representation. Our benchmark suite is representative of larger suites frequently used in the microarchitectural research community [26].

3.3 Metrics

We evaluate performance in billions of instructions per second (bips) and power in Watts (w). We measure efficiency using $bips^3/w$, a metric used by many industrial design teams for joint performance and power optimization [6, 13]. This metric is voltage invariant and derived from the cubic relationship between power and voltage (V), frequency (f). Since $w \propto V^2 f$ and $V \propto f$, $w \propto f^3$ and $f^3/w \propto k_0$ where k_0 is some constant. If $bips \propto f$, then $bips^3/w \propto k_1$ where k_1 is some constant and the metric is invariant as voltage and frequency change. Although the magnitude of efficiency gains will change under alternative metrics (*e.g.*, $bips/w$ or $bips^2/w$), the underlying performance gains and power

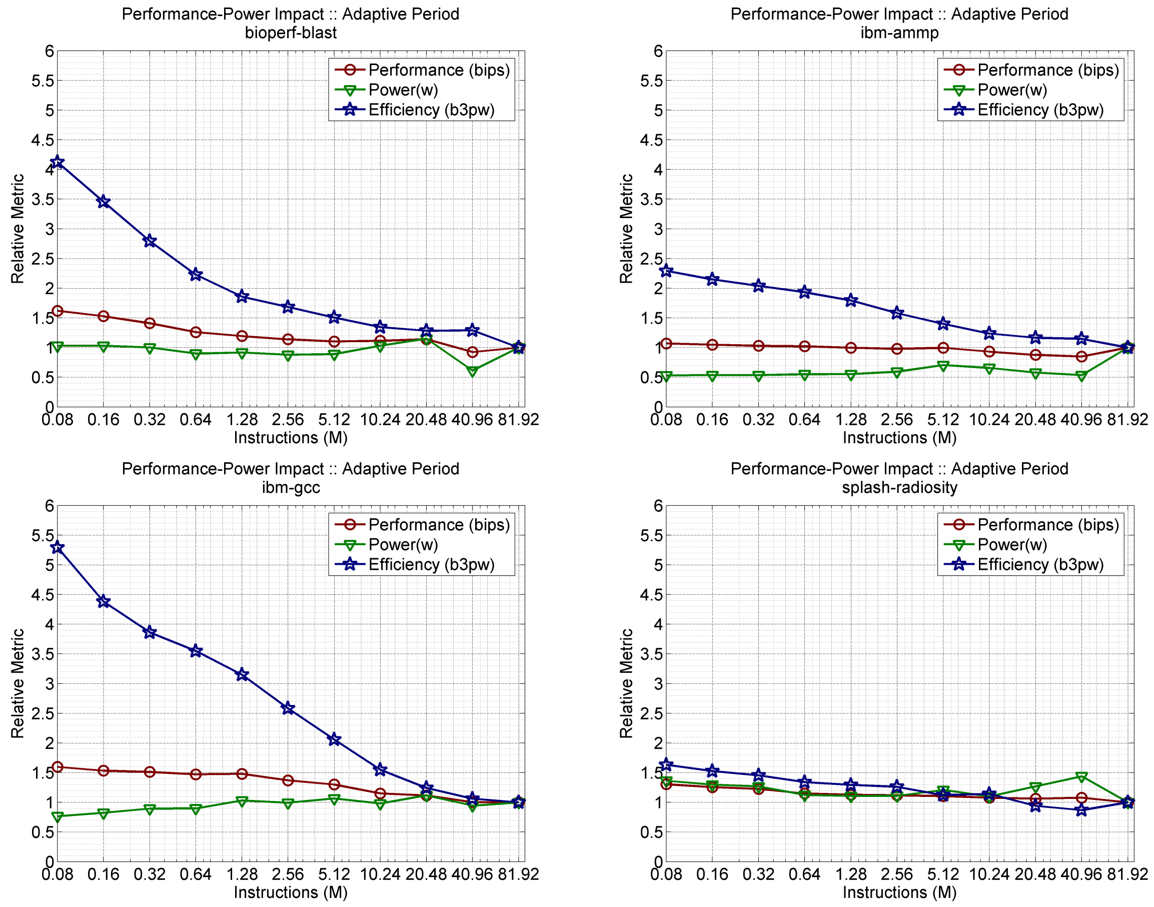


Figure 3. Temporal Adaptivity: Representative efficiency trends for *blast* (UL), *ammp* (UR), *gcc* (LL) and *radiosity* (LR). Microarchitecture reconfigures every 81.92M to 0.08M instructions.

reductions observed in our studies indicate improved efficiency regardless of metric.

3.4 Approximations

The proposed framework implements several approximations to make the adaptivity analysis tractable. We use regression models as computationally efficient surrogates for cycle-accurate simulation, which introduces median errors of approximately 6 percent for performance and power (Section 2.2). However, these errors are small relative to, for example, observed performance gains of up to 60 percent and power reductions of up to 20 percent. Thus, reported trends are well outside the margin of error. Results from genetic algorithms approximate the globally optimal design point. We cannot tractably validate these results against exhaustive search for a space of 240B points. However, we draw confidence in the algorithm since (1) the algorithm returns the same optima when repeatedly invoked with random starting populations and (2) these optima are significant improvements over the baseline.

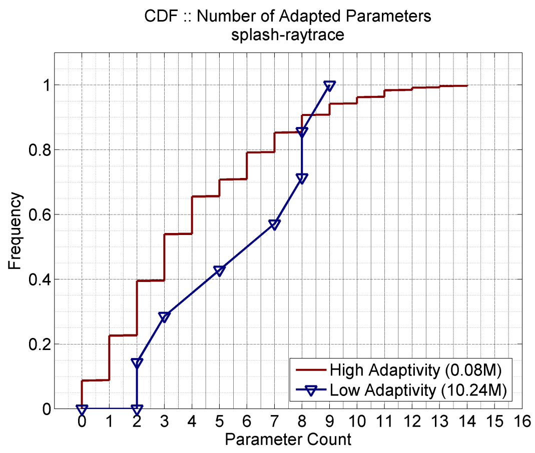
4. Temporal Adaptivity

Regression models are derived for a maximum temporal adaptivity of 80,000 (0.08M) instructions. We compare against lower degrees of temporal adaptivity by recursively combining adjacent pairs of basic 0.08M-instruction intervals to obtain longer intervals with lengths ranging from 0.16M to 81.92M instructions. Practically, combining intervals requires aggregating regression performance and power predictions from the basic 0.08M-instruction intervals to obtain a prediction for the larger interval.

We compare each benchmark's efficiency gains from sub-application adaptivity against those from application-level adaptivity in which an oracle provides the best configuration for overall workload efficiency (Table 1). These optima are similar to optimal core designs for heterogeneous multiprocessors targeting comparable workloads [22]. Each baseline architecture is identified from optimizing the 81.92M-instruction interval. Optimizing this interval from a trace of 100M instructions is roughly equivalent to identifying the overall $bips^3/w$ optimal architecture for a given benchmark since there is only one opportunity to adapt the microarchitectural configuration. Thus, each benchmark's baseline microarchitecture is already optimized for efficiency and we quantify only the additional impact from increasing sub-application temporal adaptivity.

4.1 Efficiency Trends

Figure 3 presents performance, power, and efficiency trends as the adaptive period decreases from 81.92M to 0.08M instructions. The period of 0.08M instructions represents the greatest temporal adaptivity as the microarchitecture adapts to maximize $bips^3/w$ efficiency every 0.08M instructions. These figures illustrate monotonically improving efficiency as temporal adaptivity increases with up to 5.3x efficiency gains (*gcc*). The source of efficiency gains vary across benchmarks and arise from performance improvements and/or power reductions.



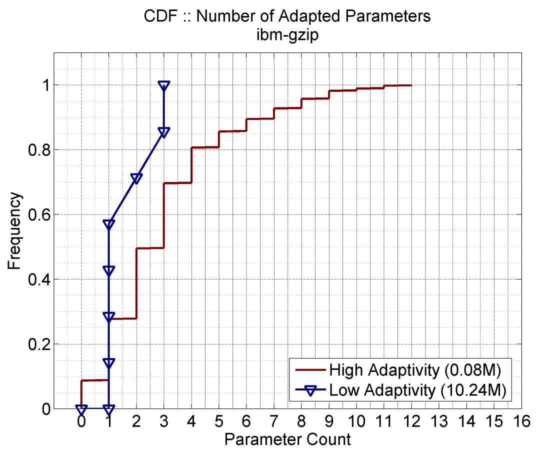


Figure 4. Number of parameters that change between consecutive intervals for *raytrace*(L) and *gzip*(R). High and low adaptivity CDF's are constructed for 1125 0.08M-instruction and 9 10.24M-instruction intervals, respectively.

For example, Figure 3UL illustrates efficiency gains for *blast* dominated by performance improvements. These trends are also representative of those for *equake* and *mcf*. Adapting the microarchitecture every 0.08M instructions improves efficiency by 4.1x, derived from a 62.0 percent increase in performance and negligible 3.1 percent increase in power relative to application-level adaptivity. Power trends are flat, illustrating scenarios where power costs are limited only to intervals utilizing more of the adaptive resources. Short, high power intervals do not appreciably raise an application's average power costs.[2] This temporal locality of power costs might be achieved, for example, by fine-grained power gating of unused resources.

In contrast, Figure 3UR illustrates efficiency gains for *ammp* characterized by significant power reductions. These trends are also representative of those for *applu* and *cholesky*. *Ammp* achieves maximum efficiency gains of 2.3x from a modest 6.9 percent increase in performance and 46.7 percent decrease in power relative to the static baseline. Although modest performance improvements between 2.56M- and 0.08M-instruction intervals provide monotonically increasing efficiency, adaptivity achieves notable power reductions between 29.4 and 46.7 percent that contribute significantly to greater efficiency across all adaptive periods.

Figure 3LL illustrates the more common case in which increasing temporal adaptivity both increases performance and decreases power. Trends are illustrated for *gcc*, but are representative of those for *gzip, jbb, raytrace*, and *ocean*. Microarchitectural reconfigurations every 0.08M instructions improves *gcc* performance by 59.6 percent and reduces power by 23.25 percent for a 5.3x increase in efficiency. Adaptive optimizations for many short intervals exploit their differing computational requirements. For example, greater power may be consumed for high performance intervals that require additional resources, but the associated high power costs are incurred only for the duration of these particular intervals and do not translate to significantly higher power dissipation for the overall workload. Similarly, low power designs with fewer pipeline resources are often favored for non-computational intensive intervals, thereby reducing power without significantly impacting performance. Thus, higher temporal adaptivity provides locality of power costs and opportunities for power reduction, simultaneously enabling net performance increases and net power reductions.

Lastly, Figure 3LR illustrates a trend observed only for *radiosity* in which performance and power increase together for a net efficiency gain. Microarchitectural reconfigurations every 0.08M instructions improves *radiosity* efficiency by 1.6x derived from a 30.5 and 36.3 percent increase in performance and power, respectively. The $bips^3/w$ metric emphasizes performance over power such that a one percent increase in performance is efficient if power increases by less than approximately three percent. In the case of *radiosity*, performance increases linearly track power increases from greater temporal adaptivity, thereby improving efficiency despite increasing power costs.

For completeness, we note that trends for *mesa* are not represented in these figures. This benchmark exhibited flat or relatively insignificant performance and power changes from increasing temporal adaptivity.

4.2 Utilized Adaptivity

The significant efficiency gains from greater temporal adaptivity suggest diverse requirements for computational resources within a given workload and significant opportunities for adaptivity. We characterize the amount of utilized adaptivity by examining (1) the number of design parameters that adapt between consecutive intervals and (2) the magnitude of these adaptive changes quantified by differences in design parameter values.

Figure 4 plots the cumulative distribution function (CDF) for the number of parameters that change between consecutive intervals. We compare high and low temporal adaptivity using short 0.08M-instruction and long 10.24M-instruction intervals, respectively. The degree of temporal adaptivity impacts the number of parameter changes between consecutive intervals. Taking Figure 4L for a representative workload *raytrace*, 50 and 75 percent of transitions between 0.08M-instruction intervals require design value changes for at most 3 and 6 parameters, respectively. 95 percent of these transitions require changes for fewer than 10 parameters. This high temporal adaptivity smoothes microarchitectural reconfigurations by enabling smaller, intermediate changes for more frequent, shorter intervals.

In contrast, reduced temporal adaptivity (*e.g.*, 10.24M-instruction intervals) degrades this smoothing effect, requiring changes to increase in scope to include more parameters. For example, Figure 4L illustrates a CDF shift where a greater number of interval transitions require changes to more parameters. For 10.24M-instruction intervals, every transition requires at least changes for 2 parameters. 50 and 75 percent of transitions now require changes for at most 6 and 8 parameters, respectively.

[2] See also Figure 7.

However, reduced temporal adaptivity also acts as a low pass filter on microarchitectural reconfiguration, removing short term variations and leaving only the long term trend. This filtering effect reduces the number of parameter changes that optimize specific short intervals with uncommon resource requirements. Figure 4L illustrates this filtering effect where reduced temporal adaptivity eliminates the uncommon 5 percent of transitions with changes to more than 10 parameters. Thus, reduced temporal adaptivity reduces smoothing and increases filtering effects to shift the distribution toward middle range parameter counts. These trends for *raytrace* are comparable to those for 9 of 14 benchmarks.

In contrast, Figure 4R illustrates trends from *gzip* that are representative of *mesa, ocean*, and *radiosity*. *Gzip* is characterized by low reconfiguration diversity with 95 percent of its transitions between 0.08M-instruction intervals requiring changes to at most 8 parameters. The filtering effects from reduced adaptivity dominate and nearly 60 percent of transitions between 10.24M-instruction intervals reconfigure only 1 parameter.

While Figure 4 illustrates the number of parameters that adapt between intervals, Figure 5 quantifies the magnitude of changes in design parameter values. We quantify relative step size by reporting the change in a parameter's value relative to the number of steps in the parameter's range. For example, we consider register file sizes from 40 to 130 entries in increments of 10 entries (9 possible steps). If the microarchitecture changes from 70 to 90 entries between two consecutive workload intervals, the register file has effectively taken 2 steps over 9 possible values for a 0.22 relative step size. As relative step sizes approach one, the interval transition approaches reconfigurations that change a parameter from its minimum to its maximum value.

Figure 5 indicates parameter values change more significantly when a greater number of parameters change simultaneously. Taking *raytrace* as an example,[3] the median relative step size increases from 0.23 to 0.67 as the number of changing parameters increases from 1 to 10. If 7 or more parameters are adapted in an interval transition, 50 percent of these transitions will require changes that span more than half of the possible values in parameters' ranges. Note the relative location of the median within each box shifts upward as the number of changed parameters increases, further indicating shifts in concentration from small step sizes to larger step sizes within asymmetric distributions. Collectively, these trends suggest synergies between parameters as they simultaneously change to ensure no bottlenecks are created. Taken together, Figures 4–5 characterize the utilized adaptivity for representative benchmarks by quantifying the number of parameters that change between consecutive intervals and the magnitude of these changes.

4.3 Potential Impact

Figure 6 summarizes the potential performance, power and efficiency impact of high temporal adaptivity (0.08M-instruction intervals) under the comprehensive spatial adaptivity of Table 2. Figure 6L illustrates diverse performance and power effects across the benchmark suite. Performance increases by up to 62.0 percent (*mcf*) and power decreases by as much as 51.7 percent (*cholesky*). As observed for representative benchmarks in Figure 3, various combinations of performance and power compromises are used to achieve greater efficiency and no single trade-off dominates. Figure 6R illustrates efficiency gains for the benchmark suite with median and maximum efficiency gains of 2.4x and 5.3x, respectively.

Figure 7 reveals the source of these efficiency gains by plotting the performance and power distributions of 0.08M-instruction intervals using boxplots. We compare these distributions against the performance and power of benchmarks running on architectures

[3] The trends for *raytrace* are representative of all benchmarks

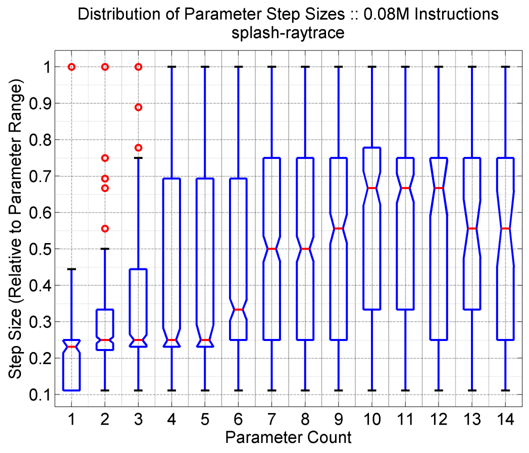


Figure 5. Magnitude of change for parameters that change between consecutive intervals for *raytrace*.

from application-level adaptivity (Table 1) using a green-dashed line. Figure 7L shows this line intersecting boxes at very low points below the 25-th percentile in nearly all cases. The most notable performance increases exceed 20 percent and correlate with these low intersections. These low intersections indicate many 0.08M-instruction intervals can achieve performance much higher than that of application-level adaptivity provided that the microarchitecture can adapt to its requirements for computational resources. For example, the largest performance increases between 59.6 and 62.0 percent are observed for benchmarks where the intersection occurs below the 25-th percentile (*e.g.*, *gcc, mcf, blast*).

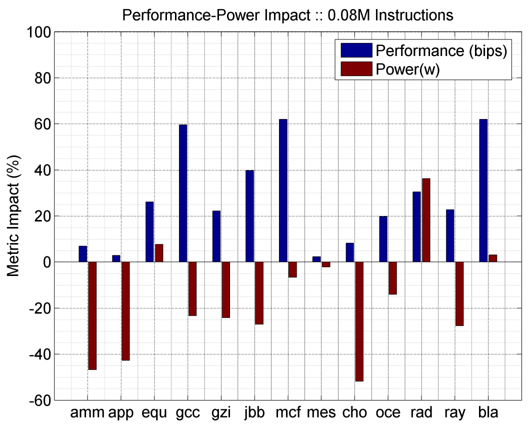
Similarly, Figure 7R shows the power dissipated by the application-level baseline is higher than the power dissipated by individual 0.08M-instruction intervals executing on their optimally adapted configuration. The most significant power reductions between 27.6 and 51.7 percent are observed for benchmarks where the intersection occurs above the 75-th percentile (eg, *ammp, applu, cholesky, raytrace*). Conversely, power increases are observed for intersections at low points in the box. For example, *radiosity* intersects below the median, indicating more than half the intervals adapt to configurations that dissipate more power than dissipated by the baseline, producing a 36.3 percent power increase.

5. Spatial Adaptivity

The previous analysis of temporal adaptivity assumed comprehensive spatial adaptivity where every parameter of Table 2 could be changed to fit each interval's requirements. However, we can also use our framework to assess the impact of reduced spatial adaptivity. A study of varying spatial adaptivity reveals the hardware and control complexity necessary to achieve the efficiencies of Section 4. By identifying the most significant adaptive parameters, we better understand the capabilities required of an adaptive hardware substrate while pruning insignificant parameters from the adaptive space. Such pruning drastically reduces control complexity; an algorithm that reconfigures two or three parameters should be much simpler than an algorithm that reconfigures fifteen parameters. For example, consider the complexity of modern control algorithms for adapting a single parameter: voltage/frequency.

5.1 Reduced Spatial Adaptivity

We consider reducing the number of parameters available for reconfiguration while assuming high temporal adaptivity (*i.e.*, reconfigurations occur every 0.08M instructions). In particular, we identify the three most significant parameters for achieving effi-



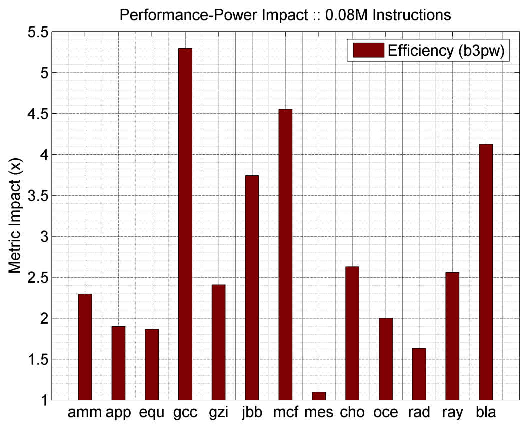
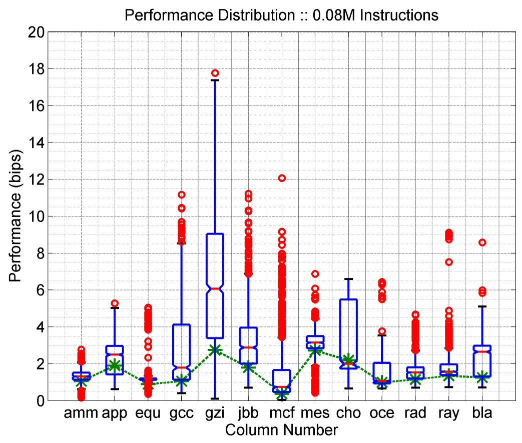


Figure 6. Temporal Adaptivity: Performance, power (L) and efficiency impact (R). Microarchitecture reconfigures every 0.08M instructions.



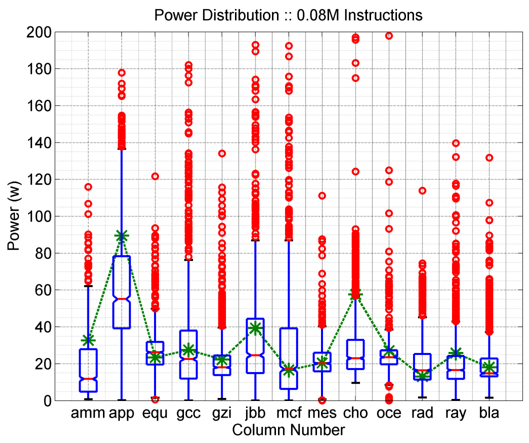


Figure 7. Distribution of performance (L), power (R) under high temporal adaptivity (boxes) versus application-level adaptivity (line).

ciency by exhaustively evaluating the C_k^{15} possible combinations for $k = 1, \ldots, 3$. In particular, there are $C_1^{15} = 15$, $C_2^{15} = 105$, $C_3^{15} = 455$ ways to select one, two, and three parameter(s) for adaptivity, respectively. For each k and each of the C_k^{15} possible combinations for a given k, we repeat the optimization of Section 4 to identify the efficiency-maximizing combination. Although prior work in adaptive microarchitectures often consider two or three parameters, parameters are often chosen prior to any analysis. In contrast, we consider two or three parameters empirically found to be most significant for each benchmark. Thus, this analysis compares comprehensive spatial adaptivity against best case scenarios in limited spatial adaptivity where the most significant parameters are considered.

Table 4 identifies the one, two, and three parameter(s) that provide the greatest efficiency gains from microarchitectural adaptivity. Adaptive pipeline depths are most promising with all benchmarks ranking the depth parameter among the top three. Also significant, but more sparsely ranked, are cache hierarchy parameters. Benchmarks benefit from adaptive caches, suggesting optimal cache sizes and associativities vary significantly across intervals. We consider adaptive `memLat` as a proxy for an adaptive off-chip L3 cache where a more effective lowest level cache will reduce effective memory latency. Logic (*e.g.*, `width`) and associated queues/tables (*e.g.*, `lsq, bp`) are less prominent relative to depth, suggesting interval-to-interval variability is greater for memory access patterns than instruction level parallelism.

From the perspective of implementation, Table 4 has significant implications for design complexity. The highly ranked parameters differ across the benchmark suite and most parameters are highly ranked for at least one benchmark (except the register file and instruction cache parameters). This motivates a hardware substrate for comprehensive adaptivity, especially for the memory hierarchy. Furthermore, these parameter rankings do not necessarily contain hierarchical subsets; a parameter that might be significant when two parameters are considered may be much less significant when three are considered. We observe this scenario for *gcc, cholesky, radiosity*, and *raytrace*.

Figure 8 quantifies the best achievable efficiency when at most three parameters are chosen for adaptivity. The efficiency for each benchmark is reported under its optimal parameter subset as shown in Table 4. Most benchmarks require only a few adaptive parameters to achieve a high fraction of potential efficiency gains. On average, benchmarks are able to achieve 60.3, 71.1, and 77.3 percent of 15-parameter efficiency as the number of adaptive parameters increases from one to three (medians of 61.4, 76.4, 82.3 percent). However, as illustrated in Table 4, the optimal choice of two or three parameters needed to deliver such efficiency may differ substantially from benchmark. This variation makes identifying any minimal adaptive microarchitectural substrate very difficult.

Four benchmarks, *ammp, gcc, jbb* and *mcf*, are notable for achieving relatively small fractions of potential. Adapting their three optimally chosen parameters only produces efficiency be-

		amm	app	equ	gcc	gzi	jbb	mcf	mes	cho	oce	rad	ray	bla
S_1	depth	1	2	1	1	1	1	1	1	1	2	1	1	2
S_2	width									2	3	2		
S_3	bp													
S_4	lsq											3		
S_5	reg													
S_6	resv											2*		
S_7	i1Size													
S_8	i1Assoc													
S_9	d1Size					2			2				2	
S_{10}	d1Assoc						3			3				
S_{11}	d1Lat						2							3
S_{12}	l2Size		3			3							3	
S_{13}	l2Assoc	3		2	2*			3		2*				
S_{14}	l2Lat			3	2			2						
S_{15}	memLat	2	1		3				3		1		2*	1

Table 4. Choice of $k = 1, \ldots, 3$ parameters that maximize adaptive efficiency gains. * denotes parameters that became less significant with additional adaptivity (*e.g.*, 2* for *gcc* `l2Assoc` indicates it was among the 2, but not the 3, most significant parameters.)

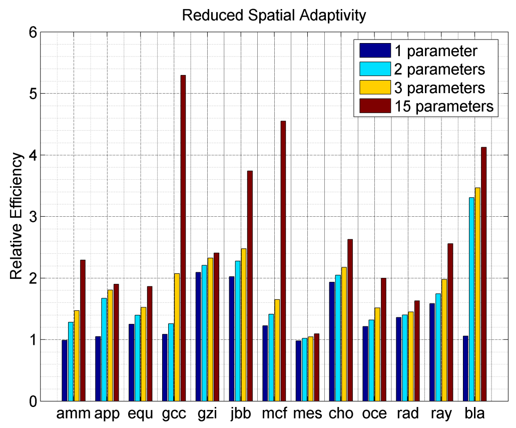


Figure 8. Reduced versus comprehensive spatial adaptivity.

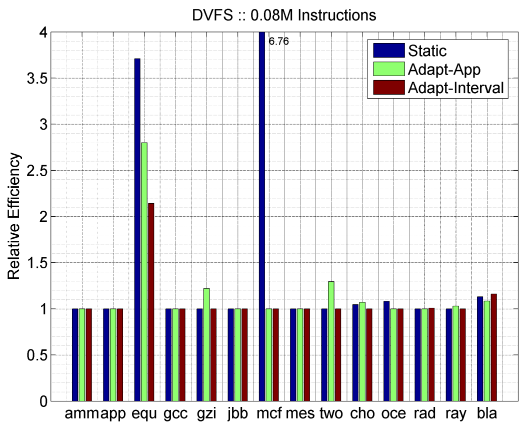


Figure 9. Additional efficiency from DVFS applied to various degrees of spatial adaptivity from none(static), high-spatial/low-temporal (Adapt-App), and high-spatial/high-temporal (Adapt-Interval). Each bar is normalized to the corresponding level of spatial adaptivity without DVFS.

tween 36.2 and 66.3 percent of potential. We observe steady, but modest, efficiency benefits from increasing the number of adaptive parameters from one to three. Given that results are reported for the three parameters exhaustively found to be the most significant, additional parameters are also likely to produce only modest and incremental efficiency increases toward the 15-parameter potential. Indeed, a subsequent search for a fourth significant parameter for these benchmarks further increases efficiency by a modest 8 to 10 percent, putting efficiency between 39.4 and 71.7 percent of potential. Thus, we expect much more comprehensive spatial adaptivity is required to close the gap for these benchmarks, drawing on incremental efficiency improvements from additional parameters as well as synergies between these parameters.

In summary, most benchmarks are able to leverage a modest number of parameters to achieve a significant fraction of the potential efficiency from 15-parameter adaptivity. However, the optimal choice of parameters differs significantly across benchmarks and a comprehensive adaptive hardware substrate would be needed to fully realize these benefits. Lastly, a few benchmarks may require significantly more than three parameters to achieve efficiency closer to the projected bound.

5.2 Voltage/Frequency Scaling

Dynamic voltage and frequency scaling (DVFS) adapts pipeline sensitivity to memory by, for example, slowing computation and reducing power dissipation during long memory stalls. We assess potential interactions between structural adaptivity and DVFS by adding voltage/frequency to the space of adaptive parameters. Specifically, we optimize each interval's microarchitectural structures and then further tune voltage/frequency, identifying the lowest voltage/frequency that maximizes $bips^3/w$. Since DVFS changes the relative clock speeds between the processor core and memory, we model its effects by scaling off-chip memory latency before evaluating regression models for performance and power. Since the performance models were derived for a baseline frequency, we must further scale regression predicted *bips* to account for the frequency change. Similarly, we scale regression predicted power to account for voltage and frequency changes.

Figure 9 quantifies additional efficiency gains from DVFS when applied to various degrees of temporal adaptivity. Most benchmarks do not realize significant $bips^3/w$ efficiency gains. This is due, in part, to the choice of efficiency metric. Recall that $bips^3/w$ is a voltage and frequency invariant metric. As shown in Equation (1), DVFS will produce gains for $bips^3/w$ only if it significantly improves pipeline throughput. Specifically, assume baseline throughput ipc_0, voltage V_0, and frequency f_0. Frequency and voltage scaling may impact throughput ipc_1 and scales voltage/frequency by Δ. Since the Δ scaling cancels in the $bips^3/w$ metric, only throughput effects remain.

$$\frac{bips_1^3}{w_1}\frac{w_0}{bips_0^3} \propto \frac{ipc_1^3 \cdot (f_0 \cdot \Delta)^3}{(V_0 \cdot \Delta)^2 \cdot (f_0 \cdot \Delta)} \frac{V_0^2 \cdot f_0}{ipc_0^3 \cdot f_0^3} = \left(\frac{ipc_1}{ipc_0}\right)^3 \quad (1)$$

Figure 9 shows, for benchmarks that benefit from DVFS, the greatest gains arise when DVFS is applied to a static, general-purpose, POWER4-like architecture with no resource adaptivity. For comparison, Figure 9 also includes the effects of DVFS when applied to the comprehensive resource adaptivity of Table 2 at the application-level (81.92M instructions) and interval level (0.08M instructions). Each of these three bars is normalized to the static, application-adaptive, or interval-adaptive architecture, respectively, with no DVFS. Benchmarks *equake* and *mcf* realize gains of 3.7x and 6.8x, respectively, relative to a static architecture with no DVFS. Microarchitectural resource adaptivity at the application-level significantly degrades the additional benefits of DVFS to 2.8x and 1.0x for these benchmarks. *Equake* efficiency gains are further degraded to 2.1x when sub-application, interval-level adaptivity is introduced. More generally, this data suggests DVFS becomes progressively less effective as an additional tuning parameter when combined with increasing spatial adaptivity. The bulk of ipc throughput gains are likely extracted from microarchitectural resource tuning and any additional throughput gains from voltage and frequency scaling on the tuned architecture are likely modest and incremental. Since Equation (1) indicates such throughput gains are required to make DVFS effective, DVFS is likely to have a diminished role in a microarchitecture capable of comprehensive spatial adaptivity.

6. Related Work

Broadly, prior work differ in their study of particular adaptive structures and/or control algorithms. These prior studies consider either high temporal, low spatial adaptivity or low spatial, high temporal adaptivity. Each study limited the adaptive scope to include limited combinations of design values for at most two or three microarchitectural parameters and implemented a heuristic control algorithm to predict the best configuration from these limited choices. Most papers cite or imply computational costs as the cause for these restrictions. In contrast, we consider both high temporal and spatial adaptivity and leverage advances in statistical inference and optimization to control computational costs.

Hardware Mechanisms. Various adaptivity studies have examined many different microarchitectural structures, which collectively represent all major design parameters. However, each study considered only a very small subset of these parameters whereas we adapt simultaneously all major microarchitectural parameters. Albonesi, *et al.*, inspired this work by describing buffering mechanisms that enable adaptivity in data caches and instruction queues [1]. Following work by Mai, *et al.*, and Balasubramonian, *et al.*, separately expanded on these ideas for the memory hierarchy [24, 4]. Folegnani considered optimizing issue logic and queues while Ponomarev studied adaptivity for the reorder buffer and various queues [11, 27]. Adaptive pipeline depth and width have also been considered for energy and reliability [8, 17, 18, 23, 31].

Control Algorithms. Control algorithms and heuristics have been proposed to trigger adaptive reconfiguration at various granularities. Ponomarev, *et al.*, uses prior occupancy to predict future occupancy of various queues every 2,000 cycles [27]. Dhodapkar, *et al.*, assess the similarity of working sets to trigger reconfiguration [7]. Hughes *et al.* consider adapting resources for multimedia workloads based on frame type while Huang, *et al.*, consider adapting resources for code sections defined by subroutine boundaries [17, 16]. Collectively, these prior works provide heuristics for dynamically determining optimal configurations and have been demonstrated for a limited number of adaptive parameters. These heuristics apply concepts from phase analysis to identify an effective degree of temporal adaptivity. Our framework may be extended to include phase information, but we currently use phase-oblivious adaptive intervals. In further contrast to prior heuristics, we use an oracle-based assessment of efficiency gains to provide an optimistic point of reference under best case scenarios.

Predictive Modeling. Ipek, *et al.*, and Joseph, *et al.*, separately predict the performance of design spaces with automated artificial neural networks (ANN) trained by gradient descent and predicted by nested weighted sums [9, 20]. Ipek, *et al.*, use sampling with feedback to identify samples most likely to improve accuracy while Joseph, *et al.*, use Latin hypercube sampling to maximize coverage. Our approach requires greater statistical analysis when constructing spline-based regression models but may be more computationally efficient, numerically solving and evaluating linear systems for training and prediction, respectively [21].

Optimization. Eyerman, *et al.*, combine synthetic trace simulation with heuristics to search for global optima within a design space [10]. These authors find variants of gradient ascent and genetic algorithms most effective. Unlike prior work, we describe the tunable parameters in genetic algorithms and describe the process to identify effective parameters. Furthermore, prior work combines optimization heuristics with statistical simulation while we combine genetic algorithms with regression models. This difference has significant implications for computational cost. Eyerman, *et al.*, require between 900 and 1,000 simulations per optimization problem. However, these simulations are specific to a given optimization problem since they simulate design points along a particular path taken to the estimate of a particular metric's optimum. In contrast, our regression models require 500 simulations per design space since they may be formulated once and used in multiple invocations of the genetic algorithm.

7. Conclusions and Future Directions

The transfer of best practices from statistics, machine learning, and combinatorial optimization has revitalized microarchitectural analysis. A synergistic combination of sampling, modeling, and optimization enables, for the first time, a comprehensive assessment of efficiency benefits from microarchitectural adaptivity. This assessment reveals significant benefits from high temporal adaptivity. Furthermore, we find these gains are most accessible using a hardware substrate with comprehensive spatial adaptivity due to differing adaptive requirements across applications.

Given this quantification of adaptivity benefits, a rigorous assessment of costs and complexities is logical future work. Overhead and complexity will necessarily increase in the design of adaptive hardware substrates. In addition to these hardware costs, control algorithm effectiveness and overheads must be considered. The framework used to identify potential adaptivity benefits in this work will not necessarily be the same framework used to implement online control mechanisms. In practice, there may be several options for implementing fine-grained adaptivity, including:

- A hardware accelerated controller might derive off-line and evaluate on-line regression models. This is a relatively heavy-weight solution likely suitable for simultaneously adapting many parameters in a complex optimization framework.
- A heuristic based on hardware counters might tune resource allocations based on prior utilization (*e.g.*, [27]). This is a relatively light-weight solution likely suitable for adapting a limited number of parameters.
- A software mechanism might identify optimal designs for each subroutine via off-line profiling before applying them on-line (*e.g.*, [16]). This is a relatively light-weight solution for simul-

taneously adapting many parameters at the temporal granularity of subroutines.

Further study of the compromises between temporal adaptivity and control logic overhead is needed. Ultimately, the significant efficiency gains identified in this work should motivate further analysis of the implementation costs for these benefits.

Acknowledgments

This work is supported by NSF grant CCF-0048313 (CAREER), Intel, and IBM. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, Intel or IBM.

References

[1] D. Albonesi. Dynamic IPC/clock rate optimization. In *International Symposium on Computer Architecture*, June 1998.

[2] D. Albonesi, R. Balasubramonian, S. Dropsho, S. Dwarkadas, E. Friedman, M. Huang, V. Kursun, G. Magklis, M. Scott, G. Semezaro, P. Bose, A. Buyuktosunoglu, P. Cook, and S. Schuster. Dynamically tuning processor resources with adaptive processing. *IEEE Computer*, December 2003.

[3] D. Bader, Y. Li, T. Li, and V. Sachdeva. Bioperf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications. In *IEEE International Symposium on Workload Characterization*, October 2005.

[4] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *International Symposium on Microarchitecture*, December 2000.

[5] D. Brooks, P. Bose, V. Srinivasan, M. Gschwind, P. G. Emma, and M. G. Rosenfield. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM Journal of Research and Development*, 47(5/6), Oct/Nov 2003.

[6] D. Brooks and et. al. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, Nov/Dec 2000.

[7] A. Dhodapkar and J. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *International Symposium on Computer Architecture*, June 2002.

[8] A. Efthymiou and J. Garside. Adaptive pipeline structures for speculation control. In *International Symposium on Asynchronous Circuits and Systems*, May 2003.

[9] E.Ipek, S.A.McKee, B. de Supinski, M. Schulz, and R. Caruana. Efficiently exploring architectural design spaces via predictive modeling. In *Architectural support for programming languages and operating systems*, October 2006.

[10] S. Eyerman, L. Eeckhout, and K. D. Bosschere. Efficient design space exploration of high performance embedded out-of-order processors. In *Design, Automation, and Test in Europe*, March 2006.

[11] D. Folegnani and A. Gonzalez. Energy-effective issue logic. In *International Symposium on Computer Architecture*, June 2001.

[12] G. Givens and J. Hoeting. *Computational Statistics*. Wiley, 2005.

[13] S. Gochman, R. Ronen, and et al. The intel pentium m processor: Micorarchitecture and performance. *Intel Technology Journal*, 7(2), May 2003.

[14] F. Harrell. *Regression modeling strategies*. Springer, 2001.

[15] J. Henning. Spec cpu2000: Measuring cpu performance in the new millenium. *IEEE Computer*, July 2000.

[16] M. Huang, J. Renau, and J. Torrellas. Positional adaptation of processors: Application to energy reduction. In *International Symposium on Computer Architecture*, June 2003.

[17] C. Hughes, J. Srinivasan, and S. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *International Symposium on Microarchitecture*, December 2001.

[18] E. Ipek, M. Kirman, N. Kirman, and J. Martinez. Core fusion: Accommodating software diversity in chip multiprocessors. In *International Symposium on Computer Architecture*, June 2007.

[19] V. Iyengar, L. Trevillyan, and P. Bose. Representative traces for processor models with infinite cache. In *International Symposium on High Performance Computer Architecture*, February 1996.

[20] P. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. A predictive performance model for superscalar processors. In *International Symposium on Microarchitecture*, December 2006.

[21] B. Lee and D. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2006.

[22] B. Lee and D. Brooks. Illustrative design space studies with microarchitectural regression models. In *International Symposium on High-Performance Computer Architecture*, February 2007.

[23] X. Liang and D. Brooks. Mitigating the impact of process variations on cpu register file and execution units. In *International Symposium on Microarchitecture*, December 2006.

[24] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. Dally, and M. Horowitz. Smart memories: A modular reconfigurable architecture. In *International Symposium on Computer Architecture*, June 2000.

[25] M. Moudgill, J. Wellman, and J. Moreno. Environment for PowerPC microarchitecture exploration. *IEEE Micro*, 19(3):9–14, May/June 1999.

[26] A. Phansalkar, A. Joshi, L. Eeckhout, and L. John. Measuring program similarity: experiments with SPEC CPU benchmark suites. In *International Symposium on Performance Analysis of Systems and Software*, March 2005.

[27] D. Ponomarev, G. Kucuk, and K. Ghose. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *International Symposium on Microarchitecture*, December 2001.

[28] R Development Team. *R Language Definition*.

[29] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[30] P. Shivakumar and N. Jouppi. An integrated cache timing, power, and area model. In *Technical Report 2001/2, Compaq Computer Corporation*, August 2001.

[31] A. Tiwari, S. Sarangi, and J. Torrellas. ReCycle: Pipeline adaptation to tolerate process variation. In *International Symposium on Computer Architecture*, June 2007.

[32] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *International Symposium on Computer Architecture*, June 1995.

[33] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *International Symposium on Computer Architecture*, June 2003.

[34] V. Zyuban. Inherently lower-power high-performance superscalar architectures. In *Ph.D. Thesis, University of Notre Dame*, March 2000.

[35] V. Zyuban, D. Brooks, V. Srinivasan, M. Gschwind, P. Bose, P. Strenski, and P. Emma. Integrated analysis of power and performance for pipelined microprocessors. *IEEE Transactions on Computers*, Aug 2004.