Applied Inference: Case Studies in Microarchitectural Design

BENJAMIN C. LEE
Stanford University
DAVID BROOKS
Harvard University

We propose and apply a new simulation paradigm for microarchitectural design evaluation and optimization. This paradigm enables more comprehensive design studies by combining spatial sampling and statistical inference. Specifically, this paradigm (1) defines a large, comprehensive design space, (2) samples points from the space for simulation, and (3) constructs regression models based on sparse simulations. This approach greatly improves the computational efficiency of microarchitectural simulation and enables new capabilities in design space exploration.

We illustrate new capabilities in three case studies for a large design space of approximately 260,000 points: (1) Pareto frontier, (2) pipeline depth, and (3) multiprocessor heterogeneity analyses. In particular, regression models are exhaustively evaluated to identify Pareto optimal designs that maximize performance for given power budgets. These models enable pipeline depth studies in which all parameters vary simultaneously with depth, thereby more effectively revealing interactions with non-depth parameters. Heterogeneity analysis combines regression based optimization with clustering heuristics to identify efficient design compromises between similar optimal architectures. These compromises are potential core designs in a heterogeneous multicore architecture. Increasing heterogeneity can improve $bips^3/w$ efficiency by as much as 2.4x, a theoretical upper bound on heterogeneity benefits that neglects contention between shared resources as well as design complexity. Collectively these studies demonstrate regression models' ability to expose trends and identify optima in diverse design regions, motivating the application of such models in statistical inference for more effective use of modern simulator infrastructure.

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

Additional Key Words and Phrases: Microarchitecture, Simulation, Statistics, Regression

1. INTRODUCTION

Microarchitectural design space exploration is a computationally expensive combinatorial problem, requiring a large number of detailed simulations for performance and power estimation. Furthermore, recent industry trends suggest a number of new challenges as designers consider the multiprocessor domain. Designers are increasingly targeting differentiated market segments each with particular metric emphases. For example, designs might implement different compromises between latency, throughput, power, and temperature depending on application and operating cost factors specific to each market segment. Thus, increasing market differentiation implies increasing *metric diversity*, which further implies more interesting optimization objectives and constraints.

Increasing metric diversity will also lead to non-intuitive design optima that potentially occupy very different regions of the design space. *Design diversity* has already been observed in the set of interesting microarchitectures considered for industry implementation. For example, the IBM POWER5, Intel Pentium 4 and Sun UltraSPARC T1 occupy very different parts of the design space. POWER5 implements relatively wide pipelines, Pentium4 implements relatively deep pipelines, and UltraSPARC T1 cores are relatively simple in-order pipelines [Intel Corporation 2001; Kongetira et al. 2005; Sinharoy et al. 2005].

Metric and design diversity illustrate the need for scalable techniques to more comprehensively explore a space and assess the relative advantages of very different design options. Current approaches to design evaluation are often inefficient and ad hoc due to the significant computational costs of modern simulator infrastructure. The detail in modeling microprocessor execution result in long simulation times. Designers circumvent these challenges by constraining the design space considered (often using intuition or experience) and reducing the size of simulator inputs via instruction trace sampling. However, by pruning the design space with intuition before a study, the designer risks obtaining conclusions that simply reinforce prior intuition and may not generalize to the broader space.

Instruction trace sampling, while effective in reducing the simulator input size by orders of magnitude, only impacts per simulation costs and does not address the number of simulations required in a comprehensive design space study. Trace sampling alone is insufficient as per simulations costs decrease linearly, albeit by a large factor, while the number of potential simulation points increase exponentially with the number of design parameters. This exponential increase is currently driven by the design of multi-core, multi-threaded microprocessors targeting diverse metrics including single-thread latency, throughput for emerging parallel workloads, and energy. These trends will also lead to more variety in the set of viable and interesting designs (*e.g.*, simpler, less aggressive cores), thereby requiring a more thorough exploration of a comprehensive design space.

Techniques in statistical inference are necessary for a scalable simulation approach that addresses these fundamental challenges, modestly reducing detail for substantial gains in speed and tractability. Even for applications in which obtaining extensive measurement data is feasible, efficient analysis of this data often lends itself to statistical modeling. Such an approach typically requires an initial data set for model formulation or training. The model responds to predictive queries by leveraging correlations in the original data for inference.

Regression modeling is integrated into a simulation paradigm designed to increase the information content for a given simulation cost (Section 2). This paradigm specifies a large, comprehensive design space, selectively simulates a modest number of designs sampled from that space, and more efficiently leverages that simulation data using regression models to identify trends and optima. Design space sampling and statistical inference enables the designer to perform a tractable number of simulations independent of design space size or resolution. Applying this simulation paradigm, we sample 1,000 points uniformly at random from a design space of 375,000 points for simulation. Given these samples, we formulate non-linear regression models for microarchitectural performance and power prediction (Section 3), achieving median error rates of 7.2 and 5.4 percent, respectively, relative to simulation. We apply the derived models to comprehensively explore a design space for three optimization problems:

(1) **Pareto Frontier Analysis:** We comprehensively characterize the design space, constructing a regression predicted Pareto frontier in power delay coordinates. We find predictions for Pareto optima exhibit median errors comparable to those for the broader space (Section 4).
(2) **Pipeline Depth Analysis:** We compare a constrained pipeline depth study against an enhanced study that varies all parameters simultaneously via regression modeling. We find constrained sensitivity studies may not generalize when many other design parameters are held at constant values. Furthermore, such generalized studies more effectively reveal interactions between design parameters (Section 5).
(3) **Multiprocessor Heterogeneity Analysis:** We identify efficiency maximizing architectures for each benchmark via regression modeling and cluster these architectures to identify design compromises. We quantify the power-performance benefits from varying degrees of core heterogeneity, quantifying a theoretical upper bound on $bips^3/w$ efficiency gains. We find modest heterogeneity may provide substantial efficiency benefits relative to homogeneity (Section 6).

For each case study, we provide an assessment of predictive error and sensitivity of observed trends to such error. Collectively these studies demonstrate the applicability of regression models for performance and power prediction in practical design space optimization.

2. EXPERIMENTAL METHODOLOGY

We use Turandot, a generic and parameterized, out-of-order, superscalar processor simulator [Moudgill et al. 1999]. Turandot is enhanced with PowerTimer to obtain power estimates based on circuit-level power analyses and resource utilization statistics [Brooks et al. 2003]. 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. pipeline width increases, using scaling factors derived for an architecture with clustered functional units [Zyuban and Kogge 2001]. Cache power and latencies scale with array size according to CACTI [Tarjan et al. 2006]. We do not leverage any particular feature of the simulator and our framework may be generally applied to other simulation frameworks. We measure billions of instructions per second (bips) and watts (w).

We use R, an open-source software environment for statistical computing, to script and automate statistical analyses. Within this environment, we use the Hmisc and Design packages [Harrell 2001].

2.1 Benchmark Suite

We consider SPEC JBB, a Java server benchmark, and eight compute intensive benchmarks from SPEC CPU 2000 (*ammp, applu, equake, gcc, gzip, mcf, mesa, twolf*). We report experimental results based on PowerPC traces of these benchmarks. Traces used in this study were sampled from the full reference input set to obtain 100 million instructions per benchmark program using graph-based heuristics to identify representative basic blocks [Iyengar et al. 1996]. Systematic validation was performed to compare the sampled traces against the full traces to

	Set	Parameters	Measure	Range	$\lvert S_i \rvert$
S_1	Depth	depth	FO4	9::3::36	10
S_2	Width	width	decode b/w	2,4,8	3
		L/S queue	entries	15::15::45	
		store queue	entries	14::14::42	
		functional units	count	1,2,4	
S_3	Physical Registers	general purpose	count	40::10::130	10
		floating-point	count	40::8::112	
		special purpose	count	42::6::96	
S_4	Reservation Stations	branch	entries	6::1::15	10
		fixed-point	entries	10::2::28	
		floating-point	entries	5::1::14	
S_5	I-L1 Cache	i-L1 cache size	KB	16::2x::256	5
S_6	D-L1 Cache	d-L1 sache size	KB	8::2x::128	5
S_7	L2 Cache	L2 cache size	MB	0.25::2x::4	5

Table I. Design Space :: range i::j::k denotes values from i to k in steps of j

ensure accurate representation. Our benchmark suite is representative of larger suites frequently used in the microarchitectural research community [Phansalkar et al. 2005]. Although specific conclusions of our design space studies may differ with different benchmarks, we do not leverage any particular benchmark feature in model formulation and our framework may be generally applied to other workloads.

2.2 Simulation Paradigm

Challenges in microarchitectural design motivate a new simulation paradigm that (1) specifies a large, comprehensive design space, (2) selectively simulates a modest number of designs sampled from that space, and (3) more efficiently leverages that simulation data using techniques in statistical inference to identify trends and optima. This paradigm begins with a comprehensive design space definition that considers many high-resolution parameters simultaneously. Given this design space, we apply techniques in *spatial sampling* to obtain a small fraction of design points for simulation. Spatial sampling allows us to decouple the high-resolution of the design space from the number of simulations required to identify a trend within it. Lastly, we construct regression models using simulations of these sparsely sampled designs to enable predictions for metrics of interest. The predictive ability and computational efficiency of these models enables new capabilities in microarchitectural design optimization.

The first part of this paradigm is implemented with the design specification of Table I.[1] This table identifies seven groups of parameters varied simultaneously. The range of values considered are specified by sets, $S_1, \ldots, S_7$. The Cartesian product of these sets, $S = \prod_{i=1}^{7} S_i$, defines the design space that contains $|S| = \prod_{i=1}^{7} |S_i| = 375,000$ points.

The second part of the paradigm requires sampling design points for simulation. Spatial sampling provides observations from the full range of parameter values and enables identification of trade-offs between parameter sets. An arbitrarily large number of values may be included in each set S_i, thereby increasing design space resolution, since the number of simulations is decoupled from set cardinality via random sampling. We sample *uniformly at random* (UAR) from the design space

[1]FO4 delay is defined as the delay of one inverter driving four copies of an equally sized inverter. When logic and latch overhead per pipeline stage is measured in terms of FO4 delay, deeper pipelines have smaller FO4 delays.

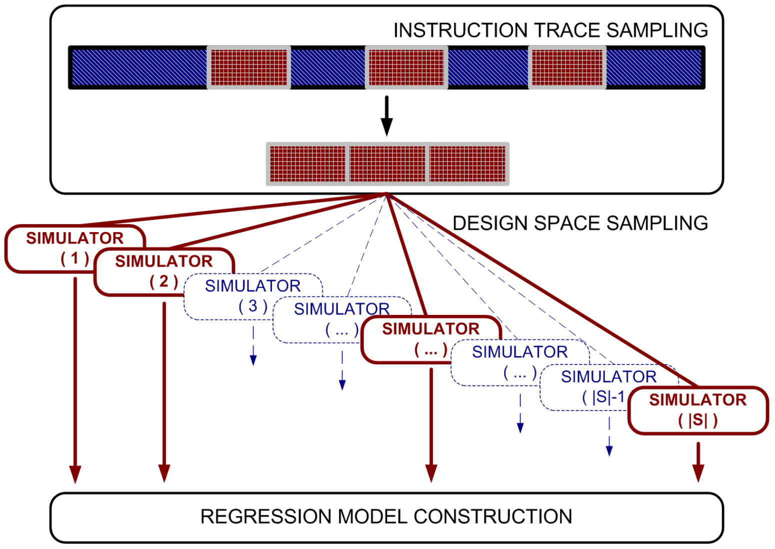


Fig. 1. Simulation Paradigm :: temporal and spatial sampling

S to obtain unbiased observations and to control the exponentially increasing number of design points as parameter count and resolution increases [Lee and Brooks 2006]. Spatial sampling complements existing techniques in trace sampling [Sherwood et al. 2002; Wunderlich et al. 2003]. Figure 1 illustrates a combination of trace and spatial sampling to reduce the costs per simulation and the number of required simulations, respectively.

2.3 Alternative Sampling Strategies

For comparison, other sampling strategies have been proposed to increase the predictive accuracy of machine learning models for the microarchitectural design space. These techniques generally increase sample coverage of the design space or emphasize samples considered more important to model accuracy.

—**Weighted sampling** is a strategy for emphasizing samples in particular design regions given samples from the broader space. Emphasized samples are weighted to increase their influence during model training. Weighted sampling may improve model accuracy for design regions known to exhibit greater error.

—**Regional sampling** also emphasizes samples from particular design regions given samples from the broader space. Instead of using a continuous range of weights, this approach specifies a region of interest and excludes undesired samples during model training (effectively binary weights). Regional sampling might be used to construct localized models from samples collected uniformly at random from the entire space. This approach may be necessary if regions of interest are unknown prior to sampling but become known after exploratory data analysis [Lee and Brooks 2006].

—**Adaptive sampling** estimate model error variances for each sampled design. Samples with larger variances are likely poorly predicted and including such samples for model training may improve accuracy. These error-prone samples are iteratively added to the training set, with each iteration choosing a sample with large error variance and most different from those already added [Ipek et al. 2006].

—**Latin hypercube sampling and space-filling** seek to maximize design space coverage. Hypercube sampling guarantees each parameter value is represented in the sampled designs. Space-filling metrics are used to select the most uniformly distributed sample from the large number of hypercube samples that exist for any given design space [Joseph et al. 2006b].

While these techniques seek to maximize design space coverage and improve the accuracy of models constructed from the resulting samples, they are also more complex and computationally expensive. Determining inclusion in regional sampling requires distances computed between all collected samples, an expensive operation in high dimensions that must be performed for each region of interest. UAR sampling is parallel, but adaptive sampling introduces a feedback loop that limits this parallelism. Hypercube sampling and space-filling techniques guarantee sample properties that are only approximated by uniform at random sampling, but such a guarantee increases sampling complexity. Collectively, these sampling strategies provide options for improving model accuracy.

3. REGRESSION MODELING

Regression modeling is the third part of the simulation paradigm. We apply regression modeling to efficiently obtain estimates of microarchitectural design metrics, such as performance and power. We apply a general class of models in which a response is modeled as a weighted sum of predictor variables plus random noise. Since basic linear estimates may not adequately capture nuances in the response-predictor relationship, we also consider more advanced techniques to account for potential predictor interactions and non-linear relationships. A statistically robust derivation applies hierarchical clustering, association and correlation analysis, and residual analysis. Lastly, we assess model effectiveness and predictive ability. This article surveys the derivation with further detail available in prior work [Lee and Brooks 2006].

3.1 Model Formulation

For a large universe of interest, suppose we have a subset of n observations for which values of the response and predictor variables are known. Let $\vec{y} = y_1, \ldots, y_n$ denote observed responses. For a particular point i in this universe, let y_i denote its response and $\vec{x_i} = x_{i,1}, \ldots, x_{i,p}$ denote its p predictors. Let $\vec{\beta} = \beta_0, \ldots, \beta_p$ denote regression coefficients used in describing the response as a linear function of predictors plus a random error e_i as shown in Equation (1). The e_i are assumed independent random variables with zero mean and constant variance; $E(e_i) = 0$ and $Var(e_i) = \sigma^2$. Transformations f and $\vec{g} = g_1, \ldots, g_p$ may be applied to the response and predictors, respectively, to improve model fit by stabilizing a non-constant error variance or accounting for non-linear predictor-response relationships.

$$f(\hat{y}_i) = \beta_0 + \sum_{j=1}^{p} \beta_j g_j(x_{ij}) + e_i \tag{1}$$

Fitting a regression model to observations, by determining the $p+1$ coefficients in $\vec{\beta}$, enables response prediction. The method of least squares is commonly used to identify the best-fitting model by minimizing $S(\vec{\beta})$, the sum of squared deviations of predicted responses given by the model from actual observed responses. $S(\vec{\beta})$ may be minimized by solving a system of $p+1$ partial derivatives of S with respect to β_j, $j \in [0,p]$. The solutions to this system are estimates of the coefficients.

$$S(\beta_0, \ldots, \beta_p) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2}$$

In the context of microprocessor design, the response y represents a metric of interest (*e.g.*, performance or power) and the predictors x represent design parameter values (*e.g.*, pipeline depth or L2 cache size).

3.2 Predictor Interaction

In some cases, the effect of two predictors x_1 and x_2 on the response cannot be separated; the effect of x_1 on y depends on the value of x_2 and vice versa. The interaction between two predictors may be modeled by constructing a third predictor $x_3 = x_1 x_2$ to obtain $y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + e_i$. Modeling predictor interactions in this manner makes it difficult to interpret β_1 and β_2 in isolation. After simple algebraic manipulation to account for interactions, we find $\beta_1 + \beta_3 x_2$ is the expected change in y per unit change in x_1 for a fixed x_2. The difficulties of these explicit interpretations of $\vec{\beta}$ for more complex models lead us to prefer more indirect interpretations of the model via its predictions.

We draw on domain-specific knowledge to specify predictor interactions. For example, domain knowledge provides Equation (3), which states the speedup from pipelining increases with pipeline depth and decreases with the number of stalls per cycle [Hennessy and Patterson 2003]. Such insight leads to a relationship between depth and cache structure, which in turn leads to the interaction specified by Equation (3). Suppose x_1 is pipeline depth and x_2 is L2 cache size. As the L2 cache size decreases, memory stalls per instruction will increase and instruction throughput gains from pipelining will be impacted.

$$Speedup_{pipe} = \frac{Depth_{pipe}}{Stalls_{pipe}} \propto Depth_{pipe}Cache \propto x_1 x_2 \tag{3}$$

Similarly, we might expect pipeline width to interact with register file and queue sizes. We also specify interactions between sizes of adjacent cache levels in the memory hierarchy (*e.g.*, L1 and L2 cache size interaction). Appendix A illustrates the specification of these interactions in the R scripting language. We do not attempt to capture all possible interactions, but seek to characterize the most significant effects through domain knowledge. While automated approaches to parameter selection (*e.g.*, step-wise regression [Harrell 2001]) might be used, the accuracy of our models suggest our high-level representation of interactions is sufficient for effective performance and power modeling [Lee and Brooks 2006].

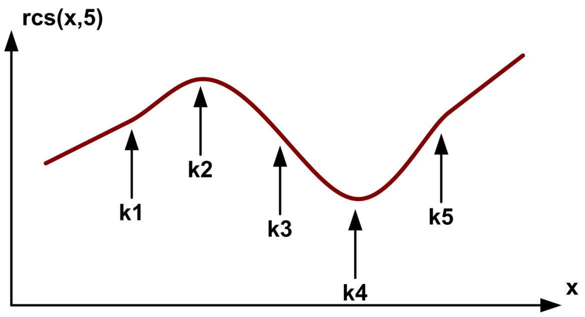


Fig. 2. Restricted Cubic Spline :: 5 knots with linear tails

3.3 Non-Linearity

Basic linear regression models assume the response behaves linearly in all predictors. This assumption is often too restrictive (*e.g.*, power increases superlinearly with pipeline depth) and several techniques for capturing non-linearity may be applied. The most simple of these techniques is a polynomial transformation on predictors suspected of having a non-linear correlation with the response. However, polynomials have undesirable peaks and valleys that are determined by the degree of the polynomial and are difficult to manipulate. Furthermore, a good fit in one region of the predictor's values may unduly impact the fit in another region of values. For these reasons, we consider splines a more effective technique for modeling non-linearity.

Spline functions are piecewise polynomials used in curve fitting [Harrell 2001]. The function is divided into intervals defining multiple different continuous polynomials with endpoints called *knots*. The number of knots can vary depending on the amount of available data for fitting the function, but more knots generally leads to better fits. Relatively simple linear splines may be inadequate for complex, highly curved relationships. Splines of higher order polynomials may offer better fits and cubic splines have been found particularly effective [Stone and Koo 1986]. Unlike linear splines, cubic splines may be made smooth at the knots by forcing the first and second derivatives of the function to agree at the knots. However, cubic splines may have poor behavior in the tails before the first knot and after the last knot. Restricted cubic splines that constrain the function to be linear in the tails are often better behaved (Figure 2).

The choice and position of knots are variable parameters when specifying non-linearity with splines. Stone has found the location of knots in a restricted cubic spline to be much less significant than the number of knots [Stone and Koo 1986]. Placing knots at fixed quantiles of a predictor's distribution is a good approach in most datasets, ensuring a sufficient number of points in each interval. As the number of knots increases, flexibility improves at the risk of over-fitting the data. In many cases, four knots offer an adequate fit of the model and is a good compromise between flexibility and loss of precision from over-fitting [Harrell 2001]. We vary the number of knots to explore the trade-offs between flexibility and fit, finding rapidly diminishing marginal returns in fit from more than five knots that do not justify the larger number of terms in the model.

The strength of a predictor's correlation with the response will determine the number of knots in the transformation. A lack of fit for predictors highly correlated with the response will have a greater negative impact on accuracy and we assign more knots to such predictors. As shown in Appendix A, predictors with stronger performance relationships will use 4 knots (*e.g.*, pipeline depth and register file size) and those with weaker relationships will use 3 knots (*e.g.*, latencies, cache sizes) [Lee and Brooks 2006].

Splines are non-linear transformations on predictors, but transformations may also be needed for the response. A square-root transformation on the response ($f(y) = \sqrt{y}$) is particularly effective for reducing error variance in our performance models. Similarly, a log transformation ($f(y) = log(y)$) more effectively captures superlinear trends in our power model. The $\sqrt{y}$ and $log(y)$ transformations are standard from the statistics literature and were empirically shown effective for reducing error and bias in our analyses [Harrell 2001]. We fit a transformed response $f(y)$ but quantify accuracy for the original response y (Section 3.5).

3.4 Model Derivation

The statistically rigorous derivation of performance and power models emphasizes the role of domain knowledge in computer engineering when specifying the model's functional form. This approach leads to models consistent with prior intuition about the design space. Furthermore, association and correlation analyses before model specification prune unnecessary, ineffective predictors to improve model efficiency. Specifically, we consider the following design process for regression modeling:

(1) **Hierarchical Clustering:** Clustering examines correlations between potential predictors and enables elimination of redundant predictors. Predictor pruning controls model size, thereby reducing risk of over-fitting and improving model efficiency during formulation and prediction.

(2) **Association Analysis:** Scatterplots qualitatively capture approximate trends of predictor-response relationships, revealing the degree of non-monotonicity or non-linearity. Scatterplots with low response variation as predictor values change may suggest predictor insignificance, enabling further pruning.

(3) **Correlation Analysis:** Correlation coefficients quantify the relative strength of predictor-response relationships observed in the scatterplots of association analysis. These coefficients impact our choice in non-linear transformations for each predictor.

(4) **Model Specification:** Domain-specific knowledge is used to specify predictor interaction. The correlation analysis is used to specify the degree of flexibility in non-linear transformations. Predictors more highly correlated with the response will require more flexibility since any lack of fit for these predictors will impact overall model accuracy more. Given the model's functional form, least squares determines regression coefficients.

(5) **Assessing Fit:** The R^2 statistic quantifies the fraction of response variance captured by the model's predictors. Larger R^2 suggests a better fit to training data. Normality and randomness assumptions for model residuals are validated using quantile-quantile plots and scatterplots. Residual normality and

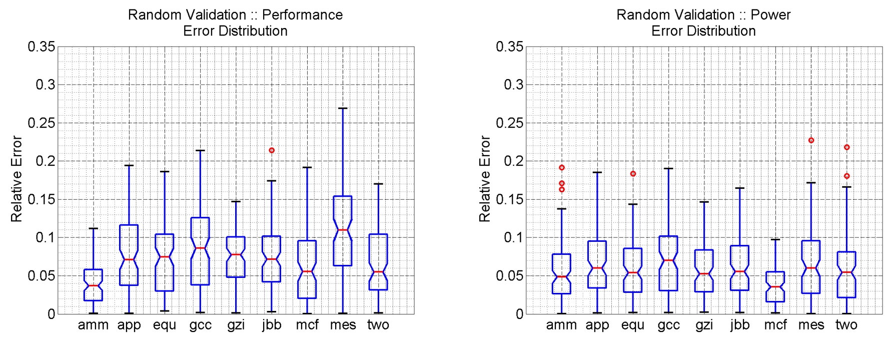


Fig. 3. Model Accuracy :: error distribution for 100 random validation designs

randomness are prerequisites to any further significance testing. Lastly, predictive ability is assessed by performance and power predictions on a set of randomly selected validation points.

This process leads to a model specification, illustrated by example in Appendix A.

3.5 Prediction

Once $\vec{\beta}$ is determined, evaluating Equation (1) for a given x_i will give the expectation of $\hat{y}_i = E[y_i]$ in Equation (4). This result follows from observing the additive property of expectations, the expectation of a constant is the constant, and the random errors are assumed to have zero mean.

$$f(\hat{y}_i) = E\big[f(y_i)\big] = E\left[\beta_0 + \sum_{j=1}^{p} \beta_j g_j(x_{ij})\right] + E\big[e_i\big] = \beta_0 + \sum_{j=1}^{p} \beta_j g_j(x_{ij}) \qquad (4)$$

Figure 3 presents boxplots of the error distributions from performance and power predictions of 100 validation points sampled UAR from the design space. Note that these 100 validation points are collected separately and independently from training points. The error is computed as $|obs - pred|/pred$. Boxplots are graphical displays of data that measure location (median) and dispersion (interquartile range), identify possible outliers, and indicate the symmetry or skewness of the distribution. Boxplots are constructed by

—horizontal lines at median and upper, lower quartiles

—vertical lines drawn up/down from upper/lower quartile to most extreme data point within a factor of 1.5 of the IQR (interquartile range - the difference between first and third quartile) of the upper/lower quartile with short horizontal lines to mark the end of the vertical lines

—circles to denote outliers

Boxplots highlight quartiles, which are more representative of accuracy than an average error; averages can be biased by outliers. Medians are less susceptible to bias and can provide a better picture of error distributions.

Figure 3 indicates the performance model achieves median errors ranging from 3.7 percent (*ammp*) to 11.0 percent (*mesa*) with an overall median error across all

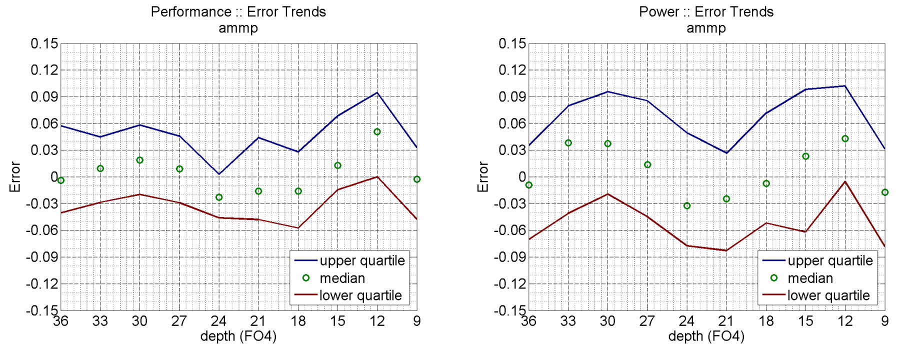


Fig. 4. Bias Analysis :: *ammp* model error for varying depths

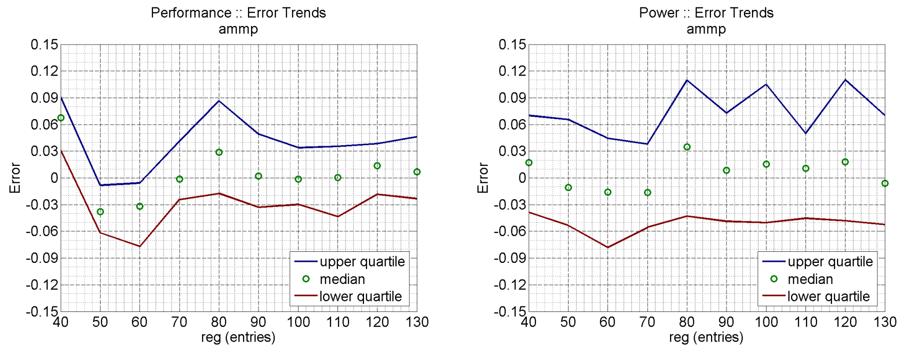


Fig. 5. Bias Analysis :: *ammp* model error for varying register counts

benchmarks of 7.2 percent. Power models are slightly more accurate with median errors ranging from 3.5 percent (*mcf*) to 7 percent (*gcc*) and an overall median of 5.4 percent.

3.6 Bias Analysis

The boxplots assess the high-level accuracy of the models across randomly chosen design points. However, we should also assess model bias for particular parameters or regions of the design space. We graphically check for biases by ensuring prediction error is random around zero for various parameter values. Figures 4–5 are representative of the trends across the benchmark suite and various design parameters. Each figure considers predicted validation points with various parameter values. For example, Figure 4 takes all validated points at a depth d and plots the error quartiles for each d from 9 to 36 FO4 delays per stage. These particular figures suggest the models are generally unbiased with median performance and power errors distributed between ± 6 percent for various pipeline depths and register file sizes. An indication of possible bias is the tail of positive performance errors for 40-entry register files. In general, however, there are no obvious deviations from randomness to suggest obvious biases and bias might be re-examined if the user observes suspicious trends when applying the model. Similar results are obtained for other benchmarks and parameters.

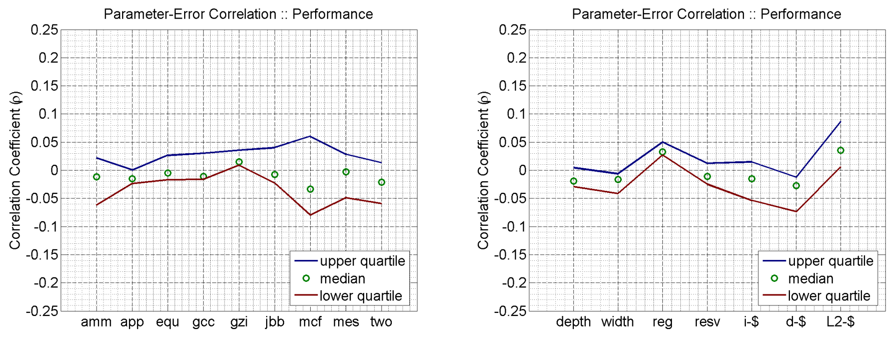


Fig. 6. Performance error correlation across benchmarks, parameters

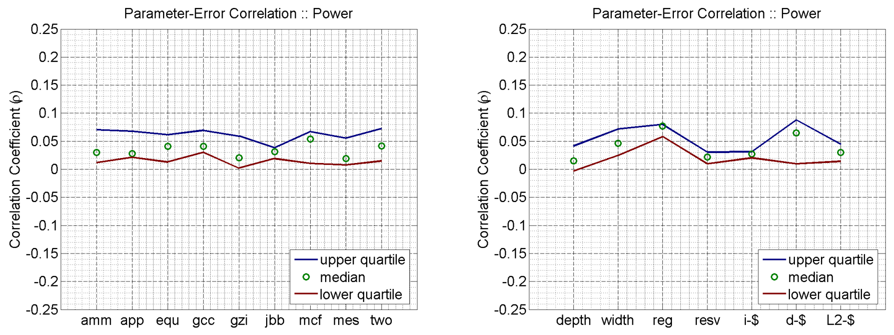


Fig. 7. Power error correlation across benchmarks, parameters

Figures 6–7 summarize the measured model bias by reporting correlations between model error, benchmarks and parameters. Given that correlation coefficients range from -1 to 1, errors from ideally unbiased models will have a correlation of zero. The figures on the left illustrate correlations between error and benchmarks summarized across all parameters. For example, Figure 6 illustrates a median error correlation of -0.011 for *ammp*. This value computes the correlation between *ammp* model error and parameter value for each of the seven parameters. The median of these seven correlation coefficients is reported as -0.011. Thus, the figures on the left summarize error correlations across the full range of parameters for each benchmark. Similarly, the figures on the right summarize error correlations across the full range of benchmarks for each parameter.

The performance correlations of Figure 6 are distributed around zero with very small correlations (less than 0.05), suggesting an unbiased performance model with errors correlated with neither benchmark nor parameter. The power analyses of Figure 7 indicate a small positive bias, suggesting errors tend to increase with larger parameter values. However, this correlation is less than 0.05 in most cases and is unlikely to cause any significant problems when applying the model.

The current bias study examines global biases at coarse granularity only. Such a study indicates the models are unbiased for predictions randomly chosen across the entire design space. However, we may observe non-trivial biases at fine granularity in which all predictions within a region of interest are biased either positive or negative. These regional biases arise from a mismatch between global samples used in model formulation and local model usage. Such biases may be mitigated by re-formulating models solely with samples from the region of interest. Since this article's studies evaluate models for points throughout the design space, a lack of global bias is sufficient.

3.7 Design Space Studies

Given the accuracy of regression models, we present applications of performance and power regression modeling to three representative design space studies:

—**Pareto Frontier Analysis:** Comprehensively characterize the design space, constructing a regression predicted Pareto frontier in the power-delay space.

—**Pipeline Depth Analysis:** Combine regression and the framework of prior pipeline depth studies to identify $bips^3/w$ maximizing depths. Enhance prior studies by varying all design parameters simultaneously instead of fixing most non-depth parameters.

—**Multiprocessor Heterogeneity Analysis:** Identify $bips^3/w$ maximizing architectures for each benchmark via regression. Cluster these architectures to identify compromise designs and power-performance benefits from varying degrees of core heterogeneity.

We formulate models using samples from the training space of 375,000 points (Table I). We explore a design space of 262,500 points ranging that includes depths from 12 to 30 FO4, which is smaller than the original sample space of 375,000 points that include 9, 33, and 36 FO4 depths. The sample space should be larger than the design space for exploration to mitigate errors from extrapolation. We exclude 9, 33, and 36 FO4 from exploration since performance and power trends do not change dramatically in these extreme design regions [Zyuban et al. 2004].

4. PARETO FRONTIER ANALYSIS

Pareto optimality is an economic concept with broad applications to engineering. Given a set of design parameters and a set of design metrics, a Pareto optimization changes the parameters to improve at least one metric without negatively impacting any other metric. A design is Pareto optimal when no further Pareto optimizations can be implemented. For the microarchitectural design space, Pareto optima are designs that minimize delay for a given power budget or minimize power for a given delay target. A Pareto frontier is defined by a set of Pareto optima.

Regression models enable a complete characterization of the microarchitectural design space. We leverage the computational efficiency of regression to perform an exhaustive evaluation of the design space containing more than 260,000 points. Such a characterization reveals all trade-offs between a large number of design parameters simultaneously compared to an approach that relies on per parameter sensitivity analyses. Given this characterization, we construct Pareto frontiers. While we cannot explicitly validate the regression identified Pareto frontier against a hypothetical frontier found by exhaustive simulation, the former is likely close to the latter given the accuracy observed in validation.

	Depth (FO4)	Width	Reg	Resv	I-$ (KB)	D-$ (KB)	L2-$ (MB)	Delay Model	Err (%)	Power Model	Err (%)
ammp	27	8	130	12	32	128	2	1.0	0.2	35.9	-3.9
applu	27	8	130	15	16	8	0.25	0.8	-0.8	39.6	0.1
equake	27	8	130	15	64	8	0.25	1.2	-0.8	41.5	-3.0
gcc	15	2	70	9	16	8	1	1.2	5.2	44.1	-6.0
gzip	15	2	70	6	16	8	0.25	0.8	8.8	24.2	0.0
jbb	15	8	80	12	16	128	1	0.6	-4.7	80.9	1.6
mcf	30	2	70	6	256	8	4	3.5	2.4	12.9	-3.0
mesa	15	8	80	13	256	32	0.25	0.4	5.2	86.9	-7.1
twolf	27	8	130	15	128	128	2	1.1	-1.2	34.5	-3.6

Table II. Efficient Designs :: $bips^3/w$ maximizing architectures per benchmark

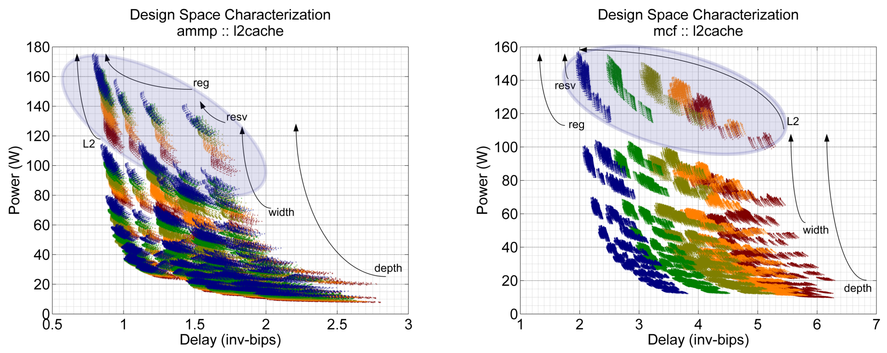


Fig. 8. Design Characterization :: predicted delay, power of all designs for representative benchmarks; arrows indicate trends as parameter values change; colors map to L2 cache sizes

4.1 Design Space Characterization

Figure 8 plots the predicted delay (inverse throughput) and power of the design space by exhaustively evaluating the regression models for representative benchmarks. The design space is characterized by several overlapping clusters of similar designs. Each cluster contains designs with a particular pipeline depth-width combination. For example, the shaded *mcf* cluster with delay ranging from 1.9 to 5.3 seconds and power ranging from 100 to 160 watts minimizes delay at the greatest power cost with depth of 12FO4 and decode bandwidth of 8 instructions per cycle.

The arrows of Figure 8 identify power-delay trends as a particular resource size increases. Consider the shaded 12FO4, 8-wide design clusters for *ammp* and *mcf*. *Mcf* experiences substantial performance benefits from larger caches with delay shifting from 5.3 to 1.9 seconds as L2 cache size shifts from 0.25 to 4MB. In contrast, *ammp* sees increasing power costs with limited performance benefits of 1.0 to 0.8 seconds as L2 cache size increases by the same amount. *Ammp* also appears to exhibit greater instruction level parallelism, effectively utilizing additional physical registers and reservation stations to reduce delay from approximately 1.8 to 0.8 seconds compared to *mcf*'s reduction of 2.5 to 2.0 seconds.

4.2 Pareto Frontier Identification

Given a design space characterization, Figure 9 plots regression predicted Pareto optima. These optima minimize delay for a given power budget. Given regression

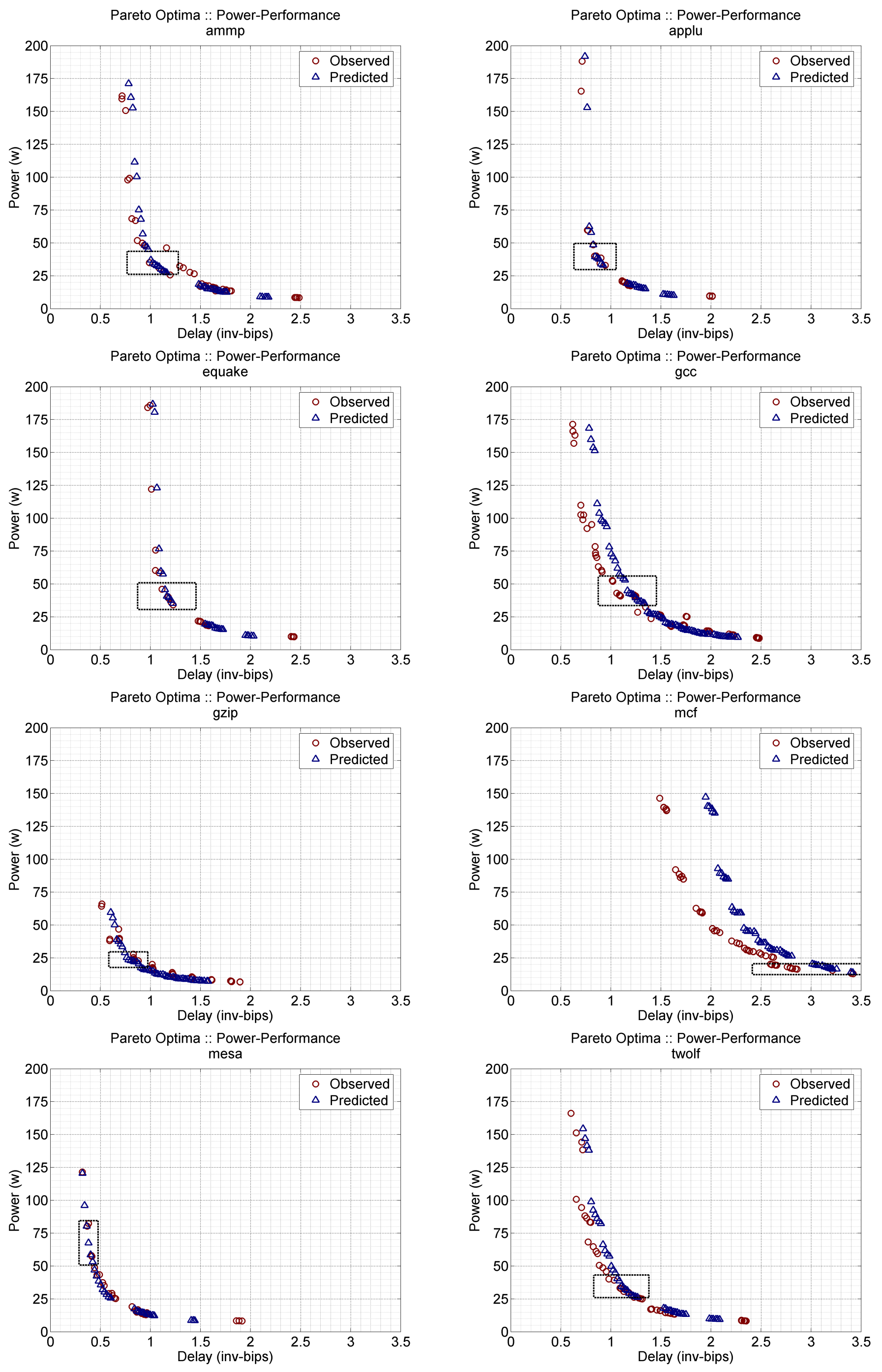


Fig. 9. Pareto Frontier :: Pareto optima for representative SPEC CPU benchmarks

models and exhaustively predicted power and delay characteristics, the frontier is constructed by discretizing the range of delays and identifying the design that minimizes power for each delay in a number of delay targets. These designs are Pareto optimal with respect to the regression models, but may not be the same optima obtained via a hypothetical exhaustive simulation of the space.

Although Pareto optima are useful for particular delay or power targets, not all Pareto optima are power-performance efficient with respect to $bips^3/w$, the inverse energy delay-squared product.[2] We compute the efficiency metric for each design on the Pareto frontier and identify the most efficient designs in Table II. The $bips^3/w$ optimal design for *ammp* is located at 1.0 seconds and 35.9 watts in the delay-power space, the knee of the Pareto optimal curve. Similarly, the *mcf* $bips^3/w$ optimal design is located at 3.5 seconds and 12.9 watts. Overall, these optima are drawn from diverse design regions motivating comprehensive space exploration.

The boxes of Figure 9 identifies a region around the $bips^3/w$ optima for each benchmark. Although Table II indicates these optima occupy very different parts of the design space, they reside in very similar regions of the power-delay space. Most of the optima are located between 0.5 and 1.5 seconds, 25 and 50 watts with obvious exceptions in *mcf* and *mesa*.

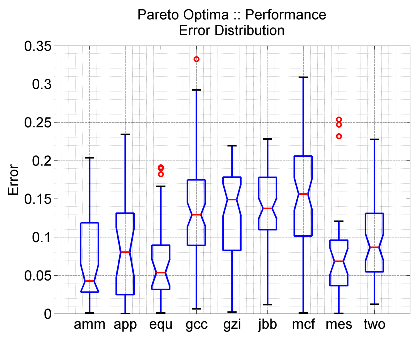
4.3 Pareto Frontier Validation

Figure 9 superimposes simulated and predicted Pareto frontiers, suggesting good relative accuracy. Regression effectively captures the delay-power trends of the Pareto frontier. As performance prediction is less accurate than power prediction, however, differences are often characterized by horizontal shifts in delay. Performance model accuracy is the limiting factor for more accurate Pareto frontier prediction across all benchmarks in our suite.

Performance errors are particularly evident for benchmark *mcf*. This application is relatively memory bound and many designs occupy the high-delay region of the space. Thus, low-delay points are rare and tend to be over-estimated, as high-delay points exert greater influence during model fitting. This bias might be addressed by customizing a sampling strategy for *mcf*, which might assign greater weight to low-delay training samples. Benchmark *mcf* performance errors are more an exception than a common case and *ammp* is more representative of Pareto frontier accuracy.

Figure 10 presents the error distributions for the performance and power prediction of points on the Pareto frontier. The median performance error ranges from 4.3 percent (*ammp*) to 15.6 percent (*mcf*) with an overall median of 8.7 percent. Similarly, the median power error ranges from 1.4 percent (*mcf*) to 9.5 percent (*applu*) with an overall median of 5.5 percent. These error rates are consistent with the performance and power median error rates of 7.2 and 5.4 percent observed in the validation of random designs (Figure 3), suggesting predictions for Pareto optima are generally as accurate as those for the overall design space. As shown in Table II, errors associated with $bips^3/w$ optimal predictions are also consistent with those for the broader space. Delay errors range from 0.2 to 8.8 percent while power errors range from 0.1 to 7.1 percent.

[2] $bips^3/w$ is a voltage invariant power-performance metric derived from the cubic relationship between power and voltage [Brooks et al. 2000].



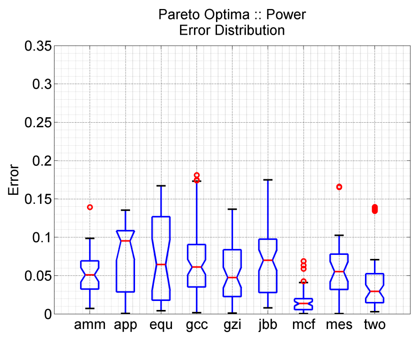


Fig. 10. Pareto Frontier Accuracy :: complete error distribution for Pareto optima

Note regression models are evaluated exhaustively for the design space to perform Pareto frontier validation; performance and power is predicted for every point in the space; the frontier is read off from these predictions. By comparing simulated and predicted metrics for designs estimated to be Pareto optimal, we find regression is accurate for designs in efficient regions of the space. However, this validation does not indicate whether regression models identify the same Pareto frontier that would have been identified by simulation alone. Identifying a frontier through exhaustive simulation to perform this comparison is prohibitively expensive.

In practice, not all Pareto optima are interesting and viable designs. The high power or high delay designs located at the frontier extrema are not particularly interesting due to unfavorable power and delay trade-offs. For the majority of benchmarks, we find our models may be more accurate for the more interesting points near the $bips^3/w$ optimum of Table II. Figure 11 presents restricted error distributions from considering only Pareto optima with delay and power within 25 percent of the $bips^3/w$ optimal delay and power (boxes of Figure 9).

Comparing complete and restricted error distributions, we find the median and interquartile range decrease for a majority of benchmarks as we examine only the region around the $bips^3/w$ optimum. The restricted performance and power error distributions are more favorable for five and six benchmarks, respectively. Models are more effective for the interior of the design space as interpolation is often more accurate than extrapolation. Since $bips^3/w$ optimal designs often reside within the interior of the design space, moderating resource allocations to balance performance and power, models are likely more accurate for $bips^3/w$ optimal designs.

The differing error distributions between Figures 10–11 motivate future work on hierarchical modeling schemes in which high-level models are constructed for a comprehensive design space to identify regions of interest around particular optima or $bips^3/w$ maximizing designs. Further detail and accuracy may be achieved by performing constrained spatial sampling and constructing localized regression models for this region of interest. Such a scheme overcomes the models' potential regional biases and may further reduce model error as we shift emphases from the complete design space to particular subspaces.

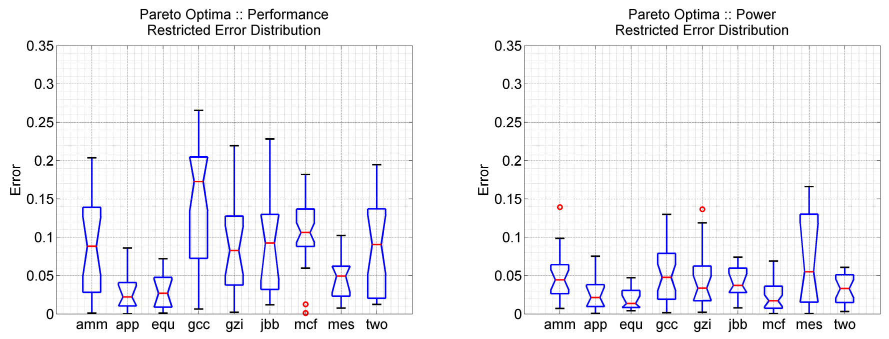


Fig. 11. Pareto Frontier Accuracy :: restricted error distribution for Pareto optima

Processor Core	
Decode Rate	4 non-branch insns/cy
Dispatch Rate	9 insns/cy
Reservation Stations	FXU(40),FPU(10),LSU(36),BR(12)
Functional Units	2 FXU, 2 FPU, 2 LSU, 2 BR
Physical Registers	80 GPR, 72 FPR
Branch Predictor	16k 1-bit entry BHT
Memory Hierarchy	
L1 DCache Size	32KB, 2-way, 128B blocks, 1-cy lat
L1 ICache Size	64KB, 1-way, 128B blocks, 1-cy lat
L2 Cache Size	2MB, 4-way, 128B blocks, 9-cy lat
Memory	77-cy lat
Pipeline Dimensions	
Pipeline Depth	19 FO4 delays per stage
Pipeline Width	4-decode

Table III. Baseline Architecture

5. PIPELINE DEPTH ANALYSIS

Prior pipeline studies considered various depths while holding most other design parameters at constant values, in part, to control the simulation costs of varying multiple parameters simultaneously [Hartstein and Puzak 2002; Hrishikesh et al. 2002; Zyuban and Strenski 2003]. Thus constraining the space may lead to narrowly defined studies with conclusions that may not generalize. Regression models enable a more complete characterization of pipeline depth trends by allowing other design parameters to vary simultaneously. A more comprehensive depth analysis ensures observed trends are not an artifact of the constant baseline values to which other parameters are held.

Pipeline depth is specified by the number of fan-out-of-four (FO4) inverter delays per pipeline stage. When logic and latch overhead per pipeline stage is measured in terms of FO4 delay, deeper pipelines have smaller FO4 delays. We consider pipeline ranging from 12 to 30 FO4 to compare and contrast the following:

—**Original Analysis:** Consider the POWER4-like baseline architecture of Table III, predicting power-performance efficiency as depth varies and all other design parameters are held constant at baseline values.

—**Enhanced Analysis:** Consider the design space of Table I, predicting efficiency as parameters vary simultaneously.

5.1 Pipeline Depth Trends

The line plot of Figure 12 presents predicted efficiency relative to the $bips^3/w$ maximizing baseline design in the constrained original analysis. 18 FO4 delays per stage is optimal for an average of the benchmark suite. Although choosing the deepest or shallowest pipeline will achieve only 85.9 or 87.6 percent of the optimal efficiency, respectively, the models suggest a plateau around the optimum and not a sharp peak. The superimposed boxplots of Figure 12 show the efficiency distribution of the 37,500 designs for each pipeline depth in the enhanced analysis.[3] By graphically presenting efficiency quartiles, the boxplot for 18 FO4 designs indicate 75, 50, and 25 percent of these designs achieve efficiency of at least 79, 102, and 131 percent of the original $bips^3/w$ optimum.

The maxima of these boxplots constitute a potential bound on $bips^3/w$ efficiency achievable in this design space with up to 2.1x improvements at the optimal 18 FO4 pipeline depth.[4] These bounding architectures are characterized by wide pipelines as well as larger queue and register file sizes. The efficiency of wide pipelines are likely a result of the energy-efficient functional unit clustering modeled by the simulator, which enables near linear power increases as width increases [Zyuban and Kogge 2001]. However, our power models also account for superlinear width power scaling for structures such as the multi-ported register file, memory units, rename table, and forwarding logic [Zyuban and Kogge 2001]. Larger queue and reservation resources result from deeper pipelines and more instructions in flight.

The points at which the line plot intersect the boxplots indicate unexploited efficiency. Intersection at a lower point in the boxplot indicates a larger number of configurations are predicted more efficient than baseline at a particular depth. More than 58 percent of 12 FO4 and 39 percent of 30 FO4 designs are predicted more efficient than baseline, corresponding to more than 21,000 and 14,000 designs, respectively. Such a large number of more efficient designs is not surprising, however, since the baseline resembles designs for server workloads with less emphasis on energy efficiency. Less efficient designs may be pruned from further study enabling more judicious use of detailed simulators should additional simulation be necessary.

Predicted efficiency penalties for sub-optimal depths are also more significant for the bound architectures. The $bips^3/w$ maximizing depth is 15-18 FO4 and the sub-optimal 30 FO4 design achieves 88 percent of the optimal efficiency, incurring a 12 percent efficiency penalty. The numbers above each boxplot in Figure 12 quantify each bound architecture's efficiency relative to that of the $bips^3/w$ maximizing bounding architecture. While the bounding architectures are also most efficient at 15 to 18 FO4, the sub-optimal 30 FO4 design achieves only 81 percent of the optimal efficiency and incurs a 19 percent penalty. This trend is observed for all depths shallower than the optimal 18 FO4. Since bound architectures are characterized by wider pipelines, choice of depth becomes more significant. For the average across

[3] Given $|S| = 272,500$ points in the design space and 7 possible depths (12-30FO4 in steps of 3FO4), there are 37,500 designs for each depth.

[4] The 2.1x improvement over the IBM Power4 18 FO4 baseline likely arises from a difference in target workloads. Customized architectures for nine specific workloads from Section 2.1 will be more efficient than the baseline IBM Power4 18 FO4 pipeline, which likely targeted a broader range of applications.

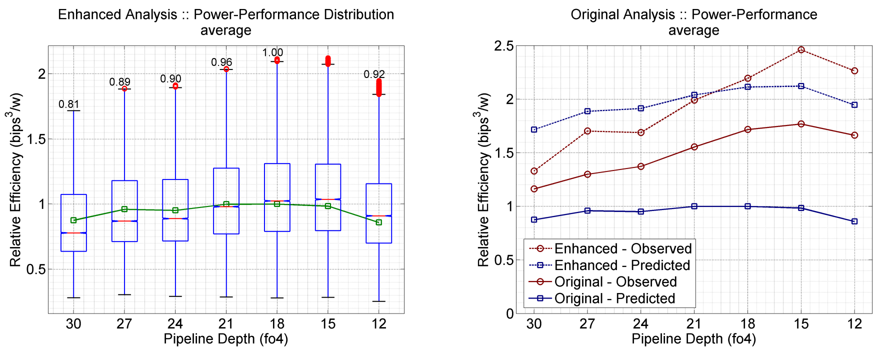


Fig. 12. Comparative Efficiency :: original [line plot] and enhanced [boxplots] analyses relative to original $bips^3/w$ optimum; $bips^3/w$ efficiency validation

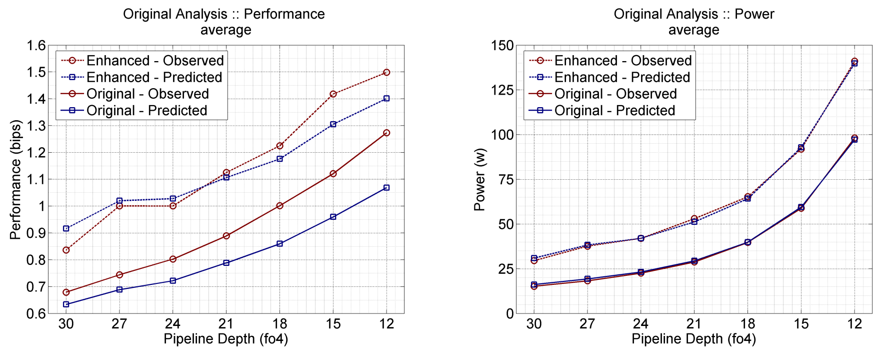


Fig. 13. Metric Validation :: performance, power validation for varying depths

our benchmark suite, wide pipelines with shallow depths will result in greater design imbalances and power-performance inefficiencies.

Figure 14 presents the distribution of data cache sizes in the most efficient designs at each depth. In particular, we take the 37,500 designs at each depth and consider designs in the 95-th percentile (*i.e.*, 1,875 designs in the top 5 percent of each depth's boxplot). Small 8KB data caches are observed for 20.3 percent of top designs at 30FO4 while such caches are optimal for only 1.4 percent of top designs at 12FO4. The percentage of top designs with larger 64KB caches increases from 22.8 to 34.4 percent with deeper pipelines. Thus, smaller caches are increasingly viable at shallow pipelines while top designs often have larger caches at deep pipelines. This frequency analysis confirms our intuition that deeper pipelines favor larger caches to mitigate the increased costs of cache misses. This analysis also illustrates variability in the most efficient designs and the effect of parameter interactions.

5.2 Pipeline Depth Validation

Figure 12 validates the $bips^3/w$ predictions, suggesting regression captures high-level trends in both analyses. The models correctly identify the most efficient depths to within 3 FO4 and capture the difference in efficiency penalties from sub-optimal depths between the two analyses. Whereas models predict 12 and 19

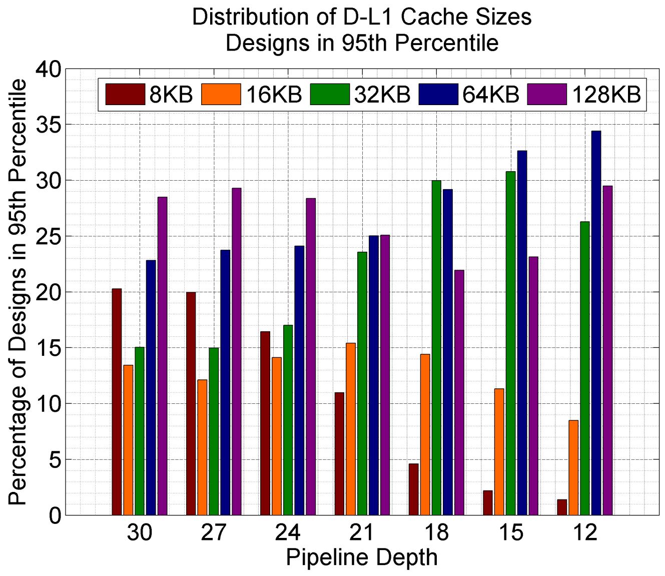


Fig. 14. Data Caches and Depth :: distribution of d-L1 cache sizes for designs in 95th percentile

percent penalties, simulation identifies 52 and 67 percent penalties relative to 15 FO4 for the original and enhanced analyses, respectively. Thus, the significance of the optimum and penalties for sub-optima are more pronounced in simulation. Sub-optima are more likely located at the extreme regions of the design space, resulting in greater extrapolation error.

Although the models are accurate for capturing high-level trends, $bips^3/w$ error rates are larger than those for performance and power. However, the $bips^3/w$ validation obscures underlying performance and power accuracy. By decomposing the validation of $bips^3/w$ in Figure 13, we find the underlying models exhibit good relative accuracy, effectively capturing performance and power trends. Since predictions from less accurate performance models must be cubed to compute $bips^3/w$, performance model errors are also cubed and negatively impact $bips^3/w$ accuracy. Countering these effects is continuing work.

6. MULTIPROCESSOR HETEROGENEITY ANALYSIS

As shown in Table II, regression models may be used to identify the $bips^3/w$ optimal architectures for each benchmark. In a uniprocessor or homogeneous multiprocessor design, the core is designed as an approximate compromise between these per benchmark optima to accommodate a range of workloads. Heterogeneous multiprocessor core design mitigates the efficiency penalties of this compromise [Kumar et al. 2004]. However, prior work considered limited design spaces due to simulation costs. We combine regression modeling and clustering analyses to enable a more general exploration of core designs in heterogeneous architectures. This study identifies design compromises for the $bips^3/w$ design metric and quantifies a theoretical upper bound on the potential efficiency gains from high-performance heterogeneity, neglecting any associated multiprocessor overhead.

In particular, we combine our regression models with *K-means clustering*. A K-clustering of a set S is a partition of the set into K subsets which optimizes some clustering criterion, usually a similarity metric. Well defined clusters are such that all objects in a cluster are very similar and any two objects from distinct clusters are very dissimilar. General K-clustering is NP-hard and K-means clustering is a heuristic approximation.

6.1 Clustering Methodology

We first completely characterize the design space via regression to identify the $bips^3/w$ maximizing architectures for each benchmark in our suite (Table II). These designs constitute the set to be partitioned into K subsets when clustering. The optimal design parameters exhibit significant diversity across benchmarks with depth ranging from 15 to 30 FO4, width ranging from 2 to 8 instructions decoded per cycle, and L2 caches ranging from 0.25 to 4 MB. Each benchmark's execution characteristics are reflected in its optimal architecture. For example, compute-intensive *gzip* has the smallest L2 cache while memory-intensive *mcf* has the largest.

We perform K-means clustering for these nine benchmark architectures to identify *compromise architectures*. The heuristic for K clusters consists of the following:

(1) Define K centroids, one for each cluster, and place randomly at initial locations in space containing objects to be clustered.
(2) Assign each object to cluster with closest centroid.
(3) When all objects have been assigned, re-compute placement of K centroids such that its distance to objects in its cluster is minimized.
(4) Since centroids may have moved in step 3, object assignment to clusters may change. Thus, steps 2 and 3 are repeated until centroid placement is stable.

We use a normalized and weighted Euclidean distance as our measure of similarity in steps 2 and 3. For a particular design parameter, we normalize its values by subtracting its mean and dividing by its standard deviation. Furthermore, we weight these normalized values by the parameter's correlation coefficient with $bips^3/w$, effectively giving greater emphasis to parameters with a greater impact on $bips^3/w$ in the distance calculation. Thus, if correlation coefficients ${\rho^2}_i > \rho^2_j$, an increase in parameter p_i will change the distance more than the same increase in parameter p_j. The distance between two architectures represented by vectors $\vec{a}$,$\vec{b}$ of p parameter values is determined by normalizing and weighting the values in $\vec{a}$,$\vec{b}$ and computing the Euclidean distance.

For example, pipeline depth values range from 12 to 30 FO4 in increments of 3 with a mean of 21 and standard deviation of 6.48. The normalized depth values range from -1.39 to 1.39 with mean 0 and standard deviation of 1.0. We then utilize the 1,000 samples used in regression model formulation to compute the correlation between depth and $bips^3/w$ and obtain a weighting factor.

6.2 Heterogeneity Efficiency

Each cluster from K-means corresponds to a grouping of similar architectures and each centroid represents its cluster's compromise architecture. We take the number

Cluster	Depth	Width	Reg	Resv	I-$ (KB)	D-$ (KB)	L2-$ (MB)	Avg Delay Model	Avg Power Model
1	15	8	80	12	64	64	0.5	2.26	82.17
2	27	8	130	14	32	32	0.5	1.05	32.53
3	15	2	70	8	16	8	0.5	0.93	37.55
4	30	2	70	6	256	8	4	0.29	12.91

Table IV. K=4 Compromise Architectures :: microarchitectural designs

Cluster	Benchmarks
1	jbb, mesa
2	ammp, applu, equake, twolf
3	gcc, gzip
4	mcf

Table V. K=4 Compromise Architectures :: benchmark mapping

of clusters as the number of distinct compromise designs and, thus, a measure of heterogeneity. Table IV uses a $K = 4$ clustering to identify compromise architectures and their average power-delay characteristics when executing their associated benchmarks. This analysis illustrates our models' ability to identify optima and compromises occupying diverse parts of the design space. For example, the four compromise architectures capture all combinations of pipeline depths and widths. Cluster 1 contains the aggressive deep, wide pipeline for *jbb* and *mesa*. Cluster 4, containing the memory-intensive *mcf*, is characterized by a large L2 cache and shallow, narrow pipeline. Clusters 2 and 3 trade-off pipeline depth and width depending on application-specific opportunities for instruction level parallelism. The ability to identify diverse optima is increasingly important as we observe microarchitectural differentiation for various market segments and applications.

Figure 15 plots the delay and power characteristics of the nine benchmark architectures executing their corresponding benchmarks (radial points). Aggressive architectures with deep, wide pipelines are located in the upper left quadrant and the less aggressive cores with shallow, narrow pipelines are located in the lower right quadrant. Deep, narrow and shallow, wide architectures both occupy the moderate center. The four compromise architectures executing their benchmark clusters are also plotted (circles) to demonstrate the delay and power compromises with associated per benchmark optima. Although we cluster in a p-dimensional microarchitectural space, the strong relationship between an architecture and its delay and power characteristics means we also observe clustering in the 2-dimensional delay-power space. Spatial locality between a centroid and its cluster's objects suggest modest delay and power penalties from architectural compromises. Thus, the delay and power characteristics of the benchmark suite executing on a heterogeneous multiprocessor with these four cores are similar to those when executing on the nine benchmark architectures. As a corollary, the benchmarks could achieve close to ideal $bips^3/w$ efficiency on this heterogeneous design.

Figure 15 also reveals new opportunities for workload similarity analysis based on resource requirements at the microarchitectural level. For example, *ammp*, *applu*, *equake*, and *twolf* may be similar workloads since they are most efficient at similar pipeline dimensions and cache sizes. Prior work in similarity analysis has been used to reduce the fraction of benchmark suites for microarchitectural simulation [Eeckhout and H. Vandierendonck 2003; Phansalkar et al. 2005; Yi et al. 2005].

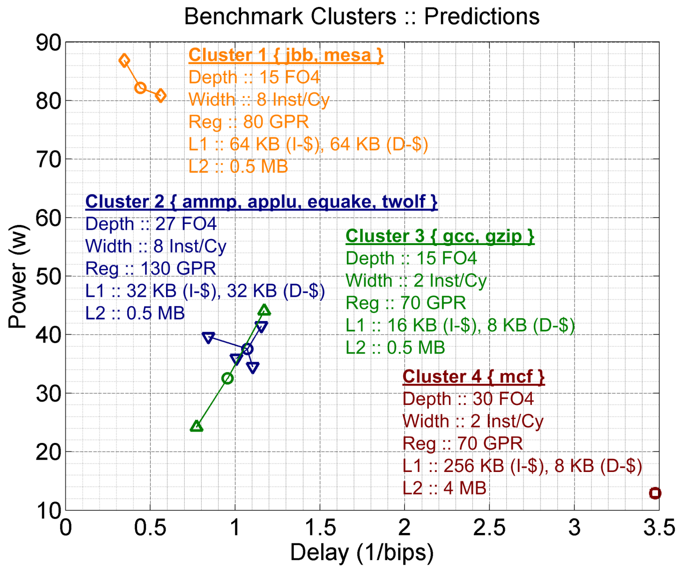


Fig. 15. Optimization and Clustering :: delay, power for per benchmark optima of Table II (radial points) and resulting compromises of Table IV (circles)

However, similarity exposed by microarchitectural clustering may be most useful for hardware accelerator design. In the ideal case, accelerators would be designed for every kernel of interest. However, resource constraints necessitate compromises and the penalties from such compromises may be minimized by designing an accelerator to meet the needs of multiple similar kernels.

Figure 16 plots predicted $bips^3/w$ efficiency gains for the nine benchmarks and the benchmark average as the number of clusters increases in the K-means algorithm. Recall cluster count quantifies the degree of heterogeneity. Efficiency is presented relative to the POWER4-like baseline (cluster count 0). The homogeneous architecture identified by K-means clustering (cluster count 1) is predicted to improve average efficiency by 1.46x with the largest gains for *mesa* (4.6x) at the expense of *mcf* (0.46x). For three cores, all benchmarks see benefits from heterogeneity resulting in an average gain of 1.9x. We observe diminishing marginal returns in heterogeneity beyond 4 cores. The four cores in Table IV are predicted to benefit efficiency by 2.2x, 8 percent less than the theoretical upper bound of 2.4x that is achievable only from the much greater heterogeneity of 7 to 9 cores. The benefits for nine different cores is the theoretical upper bound on heterogeneity benefits as each benchmark executes on its $bips^3/w$ maximizing core.

6.3 Heterogeneity Validation

Figure 17 compares the simulator reported heterogeneity gains against those of our regression models. The models are pessimistic for lower degrees of heterogeneity (*i.e.* cluster counts less than four). The gap between predicted and simulated efficiency narrows from 37.9 percent at cluster count zero to 14.4 percent at cluster count three. The simulated four core average benefit is 2.0x compared to the mod-

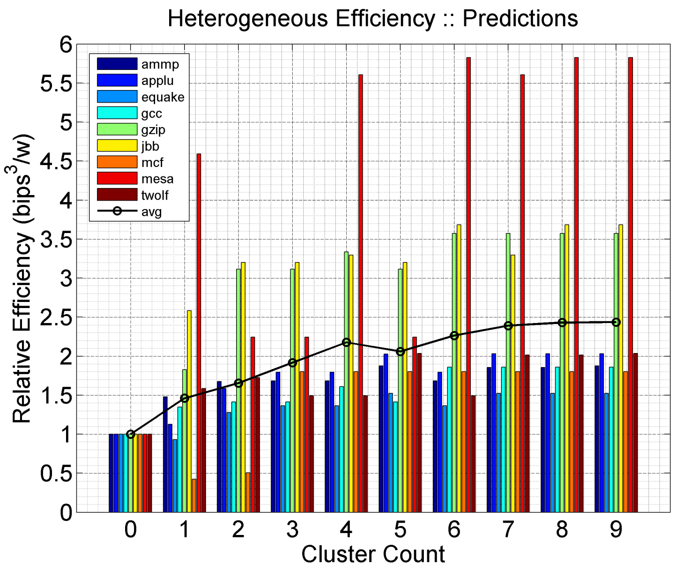


Fig. 16. Heterogeneity Trends :: predicted efficiency gains; cluster 0 is baseline of Table III, cluster 1 is homogeneous multicore from K-means, cluster 9 is heterogeneous multicore of Table II.

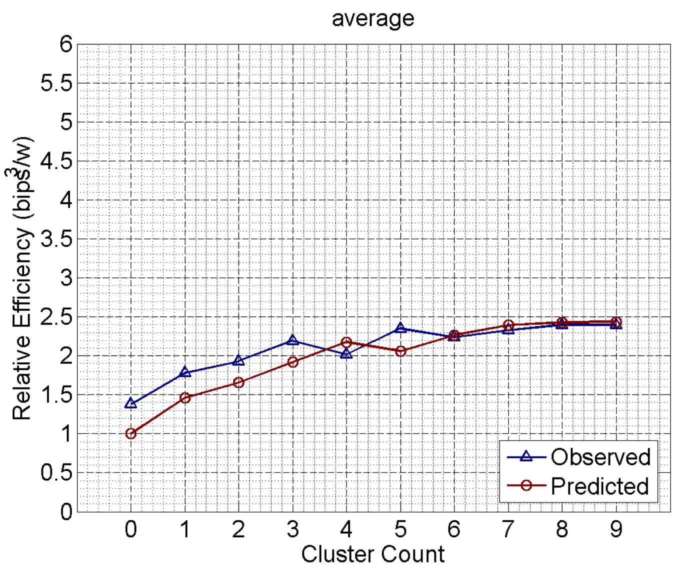


Fig. 17. Heterogeneity Validation :: average $bips^3/w$ average efficiency validation, x-axis interpreted as in Figure 16

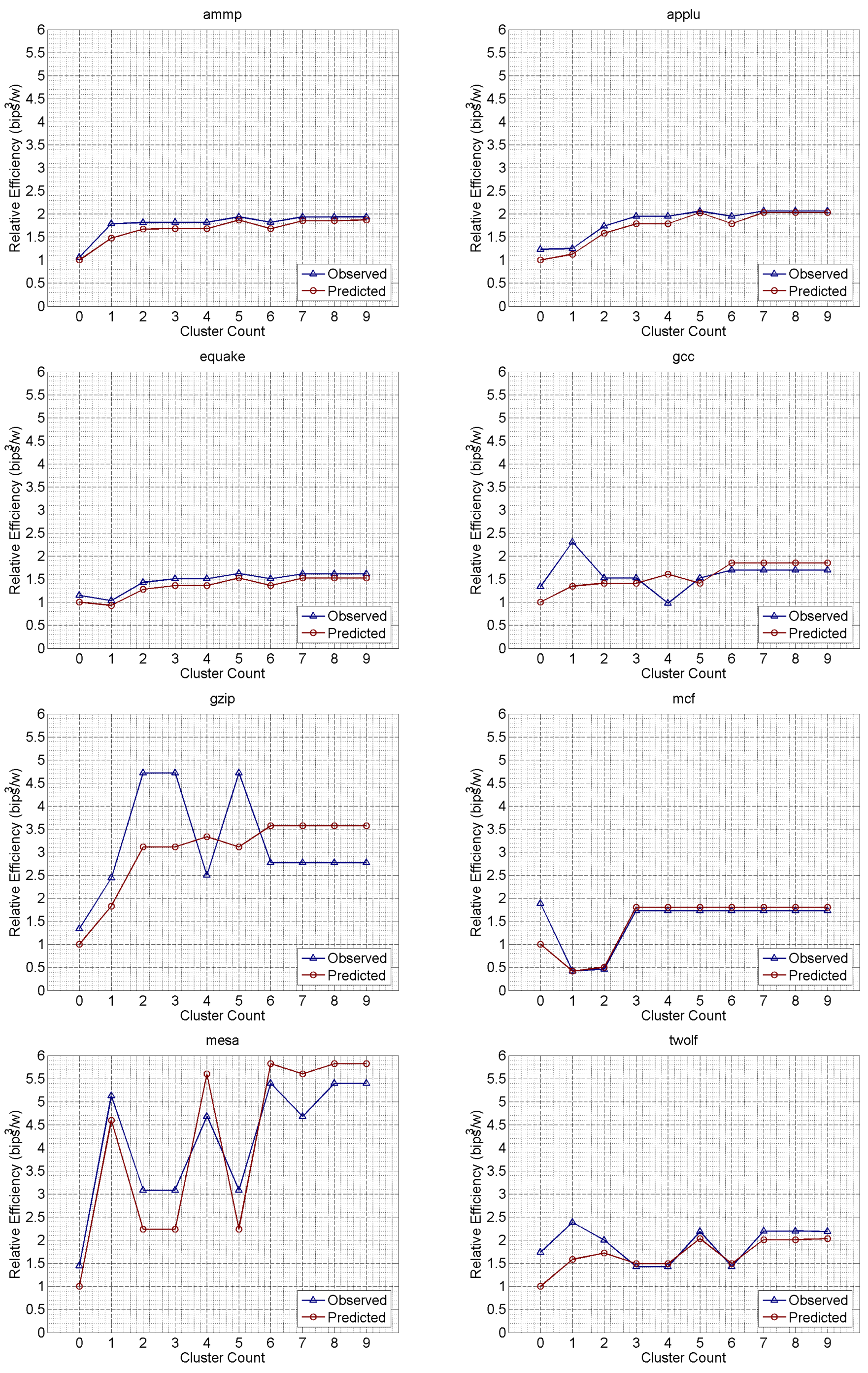


Fig. 18. Heterogeneity Validation :: $bips^3/w$ efficiency validation for representative SPEC CPU benchmarks, x-axis interpreted as in Figure 16

eled benefit of 2.2x. This point of diminishing marginal returns from additional heterogeneity is predicted with a 7.8 percent error; the regression models are relatively optimistic. At higher degrees of heterogeneity (*i.e.* cluster counts greater than 6), we observe much greater accuracy with error rates of less than three percent. The predicted upper bound on heterogeneity benefits of 2.4x is accurate with only 1.7 percent difference in simulation.

Figure 18 assesses benchmark level effects, illustrating efficiency trends at varying degrees of heterogeneity. The regression models effectively capture application-specific effects. For example, in both simulation and model, we observe significant efficiency benefits for *gzip, mesa* at the expense of *mcf* when heterogeneity is limited (*i.e.*, low cluster counts). In effect, fewer clusters lead to design compromises that favor the majority (*gzip,mesa*) over the minority (*mcf*).

Figure 18 illustrates particularly poor relative accuracy for *gzip*, which arises from a combination of model errors and K-means clustering artifacts. With the exception of cluster count 4, benchmark *gzip* is assigned to clusters with 8-way superscalar designs for cluster counts 0 to 5. At 4 clusters, however, K-means misclassifies *gzip* into a 2-way superscalar design. Refined clustering or post-processed K-means might identify and eliminate the discontinuity at K=4.

Clustering artifacts aside, fewer clusters lead *gzip* to 8-way superscalar designs for which performance tends to be under-estimated, and more clusters lead *gzip* to 4-way superscalar designs for which performance tends to be over-estimated. Given that we observe good relative accuracy within a particular superscalar width, these effects might be mitigated by a *gzip*-specific derivation that builds separate regression models for each superscalar width.

We observe similar heterogeneity trends for benchmarks within the same cluster. For example, Table V identified a cluster with *ammp*, *applu*, *equake* and *twolf*. Since these benchmarks have similar resource requirements at the microarchitectural level, their achieved efficiency gains in the range of 1.5x to 2.0x are also similar. Collectively, these figures illustrate our models' abilities to capture the relative benefits of heterogeneity across benchmarks.

7. RELATED WORK

Fast simulation and improved design space exploration have been targets of many prior efforts. Sampling and modeling reduce costs of performance and power estimation for a variety of microarchitectural optimization studies.

7.1 Sampling and Modeling

Sampling. In contrast to this work, which focuses on spatial sampling for designs, much prior work reduces simulation costs through temporal sampling for representative instructions. SimPoint identifies phases from a workload, clusters these phases, and takes phases in cluster centroids as representative of the original workload during microarchitectural simulation [Sherwood et al. 2002]. By reducing sizes of instruction traces, SimPoint reduces costs per simulation. SMARTS identifies the number of instructions needed for a representative subset of the original workload [Wunderlich et al. 2003]. The number of samples is chosen to achieve user-specified confidence intervals when estimating design metrics, such as performance. Both SimPoint and SMARTS extract instruction segments from the original trace to capture broader application behavior.

Similarly, statistical profiling reduces the fraction of a workload that must be simulated [Eeckhout et al. 2003; Nussbaum and Smith 2001; Oskin et al. 2000] Such efforts recognize detailed simulations for specific benchmarks are not feasible early in the design process. Instead, profiling produces relevant program characteristics, such as instruction mix and data dependencies between instructions. A smaller synthetic benchmark then replicates these characteristics.

Introducing sampling and statistics into simulation reduces accuracy in return for gains in speed and tractability. While researchers in instruction sampling and synthetic benchmarks suggest this trade-off for simulator inputs (*i.e.*, workloads), we propose this trade-off for simulator outputs (*i.e.*, performance and power results). Temporal and spatial sampling should be applied jointly to reduce costs per simulation and number of simulations, respectively.

Significance Testing. Plackett-Burman matrices identify critical, statistically significant microarchitectural design parameters to design optimal multi-factorial experiments [Yi et al. 2005]. This method fixes all non-critical parameters to reasonable constants and performing extensive simulations that sweep a range of values for the critical parameters. By designing experiments more intelligently, designers use simulations more effectively and reveal more about the design space.

Stepwise regression provides an automatic and iterative approach to adding and dropping terms from a model depending on measures of significance [Joseph et al. 2006a]. However, prior applications of stepwise regression use these models for significance testing only and do not actually predict performance. Although commonly used, stepwise regression has several problems cited by Harrell [Harrell 2001]: (1) R^2 values are biased high, (2) standard errors of regression coefficients are biased low leading to falsely narrow confidence intervals, (3) p-values are too small, and (4) regression coefficients are biased high.

Empirical Modeling. Like regression, artificial neural networks can predict microarchitectural [Ipek et al. 2006; Joseph et al. 2006b]. ANN training costs for new, untrained applications can be reduced by expressing their performance as a linear combination of performance predictions for previously modeled applications [Dubach et al. 2008]. Training weights in this linear model is less expensive than training completely new application-specific models.

Comparing neural networks and spline-based regression models, we find similar accuracy but also find trade-offs in efficiency and automation [Lee et al. 2007]. Regression requires more rigorous statistical analysis while neural network construction is automated; the network is often treated as a black box. Regression models are likely more computationally efficient than neural networks. Regression models are constructed by solving linear systems and evaluated by multiplying matrices and vectors. In contrast, neural networks are constructed with gradient ascent and evaluated with nested weighted sums in multi-layer networks.

Analytical Modeling. In contrast to empirical models, analytical models capture first-order design trends by encapsulating designers' prior intuition and understanding of the design space. A first-order model for analyzing pipeline depth illustrates opposing design trends: greater instruction-level parallelism decreases the optimal depth while fewer pipeline stalls increases the optimal depth [Hartstein and Puzak 2002]. While trace-driven simulation can provide measures of application parallelism that combine with analytical expressions of microarchitectural capabilities to estimate performance [Noonburg and Shen 1994]. Similarly, analytical models can estimate performance by penalizing idealized steady-state performance with miss events from the branch predictor or cache hierarchy measured with fast, functional simulation [Karkhanis and Smith 2007].

7.2 Design Space Exploration

We compare our approach to related work in characterizing the sensitivity of design parameters, such as pipeline depth. We also draw on related work in statistics to characterize the roughness of microarchitectural performance and power topologies.

Sensitivity. Metrics for hardware and voltage intensity quantify compromises between energy and delay from circuit-level tuning and voltage scaling, respectively [Zyuban and Strenski 2003]. Intensity is computed as $\frac{D}{\delta D}\frac{\delta E}{E}$ where D is delay and E is energy. These intensity metrics produce conditions for optimal microarchitectural power-performance from mathematical relations, but do not compute the needed gradients. Our proposed regression models provide a mechanism for computing these gradients. Instead of implementing symbolically derived optimality conditions, we would optimize with heuristics using empirically derived regression models as objective functions.

Given sensitivity $\frac{\delta E/\delta X}{\delta D/\delta X}$ for tunable circuit parameters X such as gate sizing, supply voltage, and threshold voltage, optimal values for circuit parameters are those that equalize sensitivity [Markovic et al. 2004]. Sensitivity is equalized by jointly optimizing registers and logic within microarchitectural blocks (*e.g.*, arithmetic-logic units). In contrast to this circuit-level emphasis, we consider high-level interactions across a wide range of microarchitectural blocks and cache structures. Furthermore, prior works calculate the needed gradients from analytical circuit equations and simulations while we illustrate the feasibility of analogous studies at the microarchitectural and macro block level using statistical inference.

Optimizing Pipeline Depth. Most prior work in optimizing pipeline depth focuses exclusively on improving performance. Vector code performance is optimized on deeper pipelines while scalar codes perform better on shallower pipelines [Kunkel and Smith 1986]. A more general analytical pipeline model shows the optimal pipeline depth decreases with increasing overhead from partitioning logic between pipeline stages [Dubey and Flynn 1990].

Prior work also finds optimal pipeline depths from simulation. In particular, detailed simulations of a four-way superscalar, out-of-order microprocessor with a memory execute pipeline identify a 10.7 FO4 performance optimal pipeline design for the SPEC2000 benchmarks [Hartstein and Puzak 2002]. Similarly, simulations for an Alpha 21264-like machine identify 8 FO4 as a performance optimal design running the SPEC2000 benchmarks [Hrishikesh et al. 2002]. 18 FO4 delays is estimated to be the power-performance optimal pipeline design point for a single-threaded microprocessor [Zyuban et al. 2004]. Analytical modeling suggests depth multiplied by square-root of width should be constant for optimality [Eyerman et al. 2009].

Optimizing Heterogeneity. Heterogeneous cores constructed from existing core designs or designed from scratch using a modestly sized design space improve power efficiency [Kumar et al. 2004]. In this prior work, design alternatives are evaluated with exhaustive simulation to illustrate the potential energy efficiency of heterogeneity. In contrast, we implement a more thorough analysis, considering heterogeneity trends as the number of design compromises increases and heterogeneity limits as we explore the full continuum between complete homogeneity and complete heterogeneity. Both analyses are intractable in simulation for a diverse, broadly defined design space.

Heterogeneity might be viewed as per application customization. Fine-grained customization within an application naturally leads to custom hardware for different application phases. Such heterogeneity motivates microarchitectural adaptivity, which dynamically provisions hardware resources as required by the application. Regression models facilitate new studies of architectural adaptivity [Lee and Brooks 2008a], building on a large body of prior work [Albonesi et al. 2003].

Optimization Heuristics. While this article exhaustively evaluates regression models to assess trade-offs, iterative heuristics (e.g., gradient descent, genetic algorithms) may be required for larger spaces. When using such heuristics, the roughness or non-linearity of the performance-power topology impacts heuristic effectiveness [Eyerman et al. 2006]. Roughness metrics penalize the least squares fit for spline-based regression [Green and Silverman 1994]. For example, a roughness term may be added to the sum of square errors minimized in least squares. Accounting for roughness when fitting regression coefficients, this penalty approach favors smooth regression equations. Alternatively, we might use roughness metrics to characterize the performance-power to implement more effective optimization heuristics [Lee and Brooks 2008b].

8. CONCLUSIONS AND FUTURE DIRECTIONS

This article presents the case for applied statistical inference in microarchitectural design, proposing a simulation paradigm that (1) defines a comprehensive design space, (2) simulates sparse samples from that space, and (3) derives inferential regression models to reveal salient trends. These regression models accurately capture performance and power associations for comprehensive multi-billion point design spaces. As computationally efficient surrogates for detailed simulation, regression models enable previously intractable analyses of energy efficiency. This article demonstrates such capabilities for design characterization and optimization.

Statistical inference enables further research in pressing microarchitectural design questions. Statistical inference and the new capabilities demonstrated by this article also establish a strong foundation for interdisciplinary research across the hardware-software interface. Inferential models have the potential to capture design trends and compromises at each abstraction layer. Clean interfaces between models at each layer enable co-optimization across the hardware-software interface.

Future Methodologies. Other techniques in statistical inference may be applicable. Quantifying and comparing the accuracy and computational efficiency of these techniques is an avenue for future work. Machine learning techniques seek to automate model construction, removing the user from the derivation process. Heuristics and algorithms drive the derivation, eliminating the need for user feedback. These automated approaches are easier to adopt and use, but tend to be less efficient. Comparing the effectiveness of statistical inference and machine learning is an avenue for future work.

This article focuses primarily on predicting spatial characteristics, performing multivariate regression to model performance or power topology as a function of design parameters. In addition to this spatial dimension, computer system design often includes a temporal dimension where past system behavior may be indicative of future system behavior. Predicting events or behavior in time may require time series regression which identifies correlations in time

Multiprocessor Modeling. This article primarily considers microprocessor cores without considering their interactions within multiprocessors. Interactions might arise from communication through shared memory, contention for shared resources, and synchronization for parallel workloads. Models for microprocessor cores and mechanisms to account for interactions would provide a more thorough assessment of multiprocessor performance and power. Building on uniprocessor core models, a potential multiprocessor framework might use a combination of uniprocessor, contention, and penalty models [Lee et al. 2008].

A modular framework for homogeneous multiprocessors extends naturally to the heterogeneous sort by generalizing the uniprocessor model with libraries of inferential models containing one model for each core type; each model would encapsulate the performance and power trends for each core's design space. The library would include models for both general-purpose and special-purpose cores.

Hardware-Software Interface. Statistical inference and regression modeling establishes a strong foundation for interdisciplinary research across the hardware-software interface. Inferential models may be constructed to encapsulate performance and power trends at each abstraction layer. Given such models, clean interfaces between models are needed for optimization across abstraction layers.

Application performance optimization is increasingly important as they are ported to novel architectures. Effective performance tuning eases the transition by parameterizing the application with knobs that impact performance. The optimal knob configurations vary from platform to platform, requiring models to explore this space. For example, parameterized numerical methods and scientific computing applications will expose knobs for data decomposition (*i.e.*, blocks of work), processor topology (*i.e.*, processor assignments to those blocks), and algorithms (*i.e.*, numerical algorithms used for each block). Early results in applying statistical machine learning to numerical methods are promising [Lee et al. 2007].

Effective back-end compiler optimizations are critical to delivering application performance, but the effects and interactions between individual optimizations are highly complex and non-intuitive. Identifying the best combination of optimization flags to activate is difficult. Iterative compilation techniques search the space of optimizations to optimize metrics, such as performance, energy, and code size [Cooper et al. 1999; Kulkarni et al. 2005; Triantafyllis et al. 2005]. Statistical machine learning further improves search efficiency [Cavazos and O'Boyle 2006]. These predictive models encapsulate the performance trends in back-end compiler optimizations.

Lastly, below the microarchitectural interface, transistor tuning becomes increasingly important in nanoscale technologies. Not only must transistors be sized correctly, circuit delay analyses must account for process variations and statistical deviations from nominal sizes. Statistical inference and machine learning may be applied to capture relationships between circuit delays and device parameters (*e.g.*, transistor length, width, threshold voltage). Such predictive models might be trained with data from detailed circuit simulations and used for circuit tuning, statistical timing analysis, and Monte Carlo experiments to evaluate process variations. Early results in linking circuit and architecture models are promising [Azizi et al. 2010; Liang et al. 2009; Lovin et al. 2009].

Statistical inference and its capabilities in performance and power analysis extend across the hardware-software interface. Inference is extensible and might be applied at each abstraction layer, ranging from applications to devices. Interfaces between adjacent layers might enable composable inference where models combine to provide designers a holistic view of computing. Achieving such a vision requires best-known practices in statistical inference, machine learning, and optimization heuristics to deliver microarchitectural efficiency.

REFERENCES

Albonesi, D., Balasubramonian, R., Dropsho, S., Dwarkadas, S., Friedman, E., Huang, M., Kursun, V., Magklis, G., Scott, M., Semezaro, G., Bose, P., Buyuktosunoglu, A., Cook, P., and Schuster, S. 2003. Dynamically tuning processor resources with adaptive processing. *IEEE Computer 36,* 12, 49–58.

Azizi, O., Stevenson, J., Patel, S., and Horowitz, M. 2010. An integrated framework for joint design space exploration of microarchitecture and circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDAA, Leuven, Belgium.

Brooks, D., Bose, P., Schuster, S., Jacobson, H., Kudva, P., Buyuktosunoglu, A., Weller, J.-D., Zyuban, V., Gupta, M., and Cook, P. 2000. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro 20,* 6, 26–44.

Brooks, D., Bose, P., Srinivasan, V., Gschwind, M., Emma, P., and Rosenfield, M. 2003. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM Journal of Research and Development 47,* 5/6, 653–670.

Cavazos, J. and O'Boyle, M. 2006. Method-specific dynamic compilation using logistic regression. In *Proceedings of the 21st Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*. IEEE Computer Society, Washington, DC, 229–240.

Cooper, K., Schielke, P., and Subramanian, D. 1999. Optimizing for reduced code space using genetic algorithms. In *Proceedings of the Workshop on Languages, Compilers, and Tools for Embedded Systems*. ACM, New York, NY, 1–9.

Dubach, C., Jones, T., and O'Boyle, M. 2008. Microarchitectural design space exploration using an architecture-centric approach. In *Proceedings of the 40th Annual International Symposium on Microarchitecture*. IEEE Computer Society, Washington, DC, 262–271.

Dubey, P. and Flynn, M. 1990. Optimal pipelining. *Journal of Parallel and Distributed Computing 8,* 1, 10–19.

Eeckhout, L. and H. Vandierendonck, K. D. 2003. Quantifying the impact of input data sets on program behavior and its applicadtions. *Journal of Instruction-Level Parallelism 5*.

Eeckhout, L., Nussbaum, S., Smith, J., and DeBosschere, K. 2003. Statistical simulation: Adding efficiency to the computer designer's toolbox. *IEEE Micro 23,* 5, 26–38.

Eyerman, S., Eeckhout, L., and DeBosschere, K. 2006. Efficient design space exploration of high performance embedded out-of-order processors. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDAA, Leuven, Belgium, 351–356.

Eyerman, S., Eeckhout, L., Karkhanis, T., and Smith, J. 2009. A mechanistic performance modeling for studying resource scaling in out-of-order processors. *ACM Transactions on Computer Systems 27,* 2, 1–37.

GREEN, P. AND SILVERMAN, B. 1994. *Nonparametric regression and generalized linear models: A roughness penalty approach.* Chapman and Hall/CRC, Boca Raton, FL.

HARRELL, F. 2001. *Regression modeling strategies.* Springer-Verlag, New York, NY.

HARTSTEIN, A. AND PUZAK, T. 2002. The optimum pipeline depth for a microprocessor. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*. IEEE Computer Society, Washington, DC, 7–13.

HENNESSY, J. AND PATTERSON, D. 2003. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, San Francisco, CA.

HRISHIKESH, M., FARKAS, K., JOUPPI, N., BURGER, D., KECKLER, S., AND SIVAKUMAR, P. 2002. The optimal logic depth per pipeline stage is 6 to 8 fo4 inverter delays. In *Proceedings of the 29th Annual Symposium on Computer Architecture*. IEEE Computer Society, Washington, DC, 14–24.

INTEL CORPORATION. 2001. Desktop performance and optimization for Intel Pentium 4 processor. *Intel Corporation White Paper 249438-01*.

IPEK, E., MCKEE, S., DE SUPINSKI, B., SCHULZ, M., AND CARUANA, R. 2006. Efficiently exploring architectural design spaces via predictive modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, NY, 195–206.

IYENGAR, V., TREVILLYAN, L., AND BOSE, P. 1996. Representative traces for processor models with infinite cache. In *Proceedings of the 2nd Symposium on High Performance Computer Architecture*. IEEE Computer Society, Washington, DC, 62–72.

JOSEPH, P., VASWANI, K., AND THAZHUTHAVEETIL, M. J. 2006a. Construction and use of linear regression models for processor performance analysis. In *Proceedings of the 12th Symposium on High Performance Computer Architecture*. IEEE Computer Society, Washington, DC, 99–108.

JOSEPH, P., VASWANI, K., AND THAZHUTHAVEETIL, M. J. 2006b. A predictive performance model for superscalar processors. In *Proceedings of the 39th Annual International Symposium on Microarchitecture*. IEEE Computer Society, Washington, DC, 161–170.

KARKHANIS, T. AND SMITH, J. 2007. Automated design of application specific superscalar processors: An analytical approach. In *Proceedings of the 34st Annual Symposium on Computer Architecture*. ACM, New York, NY, 402–411.

KONGETIRA, P., AINGARAN, K., AND OLUKOTUN, K. 2005. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro 25,* 2, 21–29.

KULKARNI, P., HINES, S., WHALLEY, D., HISER, J., DAVIDSON, J., AND JONES, D. 2005. Fast and efficient searches for effective optimization-phase sequences. *ACM Transactions on Architecture and Code Optimization 2,* 2, 165–198.

KUMAR, R., TULLSEN, D., RANGANATHAN, P., JOUPPI, N., AND FARKAS, K. 2004. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*. IEEE Computer Society, Washington, DC, 64–75.

KUNKEL, S. AND SMITH, J. 1986. Optimal pipelining in supercomputers. In *Proceedings of the 13th Annual International Symposium on Computer Architecture*. IEEE Computer Society, Los Alamitos, CA, 404–411.

LEE, B. AND BROOKS, D. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, NY, 185–194.

LEE, B. AND BROOKS, D. 2008a. Efficiency trends and limits from comprehensive microarchitectural adaptivity. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, NY, 36–47.

LEE, B. AND BROOKS, D. 2008b. Roughness of microarchitectural design topologies and its implications for optimization. In *Proceedings of the 14th Symposium on High Performance Computer Architecture*. IEEE Computer Society, Washington, DC, 240–251.

LEE, B., BROOKS, D., DE SUPINSKI, B., SCHULZ, M., SINGH, K., AND MCKEE, S. 2007. Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of* *the 12th Symposium on Principles and Practice of Parallel Programming*. ACM, New York, NY, 249–258.

Lee, B., Collins, J., Wang, H., and Brooks, D. 2008. CPR: composable performance regression for scalable multiprocessor models. In *Proceedings of the 41st International Symposium on Microarchitecture*. IEEE Computer Society, Washington, DC, 270–281.

Liang, X., Lee, B., Wei, G.-Y., and Brooks, D. 2009. Design and test strategies for microarchitectural post-fabrication tuning. In *Proceedings of XXVII International Conference on Computer Design*. 84–90.

Lovin, K., Lee, B., Liang, X., Brooks, D., and Wei, G.-Y. 2009. Empirical performance models for 3T1D memories. In *Proceedings of XXVII International Conference on Computer Design*. 398–403.

Markovic, D., Stojanovic, V., Nikolic, B., Horowitz, M., and Broderson, R. 2004. Methods for true energy-performance optimization. *IEEE Journal of Solid-State Circuits 39,* 8, 1282–1293.

Moudgill, M., Wellman, J., and Moreno, J. 1999. Environment for PowerPC microarchitecture exploration. *IEEE Micro 19,* 3, 9–14.

Noonburg, D. and Shen, J. 1994. Theoretical modeling of superscalar processor performance. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*. ACM, New York, NY, 52–62.

Nussbaum, S. and Smith, J. 2001. Modeling superscalar processors via statistical simulation. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. IEEE Computer Society, Washington, DC, 15–24.

Oskin, M., Chong, F., and Farren, M. 2000. HLS: Combining statistical and symbolic simulation to guide microprocessor designs. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*. ACM, New York, NY, 71–82.

Phansalkar, A., Joshi, A., Eeckhout, L., and John, L. 2005. Measuring program similarity: Experiments with SPEC CPU benchmark suites. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*. IEEE Computer Society, Washington, DC, 10–20.

Sherwood, T., Perelman, E., Hamerly, G., and Calder, B. 2002. Automatically characterizing large scale program behavior. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, NY, 45–57.

Sinharoy, B., Kalla, R., Tendler, J., Eickemeyer, R., and Joyner, J. 2005. Power5 system microarchitecture. *IBM Journal of Research and Development 49,* 4/5, 505–521.

Stone, C. and Koo, C. 1986. Additive splines in statistics. In *Proceedings of the Statistical Computer Section*. ASA, Washington, DC, 45–48.

Tarjan, D., Thoziyor, S., and Jouppi, N. 2006. CACTI 4.0. *HPL Tech Report HPL-2006-86*.

Triantafyllis, S., Vacharajani, M., and August, D. 2005. Compiler optimization space exploration. *Journal of Instruction-Level Parallelism 7*.

Wunderlich, R., Wenisch, T., Falsafi, B., and Hoe, J. 2003. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*. ACM, New York, NY, 84–97.

Yi, J., Lilja, D., and Hawkins, D. 2005. Improving computer architecture simulation methodology by adding statistical rigor. *IEEE Computer 54,* 11, 1360–1373.

Zyuban, V., Brooks, D., Srinivasan, V., Gschwind, M., Bose, P., Strenski, P., and Emma, P. 2004. Integrated analysis of power and performance for pipelined microprocessors. *IEEE Transactions on Computers 53,* 8, 1004–1016.

Zyuban, V. and Kogge, P. 2001. Inherently lower-power high-performance superscalar architectures. *IEEE Transactions on Computers 50,* 3, 268–285.

Zyuban, V. and Strenski, P. 2003. Balancing hardware intensity in microprocessor pipelines. *IBM Journal of Research and Development 47,* 5/6, 585–598.

A. MODEL SPECIFICATION

The R specification of a performance model. Note the square-root transformation on the `bips` response.

The `rcs(p,k)` command implements restricted cubic splines on parameter `p` with `k` knots. Cubic splines fit piecewise cubic polynomials and restricted splines constrain the end pieces to use linear fits, which improve model behavior at the extreme regions of the space.

Interactions are specified by the `%ia%` operator. The `%ia%` operator specifies product terms between splines stripping out doubly-non-linear terms that arise when multiplying two cubic polynomials for pairwise interactions. Only terms that contain a linear factor are included, which controls model size when multiplying polynomials.

The power model is specified by replacing the `sqrt(bips)` response with the `log(power)` response.

```
m.app <- (sqrt(bips) ~(
                       # first-order effects
                       rcs(depth,4) + width + rcs(phys_reg,4)
                       + rcs(resv,3) + rcs(l2cache_size,3)
                       + rcs(icache_size,3) + rcs(dcache_size,3)

                       # second-order effects
                       # interactions of pipe dimensions and in-flight queues
                       + width %ia% rcs(depth,4)
                       + rcs(depth,4) %ia% rcs(phys_reg,4)
                       + width %ia% rcs(phys_reg,4)

                       # interactions of depth and hazards
                       + width %ia% rcs(icache_size,3)
                       + rcs(depth,4) %ia% rcs(dcache_size,3)
                       + rcs(depth,4) %ia% rcs(l2cache_size,3)

                       # interactions in memory hierarchy
                       + rcs(icache_size,3) %ia% rcs(l2cache_size,3)
                       + rcs(dcache_size,3) %ia% rcs(l2cache_size,3)
                       ));
```