SPATIAL SAMPLING AND REGRESSION STRATEGIES

THIS NEW SIMULATION PARADIGM FOR MICROARCHITECTURAL DESIGN EVALUATION AND OPTIMIZATION COUNTERS GROWING SIMULATION COSTS STEMMING FROM THE EXPONENTIALLY INCREASING SIZE OF DESIGN SPACES. THE AUTHORS DEMONSTRATE HOW TO OBTAIN A MORE COMPREHENSIVE UNDERSTANDING OF THE DESIGN SPACE BY SELECTIVELY SIMULATING A MODEST NUMBER OF DESIGNS FROM THAT SPACE AND THEN MORE EFFECTIVELY LEVERAGING THE SIMULATION DATA USING TECHNIQUES IN STATISTICAL INFERENCE.

Benjamin C. Lee
David M. Brooks
Harvard University

Microarchitectural design space exploration is often inefficient and ad hoc owing to the significant computational costs of current simulator infrastructures. Although simulators provide insight into application performance for a broad range of microarchitectural designs, the inherent costs of modeling microprocessor execution result in long simulation times. These simulation costs are further exacerbated for multicore, multithreaded architectures, which significantly contribute to exponentially increasing design space sizes. Potentially, a designer might perform m^p simulations for a design space of p parameters, each of which might take one of m values. Designers circumvent these challenges by constraining the design space using parameter subsets (p) or reducing the design space resolution (m). However, by subjectively determining these constraints at the study's outset on the basis of experience or intuition, the designer risks obtaining conclusions that simply reinforce prior inclinations and thereby limit the study's value.

In conjunction with design space growth, designers are increasingly differentiating their market segments and targeting different metrics on the basis of each segment's priorities—for example, single-thread latency, aggregate throughput, or energy. These trends will lead to increasing diversity in the set of designs considered interesting and viable for implementation. For example, the Intel Core, IBM Power 5, and Sun UltraSparc T1 inhabit very different parts of the design space, yet each was considered viable for implementation. This trend toward increasing design diversity will require tractable techniques to quantify trends across comprehensive design spaces, so that designers can compare and evaluate options from very different parts of a space.

These challenges in microarchitectural design motivated our formulation of a new simulation paradigm. By this paradigm, designers will achieve a more comprehensive understanding of the design space by selectively simulating a modest number of designs from that space and then more

Published by the IEEE Computer Society.

0272-1732/07/$20.00 © 2007 IEEE

efficiently leveraging the simulation data using techniques in statistical inference. The 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 lets us decouple the high resolution of the design space from the number of simulations required to identify a trend within that design space. Finally, we construct regression models using these sparsely sampled simulations to enable prediction for metrics of interest. These models' predictive ability and computational efficiency enable new capabilities in microarchitectural design optimization.

This tutorial details each of the three elements in the simulation paradigm. We define a large, high-resolution design space of nearly one billion points and evaluate sampled points within this space using cycle-accurate simulation. We describe the synergies between spatial sampling and existing techniques for controlling simulation costs. We propose performing uniform at random (UAR) design sampling for simulation, and we compare this approach to several alternatives. We show how to perform spline-based regression, using several supporting statistical analyses to produce robust, efficient models. To illustrate the derivation more concretely, we interleave code and scripts from R, an open-source software environment for statistical computing. These example scripts and data sets are available online for download (www.seas.harvard.edu/~bclee/publications.html). Overall, we show that regression models are accurate for predicting microarchitectural design metrics and applicable to practical design optimization.

Simulation framework

We use Turandot, a generic and parameterized, out-of-order, superscalar processor simulator.[1] Turandot is enhanced with PowerTimer to obtain power estimates based on circuit-level power analyses and resource utilization statistics.[2] Turandot's modeled baseline architecture is similar to the current Power4 and Power5 architectures. The simulator has been validated against both a Power4 register-transfer-level (RTL) model and a hardware implementation. This simulator implements pipeline depth performance and power models on the basis of prior work.[3] Power scales superlinearly as pipeline width increases, using scaling factors derived for an architecture with clustered functional units.[4] Cache power and latencies scale with array size according to Cacti.[5] We do not leverage any particular feature of the simulator in our models and our framework can generally be applied to other simulation frameworks with similar accuracy.

Benchmark suite

We consider SPECjbb, a Java server benchmark, and eight computation-intensive benchmarks from SPEC2k (ammp, applu, equake, gcc, gzip, mcf, mesa, and twolf). We report experimental results based on PowerPC traces of these benchmarks. The SPEC2k traces we used in this study were sampled from the full reference input set to obtain 100 million instructions per benchmark program.[6] We performed systematic validation to compare the sampled traces against the full traces to ensure accurate representation. Our benchmark suite is representative of larger suites frequently used in the microarchitectural research community.[7] We do not leverage any particular benchmark feature in model formulation, and our framework is generally applicable to other workloads with similar accuracy.

Design space

The first element of the simulation paradigm is the definition of the design space. This tutorial demonstrates spatial sampling and regression modeling for the design space described in Table 1. Parameters within a set are varied together to avoid fundamental design imbalances. The range of values considered for each parameter group is specified by a set of values, S_1, …, S_{12}. The Cartesian product of these sets,

$$S = \Pi_{i=1}^{12} S_i,$$

defines the entire design space. The cardi-

Table 1. Design space; *i*::*j*::*k* denotes a set of possible values from *i* to *k* in steps of *j*.

Set	Parameters	Measure	Range	$\lvert S_i \rvert$
S_1: Depth	Depth	FO4*	9::3::36	10
S_2: Width	Width	Instruction bandwidth	4,8,16	3
	Load and store reorder 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 and memory	Entries	10::2::28	
	Floating-point	Entries	5::1::14	
S_5: Instruction L1 cache	I-L1 cache size	$\log_2$ (entries)	7::1::11	5
S_6: Data L1 cache	D-L1 cache size	$\log_2$ (entries)	6::1::10	5
S_7: L2 cache	L2 cache size	$\log_2$ (entries)	11::1::15	5
	L2 cache latency	Cycles	6::2::14	
S_8: Control latency	Branch latency	Cycles	1,2	2
S_9: Fixed-point latency	ALU latency	Cycles	1::1::5	5
	FX-multiply latency	Cycles	4::1::8	
	FX-divide latency	Cycles	35::5::55	
S_{10}: Floating-point latency	FPU latency	Cycles	5::1::9	5
	FP-divide latency	Cycles	25::5::45	
S_{11}: Load/store latency	Load and store latency	Cycles	3::1::7	5
S_{12}: Memory latency	Main-memory latency	Cycles	70::5::115	10

* FO4: fan-out of four delays per pipeline stage.

nality of this product is $|S| = 9.38 \times 10^8$, or approximately one billion design points. Fully assessing the performance for each of the nine benchmarks on these configurations would further scale the number of simulations.

Spatial sampling

Spatial sampling, the second part of our proposed simulation paradigm, decouples the size of the design space from the number of simulations required to understand design trends by selectively simulating a modest number of points within the space.

Spatial and temporal synergies

Prior efforts to control simulation costs have focused primarily on temporal sampling. These techniques obtain samples from instruction traces in the time domain, reducing the costs per simulation by reducing the size of simulator inputs. Eeckhout et al. study profiling techniques to simplify workloads in microarchitectural simulation.[8] Nussbaum and Smith examine similar approaches for superscalar and symmetric multiprocessor simulation.[9] Both profile benchmarks to construct smaller, synthetic benchmarks with similar characteristics. Sherwood and Wunderlich separately propose techniques to identify representative simulation points within an instruction trace to reduce the total number of instructions simulated.[10,11] Temporal sampling effectively decouples the number of simulated instructions from the program length to reduce per simulation costs. However, it does not impact the number of simulations required to identify trends within a large design space. This limitation often constrains design space exploration since space sizes increase exponentially.

To control exponentially increasing design space sizes, we must therefore supplement temporal sampling with spatial sampling, an orthogonal technique that samples points from the design space for simulation. Figure 1 illustrates the combination of these techniques for regression model construction. Spatial sampling also mitigates the intractability and inefficiencies of traditional techniques that sweep design parameter values and exhaustively simulate all points defined within a constrained space. By decoupling the size of the space from the number of simulations required to extract design trends, spatial sampling enables the study of larger, higher-resolution design spaces. Specifically, this technique lets us consider many design parameters simultaneously, and each parameter can assume many different values.

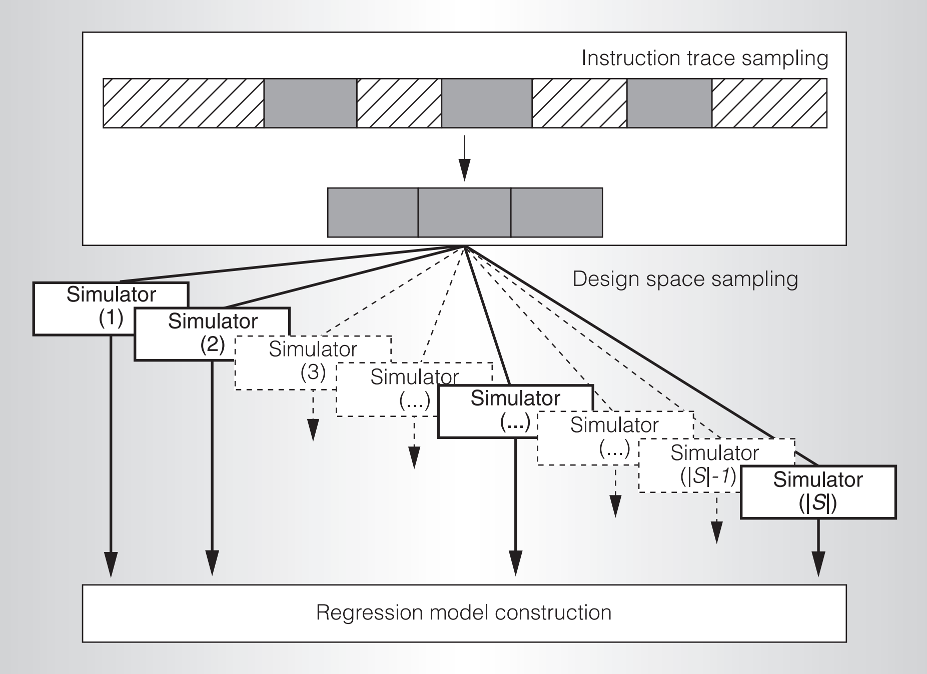


Figure 1. Simulation paradigm: temporal and spatial sampling reduces per-simulation costs and number of required simulations.

Uniformly random sampling

We propose sampling designs uniformly at random from the design space *S*. This approach provides observations drawn from the full range of parameter values. An arbitrarily large number of values can be included in each parameter's range because the number of simulations is decoupled from parameter resolution. Furthermore, UAR sampling does not bias simulated data toward particular designs. It produces, on average, equal representation for each parameter value in the set of sampled designs.

Suppose we treat the designs for which responses are not simulated as missing data from a full data set with all $|S|$ simulations. Then UAR sampling ensures the simulations are *missing completely at random* (MCAR). Under MCAR, data elements are missing for reasons unrelated to any characteristic or response of the design. In this context, the fact that a design point is unobserved is unrelated to the design's performance, power, or configuration.

In contrast, *informative missing* describes the case when elements are more likely to be missing if their responses are systematically higher or lower. For example, simulator limitations can prevent data collection for very low-performance architectures, and a configuration's "missingness" correlates with its performance. In such cases, the missingness is not ignorable, and we must formulate an additional model to predict whether the simulator can observe a design point. With UAR sampling from the design space, we ensure that observations are MCAR and so avoid such modeling complications.

We construct regression models using 1,000 design space samples. Each sampled design is simulated for every benchmark. Simulator-reported performance and power numbers provide the data necessary for the regression model construction. Although we use UAR-obtained samples from the design space, we also survey several alternative sampling strategies.

Alternative sampling strategies

Researchers have applied several other sampling strategies to increase the predictive accuracy of machine-learning models (such as neural networks) for the microarchitectural design space. Although we consider spline-based regression models, these various sampling strategies are broadly applicable to predictive modeling. These alternatives to UAR generally increase sampling 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 simulated samples from the broader space. This technique weights emphasized samples to increase their influence during model training. Weighted sampling can be used to improve model accuracy for design space regions known to exhibit greater error.
- *Regional sampling* also emphasizes samples from particular design regions given samples from the broader space. Instead of weighting, this approach specifies a region of interest and excludes undesired samples during model training. Regional sampling can be used to construct strictly localized models from UAR-collected samples. This approach might be necessary if regions of interest are unknown prior to sampling.[12]
- *Intelligent and 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 could improve accuracy. These samples are iteratively added to the training set, with each iteration choosing a sample that has a large error variance and that is the most different from those already added.[13]
- *Latin hypercube-sampling and space-filling* techniques 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.[14]

Although 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 than UAR sampling. Identifying samples for inclusion in regional sampling requires computing Euclidean distances between all collected samples, an expensive operation that must be performed for each region of interest. While UAR sampling is completely parallel, adaptive sampling introduces a feedback loop that limits this parallelism. Hypercube sampling and space filling guarantee sample properties that are only approximated by UAR sampling, but the trade-off between complexity and model accuracy is still an open question. Collectively, these sampling strategies provide options for improving the accuracy of the models constructed with the samples. We have found, however, that UAR sampling provides the basis for sufficiently accurate models and comprehensive design optimization.

Regression modeling

Now we turn to the background for relevant statistics and regression theory. The statistically rigorous derivation of microarchitectural performance and power models emphasizes the role of domain-specific knowledge when specifying the model's functional form, leading to models consistent with prior intuition about the design space. Furthermore, statistical significance testing prunes unnecessary and ineffective predictors to improve model efficiency. Specifically, we construct models with the following steps:

- *Hierarchical clustering*. Clustering examines correlations between potential predictors and enables elimination of redundant predictors. Predictor pruning controls model size, thereby reducing the risk of overfitting and improving model efficiency during formulation and prediction.

- *Association analysis*. Scatterplots qualitatively capture approximate trends of predictor-response relationships, revealing the degree of nonmonotonicity or nonlinearity. Scatterplots with low response variation as predictor values change may suggest predictor insignificance, enabling further pruning.
- *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 nonlinear transformations for each predictor.
- *Model specification*. We use domain-specific knowledge to specify predictor interaction. We use correlation analysis to specify the degree of flexibility in nonlinear transformations. Predictors more highly correlated with the response require more flexibility because any lack of fit for these predictors will more greatly affect overall model accuracy. Given the model's functional form, least-squares optimization fits the regression coefficients.
- *Assessing fit*. Multiple-correlation statistic R^2 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. Finally, predictive ability is assessed by predicting performance and power for a set of randomly selected validation points.

This derivation assumes two sets of data are available: a sizable training set and a smaller validation set. Both data sets are assumed to be sampled UAR from the design space and evaluated via detailed microarchitectural simulations. Determining the required number of samples to achieve a particular level of accuracy prior to model construction is difficult. The amount of required training data likely depends on the roughness of the design topology as well as the complexity of relationships between design parameters and metrics of interest. Since these effects are largely unknown a priori, the model derivation may proceed iteratively. If the models obtained from a given training set are inaccurate or biased, we may collect additional training data to improve sample resolution or extend design space boundaries to reduce extrapolation errors when predicting designs near existing boundaries. The derivation process is then repeated with additional iterations as necessary. In practice, however, very few iterations are necessary if samples and design spaces are specified conservatively (that is, large training sets, comprehensive design spaces).

Preliminaries

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 implemented by Harrell.[15] Sources, binaries, and documentation for R are available from the Comprehensive R Archive Network (http://www.r-project.org). Manuals providing a broad introduction to R commands are also available online at the R-project Web site. This tutorial shows how to implement regression techniques using the R statistical computing package, and how to interpret the resulting data and figures. Before beginning the regression modeling process we load necessary libraries and preprocess simulation data to be used for model training.

Implementation. We first load the necessary libraries for regression and graphics support. The `Hmisc` and `Design` packages provide support for nonlinear regression modeling, and the `lattice` library provides support for trellis graphics. We use `read.table` to read simulated data residing in a tab-delimited text file (`sep`) with column headers (`header`). The data sets for model construction and validation are placed in separate data frames (`data_*.df`), a structured data type that enables named fields. Field names are extracted from the text files' column names and are accessed with the **$** operator.

For example, we read two files containing data for model training (**data_model.txt**) and validation (**data_valid.txt**). Suppose each file contains three columns headed by column names of **depth**, **width**, and **bips**. We can then extract the column of performance data by invoking **data_model.df$bips**. We assume this data set contains simulator-reported performance and power values for the design parameters of Table 1, and application characteristics (such as cache miss rates).

```
## Load Libraries and Data
 library(Hmisc, T);
 library(Design, T);
 library(lattice);
data_model.df = read.table(
  file = ‘‘data_model.txt’’,
  sep = ‘‘\t’’, header = T);
data_valid.df = read.table(
  file = ‘‘data_valid.txt’’,
  sep = ‘‘\t’’, header = T);

## Data Description
## Use information about
## data distribution to
## set options for other
## Design, Hmisc functions

describe(data_model.df);
dd<=datadist(data_model.df);
options(datadist='dd');
```

Analysis. Invoking the **describe** function provides summary statistics of variables in the data frame. The following summary for performance, **bips**, specifies the number of observations (2,000, none missing, all unique). The mean and various quantiles (0.05 to 0.95 in increments of 0.05) provide a sense of the data distribution. For example, 10 percent of simulated **bips** in the training data are less than or equal to 0.357. In this case, the mean is close to the median, suggesting a symmetric distribution. Finally, the lists of "lowest" and "highest" values report five outliers at both extremes.

```
bips
n      missing  unique  Mean
2000   0        2000    0.764

.05    .10      .25     .50
0.261  0.357    0.497   0.712

.75    .90      .95
0.977  1.260    1.418

lowest: 0.114 0.119 0.120
0.123 0.128

highest: 2.106 2.135 2.298
2.465 2.809
```

Hierarchical clustering

Data clustering classifies N data elements into clusters according to a measure of similarity represented by a symmetric $N \times N$ matrix, S, where $S(i, j)$ quantifies the similarity between data elements i and j. Hierarchical clustering is an iterative approach that identifies successive clusters on the basis of previously identified clusters. Specifically, the clustering algorithm implements the following steps:

- *Initialize*. Assign each element to its own cluster to obtain N single-element clusters.
- *Merge*. Combine the most similar pair of clusters into a single cluster.
- *Iterate*. Repeat the merge step until obtaining one N-element cluster.

The similarity between two clusters A and B is the maximum similarity between elements of each cluster: $\max\{S(x, y) : x \in A, y \in B\}$. We use the squared correlation coefficient to quantify the similarity of two variables, enabling the user to identify potential redundancy in the data set. If multiple predictors are highly correlated, a single representative predictor may capture the cluster's impact on the response. Similarly, if multiple responses are highly correlated, a single representative response may be modeled, because correlated responses will likely scale with the modeled response.

Pruning predictors is important to control model size, not only by controlling the number of predictors, but also by controlling the number of potential interactions between predictors. Smaller models are preferable because they reduce the number of sampled observations required for model formulation. Several studies that validated models on independent data sets have shown that a fitted regression model is likely reliable (without overfitting) when the number of samples is 20 times the number of model terms.[15]

Implementation. We perform clustering with **varclus** on the potential predictors and responses, specifying variables and the data frame where the variables are defined.

```
## Hierarchical Clustering
 v = varclus(~ (depth + width
   + phys_reg + resv
   + mem_lat + ls_lat
   + ctl_lat + fix_lat
   + fpu_lat + d2cache_lat
   + l2cache_size + icache_size
   + dcache_size
   + il1miss_rate + il2miss_rate
   + dl1miss_rate + dl2miss_rate
   + br_rate + br_stall
   + br_mis_rate
   + stall_inflight + stall_dmissq
   + stall_cast
   + stall_storeq + stall_reorderq
   + stall_resv + stall_rename
   + bips + base_bips),
   data = data_model.df);
 print(v);
 trellis.device(''pdf'',
 file=''varclus_plot.pdf'');
 plot(v);
 dev.off();
```

The ~ operator specifies a relationship between $x \sim y + z$, where x is the response and y and z are the predictors. For **varclus**, no response is needed, and the sum of terms on the right-hand side says we want to examine pairwise correlations in a similarity matrix. We correlate various design parameters ranging from pipeline depth to cache sizes. For illustrative purposes, we also examine application characteristics such as cache miss rates and sources of pipeline stalls when the applications run on a baseline Power4-like architecture. Lastly, we specify the data frame in which all of these variables are defined (**data=data_model.df**).

Storing the clustering results to **v** lets us print the similarity matrix. The **print(v)** command will produce a $p \times p$ matrix, where p is the number of predictors, and each entry contains pairwise correlation coefficients between variables. We can more easily observe the correlations between variables in a clustering figure, which we generate by creating a trellis device with the **trellis.device** function. We close the device with the **dev.off** function.

Analysis. Figure 2 presents the results of hierarchical clustering with correlation coefficients used as a similarity metric (larger ρ^2 indicates greater correlation). The level at which clusters connect indicates their degree of similarity. L1 and L2 misses due to instruction cache accesses are highly correlated. Separately examining the raw data, we find the absolute number of L2 cache misses from the instruction probes to be negligible, and we eliminate **il2miss_rate** from consideration. Similarly, the branch rate is highly correlated with the number of branch-induced stalls, and we eliminate **br_stall**.

Pipeline depth is highly correlated with latency, because we scale the original functional-unit latencies with depth. Because final latencies are a function of original latencies and pipeline depth, we choose to keep these original latency variables. Including both predictors lets us differentiate the performance impact of individual functional-unit latency changes from the global latency changes as depth varies. However, if we later determine these effects to be insignificant, we could remove these extra latency parameters to improve model efficiency. Similarly, we keep both the data L1 cache miss rate and the baseline performance predictors to differentiate cache performance from global performance.

Association analysis

Scatterplots qualitatively represent the association between variables. Such plots are useful for examining associations between predictors and responses, revealing

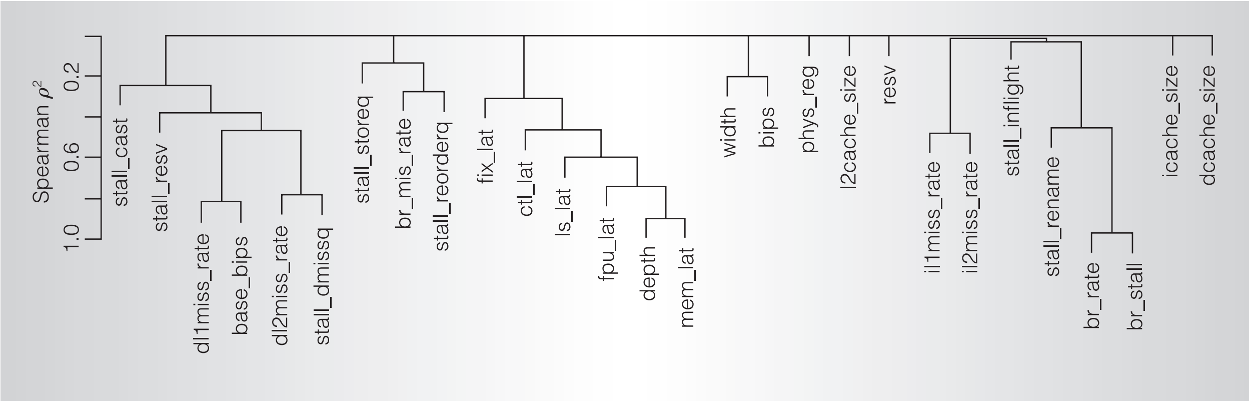


Figure 2. Hierarchical clustering: The level at which clusters connect indicates their degree of similarity. Spearman ρ^2 is a rank-based measure of correlation.

potential nonmonotonicity or nonlinearity. Scatterplots could quickly identify more significant predictors by showing, for example, a clear monotonic relationship with a response. Conversely, plots that exhibit low response variation despite a changing predictor value might suggest predictor insignificance. Overall, scatterplots let the user understand the parameter space quickly and at a high level.

Implementation. The **summary** function takes a relationship specified by the ~ operator. For example, **bips** ~ **depth** + **width** + **phys_reg** + **resv** says we wish to consider the relationship between performance and the four specified design parameters. The **summary** function divides the predictor domain into intervals. For each interval, it computes the average response of all samples in the interval. The following commands illustrate these commands for our data set.

```
## Association Analysis
   s = summary(bips ~ depth
     + width + phys_reg
     + resv,
     data = data_model.df)
 print(s);
 trellis.device(‘‘pdf’’,
 file=‘‘assoc_plot.pdf’’);
 plot(s);
 dev.off();
```

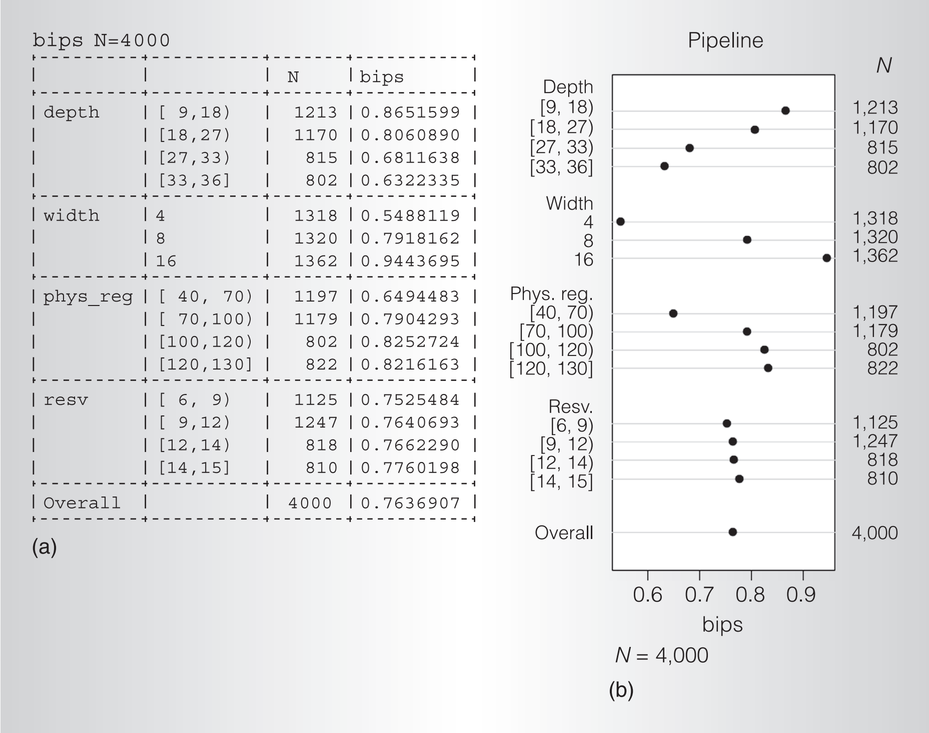
Analysis. Invoking the **print** function produces the data table of Figure 3a, relating predictor intervals to the response. For example, pipeline depth takes a value between 9 and 15 FO4 delays per stage for 1,213 of 4,000 samples. The average performance for these designs is 0.865 billion instructions per second. Figure 3b shows this data in scatterplot form, illustrating strong monotonic relationships between performance and pipeline dimensions. The register file size appears to have a significant but nonlinear relationship with performance. In contrast, the number of entries in reservation stations seems to have a negligible performance impact.

Correlation analysis

Although we observe qualitative associations between performance and parameter values using scatterplots, we would also like to quantify the strength of these associations. Correlation coefficients are a common metric for quantitatively assessing the strength of empirically observed relationships. Pearson’s correlation coefficient between two random variables is computed in the following equation, where X and Y are random variables with expectations μ_x and μ_y and standard deviations σ_x and σ_y:

$$\rho = \frac{E\left[(X - \mu_x)(Y - \mu_y)\right]}{\sigma_x \sigma_y} \tag{1}$$

When the distribution of X and Y are unknown, nonparametric statistics can be



```
bips N=4000
+---------+----------+-----+----------+
|         |          |N    |bips      |
+---------+----------+-----+----------+
|depth    |[ 9,18)   | 1213|0.8651599 |
|         |[18,27)   | 1170|0.8060890 |
|         |[27,33)   |  815|0.6811638 |
|         |[33,36]   |  802|0.6322335 |
+---------+----------+-----+----------+
|width    |4         | 1318|0.5488119 |
|         |8         | 1320|0.7918162 |
|         |16        | 1362|0.9443695 |
+---------+----------+-----+----------+
|phys_reg |[ 40, 70) | 1197|0.6494483 |
|         |[ 70,100) | 1179|0.7904293 |
|         |[100,120) |  802|0.8252724 |
|         |[120,130] |  822|0.8216163 |
+---------+----------+-----+----------+
|resv     |[ 6, 9)   | 1125|0.7525484 |
|         |[ 9,12)   | 1247|0.7640693 |
|         |[12,14)   |  818|0.7662290 |
|         |[14,15]   |  810|0.7760198 |
+---------+----------+-----+----------+
|Overall  |          | 4000|0.7636907 |
+---------+----------+-----+----------+
```

Figure 3. Association analysis: data table produced by the R **print** function, summarizing the range of parameter values (second column), number of samples in each range (third column), and average performance of these samples (fourth column) (a), and scatterplots visualizing the same data (b).

more robust. In particular, we prefer to use the Spearman rank correlation coefficient ρ_{sp} to quantify association independently of variable distribution. The computationally efficient approximation requires only d_i, the difference in ordinal rank of x_i in X and y_i in Y. Suppose $X = (x_1, x_2, \ldots, x_N)$ and $Y = (y_1, y_2, \ldots, y_N)$. Compute the rank of x_i in X, and the rank of y_i in Y. Spearman rank correlation computes a coefficient using the N differences in rank:

$$\rho_{\text{sp}} = \frac{\sum_{i=1}^{N} x_i y_i}{\sqrt{\sum_{i=1}^{N} x_i^2 \sum_{j=1}^{N} y_j^2}} \approx 1 - \left(6 \sum_{i=1}^{N} \frac{d_i^2}{N(N^2 - 1)}\right) \qquad (2)$$

Implementation. Given a predictor-response relationship, the **spearman2** function computes the squared Spearman rank correlation coefficient. Specifically, we compute the correlation of performance (left-hand side) against each predictor (right-hand side) of the relationship. For example, **spearman2(bips ~ depth, data = data_model.df)** computes the Spearman rank correlation between performance and pipeline depth. These coefficients may be printed and plotted.

```
## Correlation Analysis
 sp = spearman2(bips ~
   (depth + width
   + phys_reg + resv
   + mem_lat+ ls_lat
   + ctl_lat + fix_lat
   + fpu_lat + d2cache_lat
   + l2cache_size + icache_size
   + dcache_size
   + il1miss_rate+ dl1miss_rate
   + dl2miss_rate
   + br_rate + br_mis_rate
   + stall_inflight + stall_dmissq
   + stall_cast + stall_storeq
   + stall_reorderq + stall_resv
   + stall_rename + bips
```

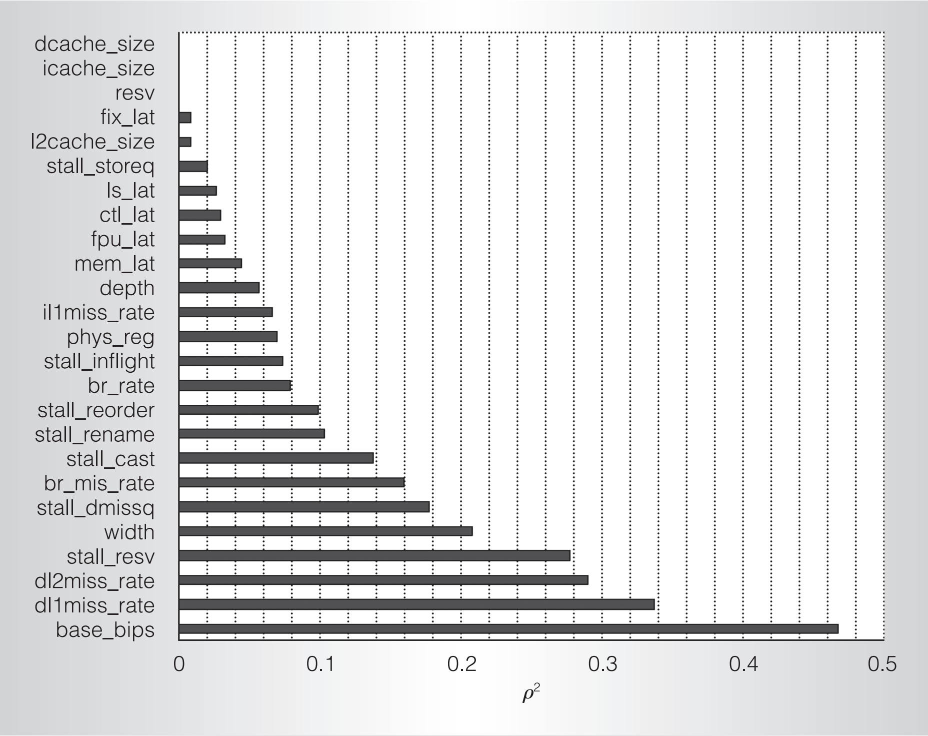


Figure 4. Correlation analysis: plots squared Spearman rank correlation coefficients for each parameter and characteristic.

```
  + base_bips),
  data = data_model.df);
print(sp);
trellis.device(‘‘pdf’’,
file=‘‘spearman_plot.pdf’’);
plot(sp);
dev.off();
```

Analysis. Figure 4 plots the nonmonotonic generalization of the Spearman rank correlation coefficient for each of the prespecified predictor variables. This information will guide our choice of the number of spline knots when specifying the functional form of the regression model. A greater number of spline knots provides greater flexibility and potentially better model fit. A lack of fit for predictors with higher ρ^2 will have a greater negative impact on performance prediction. For architectural predictors, a lack of fit will be more consequential (in descending order of importance) for width, depth, physical registers, functional-unit latencies, cache sizes, and reservation stations.

Application characteristics are the most highly correlated variables and are likely the primary determinants of performance. For this reason, we choose to formulate one model per benchmark in which these characteristics become invariant and may be dropped from the model specification. Given a particular architecture, performance varies significantly across applications, depending on their sources of bottlenecks. Keeping the application constant in the model eliminates this variance. Thus, per-benchmark models let us more effectively model performance because they consider only the microarchitectural impact on performance.

Model specification

We apply regression modeling techniques to obtain empirical estimates for metrics of interest efficiently. We apply a general class of models in which a response is modeled as a weighted sum of predictor variables plus random noise. Because basic models linear in the predictors might not adequately capture nuances in the response-predictor relationship, we also consider more advanced techniques to account for potentially nonlinear relationships.

Notation. 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 $y = y_1, \ldots, y_n$ denote the vector of observed responses. For a particular point i in this universe, let y_i denote its response variable and let $x_i = x_{i,1}, \ldots, x_{i,p}$ denote its p predictors. These variables are constant for a given point in the universe. Let $\beta = \beta_0, \ldots, \beta_p$ denote the corresponding set of regression coefficients used in describing the response as a linear function of predictors plus a random error e_i, as shown in the following equation:

$$f(y_i) = \beta g(x_i) + e_i$$
$$= \beta_0 + \sum_{j=1}^{p} \beta_j g_j(x_{ij}) + e_i \qquad (3)$$

Mathematically, β_j can be interpreted as the expected change in y_i per unit change in the predictor variable $x_{i,j}$. The e_i are assumed to be independent random variables with zero mean and constant variance; $E(e_i) = 0$ and $\mathrm{Var}(e_i) = \sigma^2$.

Transformations f and $g = g_1, \ldots, g_p$ can be applied to the response and predictors, respectively, to improve model fit by stabilizing a nonconstant error variance or accounting for nonlinear correlations between the response and predictors.

Fitting a regression model to observations, by determining the $p + 1$ coefficients in β, enables response prediction. The method of least squares is commonly used to identify the best-fitting model for a set of observations by minimizing the sum of squared deviations between the predicted responses given by the model and the actual observed responses. Thus, using the least-squares method, we find the $p + 1$ coefficients to minimize $S(\beta)$ 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 in Equation 3:

$$S(\beta_0, \ldots, \beta_p) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

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. We can model the interaction between two predictors by constructing a third predictor, $x_3 = x_1x_2$, to obtain $y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \beta_3x_1x_2 + e$. 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 $\beta_1 + \beta_3x_2$ is the expected change in y per unit change in x_1 for a fixed x_2.

Nonlinearity. Bacic linear regression often assumes that response behaves linearly in all predictors. This assumption is often too restrictive, and several techniques for capturing nonlinearity can be applied. The most simple of these techniques is a polynomial transformation on predictors suspected of having a nonlinear correlation with the response. However, polynomials have undesirable peaks and valleys. Furthermore, a good fit in one region of the predictor's values can unduly affect the fit in another region of values. For these reasons, we consider splines a more effective technique for modeling nonlinearity.

Spline functions are piecewise polynomials used in curve fitting. 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 lead to better fits. The following equation gives a restricted cubic spline on x with k knots $t_1, \ldots, t_k$, where $j = 1, \ldots, k - 2$:[16]

$$y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \ldots + \beta_{k-1}x_{k-1} \qquad (5)$$

$$x_1 = x \qquad (6)$$

$$x_{j+1} = (x - t_j)_+^3 - (x - t_{k-1})_+^3 (t_k - t_j) / (t_k - t_{k-1}) + (x - t_k)_+^3 (t_{k-1} - t_j)/(t_k - t_{k-1}) \qquad (7)$$

Each variable x_i in Equation 5 is given by either Equation 6 or 7. Equation 7 implements a piecewise cubic polynomial where t_i are knot locations and the + subscript denotes the positive part of an expression. For example, $(x - t)_+$ equals $x - t$ if $x - t$ is greater than zero and equals zero otherwise.

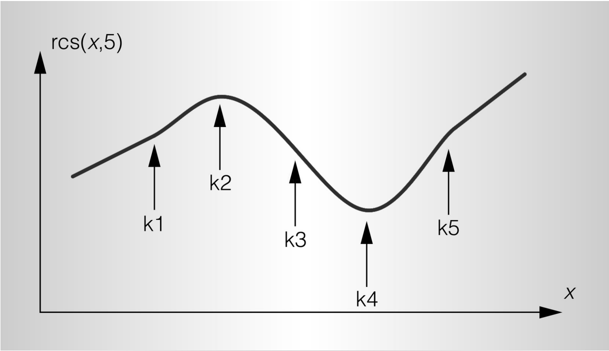


Figure 5. Restricted cubic-spline schematic.

We use restricted cubic splines because linear splines can be inadequate for complex, highly curved relationships. Splines of higher-order polynomials can offer better fits.[15] Unlike linear splines, cubic splines can be made smooth at the knots by forcing the first and second derivatives of the function to agree at the knots. However, cubic splines can behave poorly in the tails before the first knot and after the last knot.[16] Restricted cubic splines that constrain the function to be linear in the tails are often better behaved.

Figure 5 is a schematic illustration of a restricted cubic spline with five knots and linear tails. The choice and position of knots are variable parameters when specifying nonlinearity with splines. Placing knots at fixed quantiles of a predictor's distribution is a good approach in most data sets, ensuring a sufficient number of points in each interval.[16] In practice, five knots or fewer are generally sufficient for restricted cubic splines. Smaller data sets may require fewer knots. As the number of knots increases, flexibility improves at the risk of overfitting the data. In many cases, four knots offer an adequate fit of the model and a good compromise between flexibility and loss of precision from overfitting.

Implementation. We specify a regression model, predicting performance from various microarchitectural design parameters. We specify restricted cubic splines using the `rcs` function, which takes as inputs the predictor and the number of knots. Each cubic spline must use at least three knots. Predictors with greater performance correlations are assigned a greater number of knots. The `%ia%` operator allows restricted interactions between cubic splines by removing doubly nonlinear terms in the polynomial product. This operator is necessary to control the number of terms in a model with many polynomial interactions.

We draw on domain-specific knowledge to specify interactions. Pipeline depth likely interacts with cache sizes that impact hazard rates. A smaller L2 cache leads to additional memory hazards in the pipeline. These hazards affect, in turn, instruction throughput gains from pipelining. Thus, their joint impact must also be modeled. In a similar fashion, we expect pipeline width to interact with the register file. We also expect the memory hierarchy to interact with adjacent levels (for example, L1 and L2 cache size interaction). Although we capture most relevant interactions, we do not attempt to capture all significant interactions via an exhaustive search of predictor combinations. The model's accuracy suggests that this high-level representation is sufficient.

We apply a `sqrt` transformation on the `bips` response and specify its relationship to design parameters. The resulting model specification is assigned to `m`. We use the ordinary least-squares `ols` function to determine regression coefficients from the specification `m` and the training data. Least squares returns a fitted model `f`. We obtain a power model `g` from the performance model `f` by updating it with a `log` transformation on the power response and leaving the right side of the relationship unchanged, as indicated by the "`.`" (period) in `g = update(f, log(power) ~ .)`. We use a standard variance-stabilizing `sqrt` transformation on performance to mitigate any model biases and a `log` transformation on power to capture exponential trends in power as parameters (for example, depth) vary. Both `sqrt` and `log` transformations are standard in the statistics literature and are typically applied to more tractably and effectively analyze data with large values in absolute terms. The following code segment illustrates these commands for our data set.

```
## Model Specification and Fit
  m = (sqrt(bips) ~
  (## first-order effects
    rcs(depth,4) + width
    + rcs(phys_reg,4)
    + rcs(resv,3)
    + rcs(mem_lat,3)
    + fix_lat
    + rcs(fpu_lat,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)
  ));
f = ols(m, data=data_model.df);
g = update(f, log(power) ~ .);
```

Assessing fit

The model's fit to the sampled simulation data used in formulation is quantified with multiple-correlation statistic R^2 in Equation 10. This statistic quantifies regression error *SSE* (Equation 8) as a fraction of total error *SST* (Equation 9):

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \qquad (8)$$

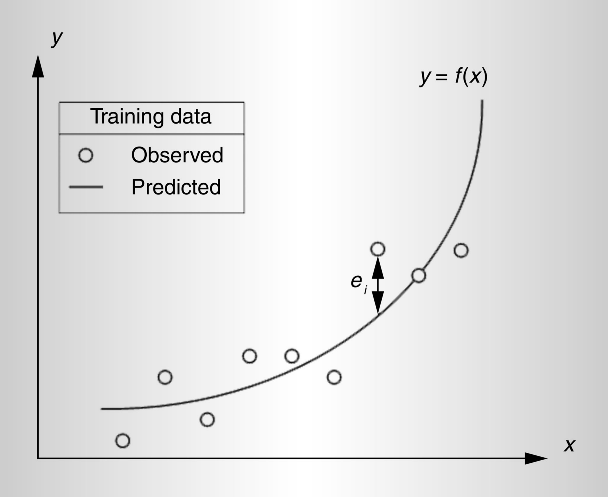


Figure 6. Residual schematic illustrating differences between observed and predicted values in training data.

$$SST = \sum_{i=1}^{n} \left(y_i - \frac{1}{n} \sum_{i=1}^{n} y_i \right)^2 \qquad (9)$$

$$R^2 = 1 - (SSE/SST) \qquad (10)$$

From Equation 10, R^2 will be 0 when the model error is just as large as the error from simply using the mean to predict responses. Larger values of R^2 suggest better fits for the observed data. Thus, R^2 is the percentage of variance in the response captured by the predictors. However, a value too close to 1 may indicate overfitting, a situation in which the model's worth is exaggerated and future observations may not agree with the modeled predictions. Overfitting typically occurs when too many predictors are used to estimate relatively small data sets.

The model should also be examined to ensure predictions exhibit no systematic bias based on an analysis of residuals in the following equation:

$$\hat{e}_i = y_i - \hat{\beta}_0 - \sum_{j=0}^{p} \hat{\beta}_j x_{ij} \qquad (11)$$

These residuals are per-sample differences between modeled and simulated performance in the training set as illustrated in

Figure 6. In particular, we should validate the following assumptions to ensure model robustness:

1. The residuals are not correlated with the predicted response.
2. The residuals' randomness is the same for all predicted responses.
3. The residuals have a normal distribution with zero mean and constant variance.

The first two assumptions are typically validated with scatterplots of residuals against predicted responses because such plots can reveal systematic deviations from randomness. The third assumption is typically validated by quantile-quantile plots, in which the quantiles of one distribution are plotted against another. Practically, this means ranking the n residuals $\hat{e}^{(1)}, \ldots, \hat{e}^{(n)}$, obtaining n ranked samples from the normal distribution $s^{(1)}, \ldots, s^{(n)}$, and producing a scatterplot of $(\hat{e}^{(i)}, s^{(i)})$ that should appear linear if the residuals follow a normal distribution.

Last, we obtain 100 additional randomly selected points from the design space and compare simulator-reported metrics against regression-predicted metrics. To visualize the error distribution for these validation points, we can use boxplots, which are graphical displays of data that measure location (median) and dispersion (interquartile range, IQR), identify possible outliers, and indicate the symmetry or skewness of the distribution. Boxplots are constructed by

- horizontal lines at median, upper, and lower quartiles;
- vertical lines drawn up from the upper quartile and down from the lower quartile to the most extreme data point within 1.5 times the IQR (the difference between the first and the third quartiles) of the upper and lower quartiles, with short horizontal lines to mark the end of vertical lines; and
- circles for each outlier.

Boxplots enable graphical analysis of basic error statistics. They also facilitate comparisons between multiple distributions, because comparing boxplots often reveals similarities more readily than comparing tables of error statistics.

Implementation. Multiple-correlation statistic R^2 is included in the model summary obtained by applying the **`print`** function to the formulated model. Residuals are plotted against the fitted values (that is, regression-predicted values) to ensure a lack of correlation between residuals and predictions, validating assumptions 1 and 2 of the previous section. The **`xYplot`** command uses the **`quantile`** method to stratify fitted values into groups of 20 elements (**`nx=20`**). This function plots the median, lower, and upper quantiles of the residuals for these groups of 20 observations. The grouping is necessary if there are too many observations for a standard scatterplot.

We validate the third assumption of the previous section (residual normality) by plotting ranked residuals against ranked samples from the normal distribution to produce a quantile-quantile plot. The **`qqnorm`** function automatically generates a plot that should appear linear if the residuals follow a normal distribution. The **`qqline`** function draws a line through the 25th and 75th percentile of the residuals to aid this analysis. The following code segment illustrates these commands for our data set.

```
## Model Summary:
## R-squared statistic
  print(f); print(g);

## Residual Analysis:
## (1) scatterplot and
## (2) quantile-quantile plot
  trellis.device
  (‘‘pdf’’,file=‘‘residf.pdf’’);
  xYplot(resid(f) ~ fitted(f),
    method=’quantile’,nx=20,
    xlab = ‘‘Fitted Values’’,
    ylab = ‘‘Residuals’’);
  dev.off();

  trellis.device
  (‘‘pdf’’,file=‘‘qqnormf.pdf’’);
  qqnorm(resid(f));
  qqline(resid(f));
  dev.off();
```

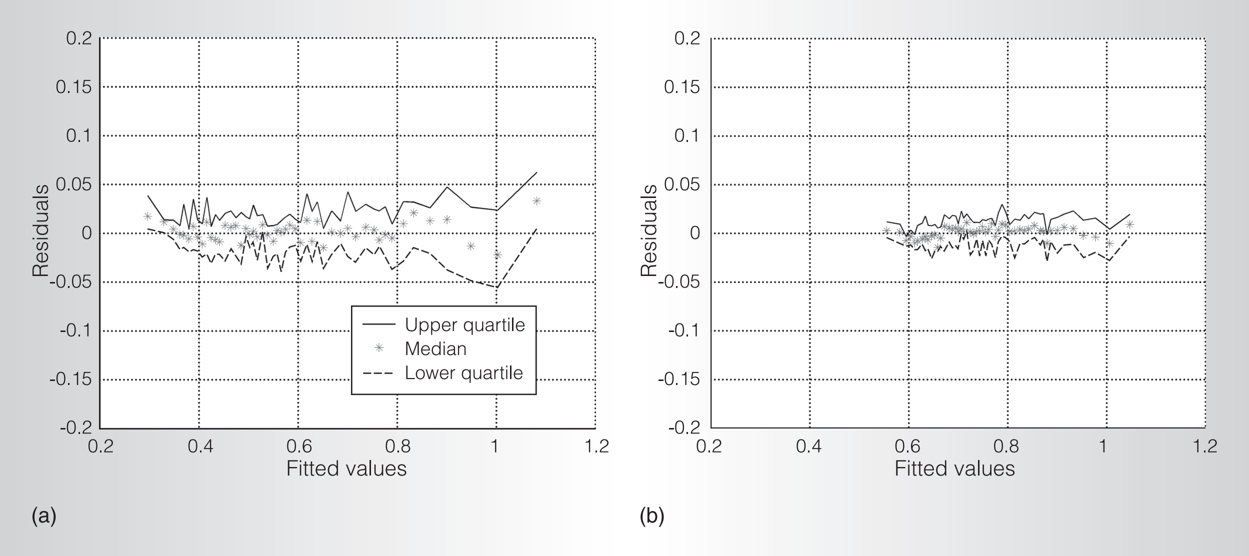


Figure 7. Residual analysis: scatterplots of residuals before (a) and after (b) square root transformation on the response. Residuals for an unbiased model should appear randomly and independently distributed around zero.

Variable `o` contains the observed true values for our validation points. The `predict` function takes a regression model object produced by the `ols` function and a set of new data for prediction. Predictions must be squared to invert the `sqrt` transformation on the performance response. The observations, predictions, and error rates are concatenated into a three-column matrix using `cbind` and written to a file. The error distribution is visualized using boxplots constructed with the `boxplot` function. The following code segment illustrates these commands for our data set.

```
## Model Prediction
  o = data_valid.df$bips;
  p = (predict(object=f,
    newdata=data_valid.df))ˆ2;
  e = abs(o-p) / o ;
  write.table(cbind(o,p,e),
    file=‘‘valid_bips.txt’’,
    sep = ‘‘\t’’,
    col.names=c(‘‘observed’’,
    ‘‘predicted’’, ‘‘error’’));
  pdf(‘‘bips_box.pdf’’);
  boxplot(e);
  dev.off();
```

Analysis. Multiple-correlation statistic R^2 is measured at 0.965 for performance and 0.993 for power, suggesting a good model fit to the training data of 1,000 sampled simulations. Overfitting should not be a concern, because the number of model terms is far smaller than the number of samples. Figure 7a suggests a correlation between residuals and fitted values for a nontransformed performance metric. Residuals are biased positive for the smallest and largest fitted values. Figure 7b indicates that the standard variance-stabilizing square-root transformation on performance reduces the magnitude of these correlations. Variance stabilization also causes residuals to follow a normal distribution more closely, as indicated by the linear trend in Figure 8.

Figure 9 indicates that the performance model achieves median errors ranging from 3.5 percent (for the ammp benchmark) to 11.1 percent (for mesa), with an overall median error across all benchmarks of 6.6 percent. Power models are slightly more accurate, with median errors ranging from 3.1 percent (mcf) to 6.5 percent (applu), and an overall median of 4.8 percent. Most predictions achieve error rates below 15 percent.

Design optimization

The proposed simulation paradigm effectively combines spatial sampling and

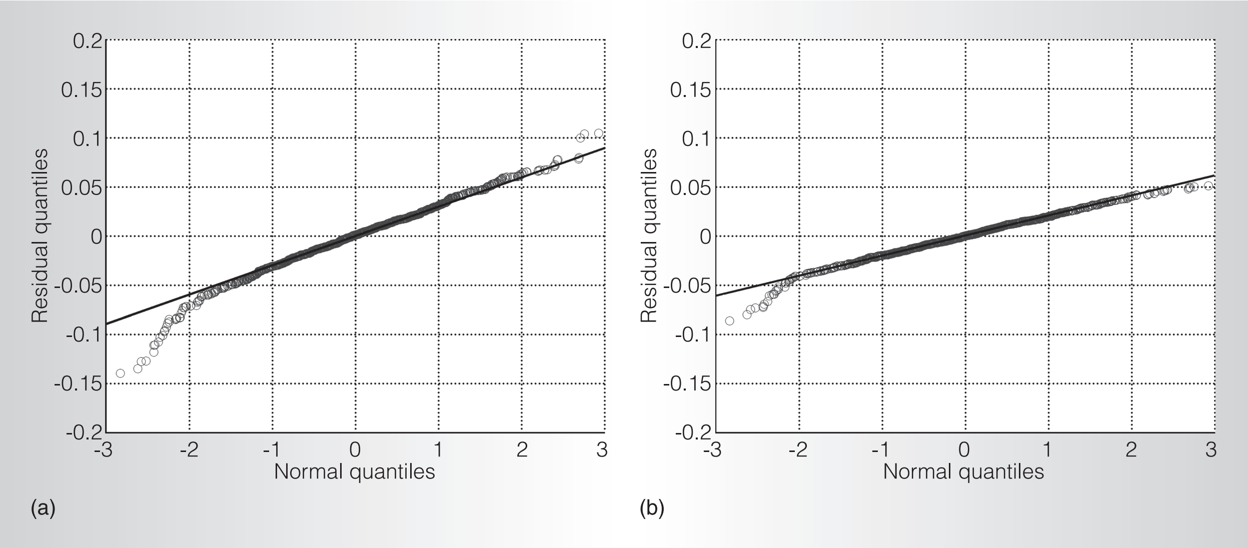


Figure 8. Residual analysis: quantile-quantile plots of residuals before (a) and after (b) square root transformation on the response. Plots for an unbiased model should appear linear.

regression modeling to increase the information content in a given number of sparsely simulated designs. Spatial sampling controls simulation costs by reducing the number of simulations required to identify a trend, whereas the computational efficiency of regression models enables hundreds of predictions per second. Collectively, these techniques provide the necessary framework for more comprehensive design space evaluations and new capabilities in traditional design optimization problems, including the following:

- *Design space characterization*. The computational efficiency of regression models enables the complete performance and power characterization of design spaces with hundreds of thousands of points. By exhaustively evaluating the models for every point in these spaces, we can quickly identify design bottlenecks and trends as design parameters vary.[17]
- *Pareto frontier analysis*. The complete design space characterization enables the construction of a Pareto frontier, which comprises designs that maximize performance for a given power budget or minimize power for a given performance target.[17]
- *Parameter sensitivity analysis*. Regression models let us consider all parameters simultaneously when performing parameter sensitivity analyses. For example, most earlier pipeline depth studies held non-depth parameters constant and considered the effects of varying depth around a particular baseline design. In contrast, regression models let us vary all parameters simultaneously and identify the best design at each depth. This approach eliminates any bottlenecks induced by assuming a single parameter varies independently from all other parameters.[17]
- *Heterogeneous multicore design*. The models enable comprehensive optimization for identifying effective core design compromises given a workload set. The models serve to identify per-workload optima (architectures maximizing a metric of interest). A clustering analysis on these optima produces design compromises for workloads requiring similar resources at the microarchitectural level. These compromises can form the basis for core design in heterogeneous multiprocessor architectures targeting these workloads.[17]

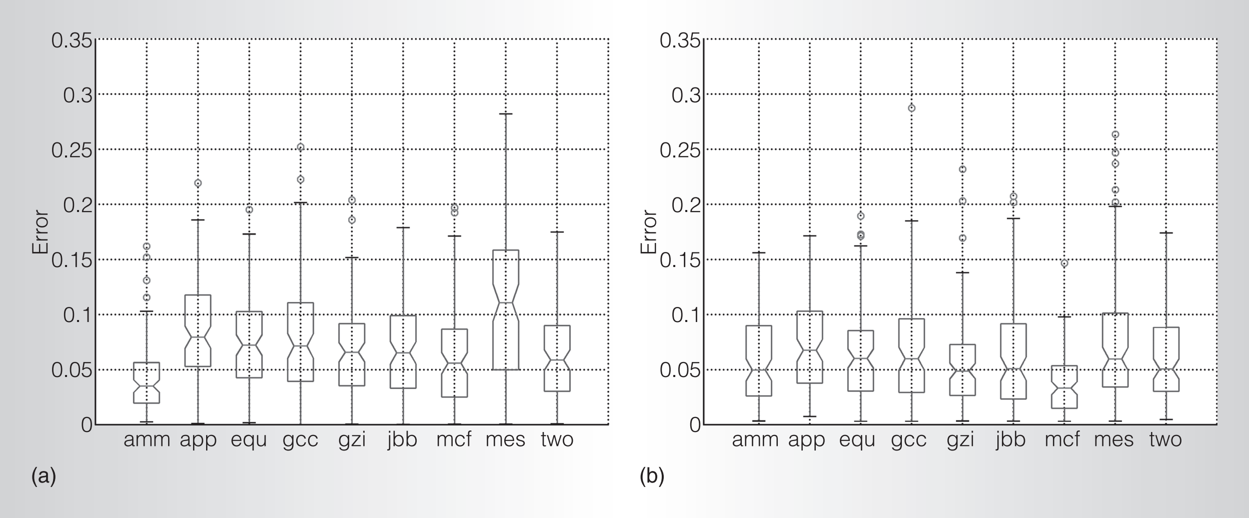


Figure 9. Error distributions: boxplots capture quartiles of model errors when predicting performance and power of validation set.

- *Topology visualization*. Microarchitectural designs occupy high-dimensional spaces. Regression models enable the visualization of performance and power topologies using 2D projections of the design space. Practically, these projections produce 2D contour maps for all pairs of design parameters in the space, illustrating nonlinearities or nonmonotonicities. These visualizations aid bottleneck analysis while motivating the need for nonlinear models.
- *Topology roughness metrics*. Given visualizations of design topologies, we can compare contour maps to subjectively assess the relative roughness of topologies. Roughness metrics supplement these contour maps, quantifying the range and variability of these contours. Furthermore, these metrics extend to higher dimensions and can be applied beyond 2D contours (for example, Equation 13 for d-dimensional roughness where $2m > d$ and $v_1,...,v_d$ are nonnegative integers such that $v_1 + ... + v_d = m$). In the microarchitectural context, $f(x)$ is a performance or power function and $x_1, \ldots, x_d$ are design parameters. Function $f(x)$ is approximated by regression models, whereas the derivatives and integrals can be approximated by differences and sums:

$$R_2 = \int_{x_2}\int_{x_1}\left[\left(\frac{\delta^2 f}{\delta x_1^2}\right)^2 + 2\left(\frac{\delta^2 f}{\delta x_1 x_2}\right)^2 + \left(\frac{\delta^2 f}{\delta x_2^2}\right)^2\right] dx_1 dx_2 \quad (12)$$

$$R_d = \int_{x_d}\cdots\int_{x_1}\sum\frac{m!}{v_1!\ldots v_d!} \times \left(\frac{\delta^m f}{\delta {x_1}^{v_1}\ldots\delta {x_d}^{v_d}}\right)^2 dx_1\ldots dx_d \quad (13)$$

- *Topology optimization*. Optimization through heuristic search is an alternative to exhaustive model evaluation. Techniques such as gradient descent and simulated annealing iteratively traverse the design space to identify optima. Each iteration requires comparisons between the current design and its neighbors based on a metric of interest. Although microarchitectural simulators could be invoked for every iteration,[18] using regression models for these comparisons significantly increases the heuristic's computational efficiency.

These capabilities, enabled by regression models, produce more comprehensive variants of traditional studies or allow new studies not possible with current usage patterns for microarchitectural simulators. Collectively, these capabilities motivated us to develop our new simulation paradigm on the basis of spatial sampling and regression modeling to more effectively use simulator cycles.

Detailed simulation will continue to play a significant role in microarchitectural design. Simulators enable early design space evaluations and are invaluable for assessing trade-offs. However, if we define simulation efficiency as the information content derived from a given number of simulation hours, we find that current simulator usage patterns are highly inefficient. Spatial sampling and statistical inference reduce these inefficiencies by identifying trends and constructing predictive models from a sparsely simulated design space. These efficiency gains translate into new capabilities in microarchitectural design and optimization. MICRO

Acknowledgments

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

References

1. M. Moudgill, J. Wellman, and J. Moreno, ''Environment for PowerPC Microarchitecture Exploration,'' *IEEE Micro*, vol. 19, no. 3, May–June 1999, pp. 9-14.
2. D. Brooks et al., ''New Methodology for Early-Stage, Microarchitecture-Level Power-Performance Analysis of Microprocessors,'' *IBM J. Research and Development*, vol. 47, no. 5 and 6, Oct.–Nov. 2003, pp. 653-670.
3. V. Zyuban et al., ''Integrated Analysis of Power and Performance for Pipelined Microprocessors,'' *IEEE Trans. Computers*, vol. 53, no. 8, Aug. 2004, pp. 1004-1016.
4. V. Zyuban, ''Inherently Lower-Power High-Performance Superscalar Architectures,'' doctoral dissertation, Dept. of Computer Science and Engineering, University of Notre Dame, 2000.
5. P. Shivakumar and N. Jouppi, *An Integrated Cache Timing, Power, and Area Model*, tech. report 2001/2, Compaq Computer Corp., 2001.
6. V. Iyengar, L. Trevillyan, and P. Bose, ''Representative Traces for Processor Models with Infinite Cache,'' *Proc. Int'l Symp. High-Performance Computer Architecture* (HPCA 96), IEEE CS Press, 1996, pp. 62-73.
7. A. Phansalkar et al., ''Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites,'' *Proc. Int'l Symp. Performance Analysis of Systems and Software* (ISPASS 05), IEEE Press, 2005, pp. 10-20.
8. L. Eeckhout et al., ''Statistical Simulation: Adding Efficiency to the Computer Designer's Toolbox,'' *IEEE Micro*, vol. 23, no. 5, Sept.–Oct. 2003, pp. 26-38.
9. S. Nussbaum and J. Smith, ''Modeling Superscalar Processors via Statistical Simulation,'' *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques* (PACT 01), IEEE CS Press, 2001, pp. 15-24.
10. T. Sherwood et al., ''Automatically Characterizing Large Scale Program Behavior,'' *Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 02), ACM Press, 2002, pp. 45-57.
11. R.E. Wunderlich et al., ''SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling,'' *Proc. Int'l Symp. Computer Architecture* (ISCA, 03), IEEE CS Press, 2003, pp. 84-97.
12. B. Lee and D. Brooks, ''Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction,'' *ACM SIGARCH Computer Architecture News*, vol. 34, no. 5, Dec. 2006, pp. 185-194.
13. E. Ipek et al., ''Efficiently Exploring Architectural Design Spaces via Predictive Modeling,'' *ACM SIGARCH Computer Architecture News*, vol. 34, no. 5, Dec. 2006, pp. 195-206.
14. P. Joseph, K. Vaswani, and M.J. Thazhuthaveetil, ''A Predictive Performance Model for Superscalar Processors,'' *Proc.*

Int'l Symp. Microarchitecture (MICRO 06), IEEE CS Press, 2006, pp. 161-170.

15. F. Harrell, *Regression Modeling Strategies*, Springer, 2001.
16. C. Stone and C. Koo, ''Additive Splines in Statistics,'' *Proc. Statistical Computing Section, ASA*, American Statistics Assoc., 1985, pp. 45-48.
17. B. Lee and D. Brooks, ''Illustrative Design Space Studies with Microarchitectural Regression Models,'' *Proc. Int'l Symp. High Performance Computer Architecture* (HPCA 07), IEEE Press, 2007, pp. 340-351.
18. S. Eyerman, L. Eeckhout, and K.D. Bosschere, ''Efficient Design Space Exploration of High Performance Embedded Out-of-Order Processors,'' *Proc. Design, Automation and Test in Europe Conf.* (DATE 06), IEEE CS Press, 2006, pp. 351-356.

Benjamin C. Lee is a doctoral candidate in computer science at Harvard University. His research interests include modeling and optimization for power-efficient, high-performance computer architectures (such as heterogeneous and adaptive multicores) and applications (such as scientific computing and numerical methods). Lee has a BS from the University of California at Berkeley in electrical engineering and computer science and an MA from Harvard University in computer science.

David M. Brooks is an associate professor of computer science at Harvard University. His research interests include power- and thermal-efficient hardware-software system design and approaches to cope with reliability and variability in next-generation computing systems. Brooks has a BS from the University of Southern California and an MA and a PhD from Princeton University, all in electrical engineering.

Direct questions and comments about this article to Benjamin C. Lee, School of Engineering and Applied Sciences, Harvard University, Maxwell Dworkin 307, 33 Oxford St., Cambridge, MA 02138; bclee@seas.harvard.edu.

For further information on this or any other computing topic, visit our Digital Library at http://www.computer.org/publications/dlib.