Spatial Sampling and Regression Strategies

Benjamin C. Lee *David M. Brooks*

{bclee, dbrooks}@eecs.harvard.edu
Phone: 617-852-2210
Fax: 617-495-2489

Maxwell Dworkin 141
33 Oxford Street
Division of Engineering and Applied Sciences
Harvard University
Cambridge, Massachusetts 02138

Abstract

We present a new simulation paradigm for microarchitectural design evaluation and optimization. This paradigm counters increasing simulation costs attributed to the exponentially increasing size of design spaces and the need for more thorough, comprehensive studies when evaluating increasingly diverse design options. We present a tutorial for (1) obtaining a more comprehensive understanding of the design space by (2) selectively simulating a modest number of designs from that space and then (3) more effectively leveraging that simulation data using techniques in statistical inference. We survey techniques in spatial sampling to obtain designs for simulation. We also detail the statistical techniques required to derive efficient and robust models, interleaving code segments from scripts performing these analyses. The predictive ability and computational efficiency of these regression models enable new capabilities in microarchitectural design space studies. Collectively, our experiences with this paradigm suggest significant potential for accurate, efficient statistical inference in the microarchitectural domain.

1 Introduction

Microarchitectural design space exploration is often inefficient and ad hoc due to the significant computational costs of current simulator infrastructure. While 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 by exponentially increasing design space sizes with increases driven by the design of multi-core, multi-threaded architectures. m^p simulations might potentially be performed for a design space of p parameters each of which may take one of m values. Designers circumvent these challenges by constraining the design space using parameter subsets (p) and/or reducing the design space resolution (m). However, by applying these constraints based on intuition or experience prior to a study, the designer risks obtaining conclusions that simply reinforce prior intuition, thereby limiting the value of the study.

In conjunction with increases in design space size, designers are increasingly differentiating their market segments and targeting different metrics (*e.g.*, single-thread latency, aggregate throughput, 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, Sun UltraSPARC T1 reside in 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 such that options from very different parts of the space may be compared and evaluated.

These challenges in microarchitectural design motivate a new simulation paradigm. This paradigm enables (1) a more comprehensive understanding of the design space by (2) selectively simulating a modest number of designs from that space and then (3) more efficiently leveraging that simulation data using techniques in statistical inference. 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. Lastly, we construct regression models using these sparsely sampled simulations to enable predictions for metrics of interest. The predictive ability and computational efficiency of these models enables new capabilities in microarchitectural design optimization.

This tutorial will detail each of the three steps 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 (Section 2). We describe spatial sampling and its synergies with existing techniques for controlling simulation costs (Section 3). We propose sampling designs uniformly at random for simulation and compare this approach to several alternatives. Spline-based regression models are constructed with a number of supporting statistical analyses to produce robust, efficient models (Section 4). Each step in this derivation describes the theory, implementation, and analysis. To illustrate the derivation more concretely, we interleave code and scripts from R, an open-source software environment for statistical computing. These regression models are shown to be accurate for predicting microarchitectural design metrics and applicable to practical design optimization (Section 5).

2 Design Space

2.1 Simulation Framework

We use Turandot, a generic and parameterized, out-of-order, superscalar processor simulator [10]. Turandot is enhanced with PowerTimer to obtain power estimates based on circuit-level power analyses and resource utilization statistics [1]. The modeled baseline architecture is similar to the current POWER4/POWER5. The simulator has been validated against both a POWER4 RTL model and a hardware implementation. This simulator implements pipeline depth performance and power models based on prior work [19]. Power scales superlinearly as pipeline width increases, using scaling factors derived for an architecture with clustered functional units [18]. Cache power and latencies scale with array size according to CACTI [14]. We do not leverage any particular feature of the simulator in our models and believe our framework may be generally applied to other simulation frameworks with similar accuracy.

2.2 Benchmark Suite

We consider SPECjbb, a Java server benchmark, and eight compute intensive benchmarks from SPEC2k (ammp, applu, equake, gcc, gzip, mcf, mesa, twolf). We report experimental results based on PowerPC traces of these benchmarks. The SPEC2k traces used in this study were sampled from the full reference

	Set	Parameters	Measure	Range	$\|S_i\|$
S_1	Depth	depth	FO4	9::3::36	10
S_2	Width	width	insn b/w	4,8,16	3
		L/S reorder queue	entries	15::15::45	
		store queue	entries	14::14::42	
		functional units	count	1,2,4	
S_3	Physical Registers	general purpose (GP)	count	40::10::130	10
		floating-point (FP)	count	40::8::112	
		special purpose (SP)	count	42::6::96	
S_4	Reservation Stations	branch	entries	6::1::15	10
		fixed-point/memory	entries	10::2::28	
		floating-point	entries	5::1::14	
S_5	I-L1 Cache	i-L1 cache size	log_2(entries)	7::1::11	5
S_6	D-L1 Cache	d-L1 sache 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	FX Latency	ALU latency	cycles	1::1::5	5
		FX-multiply latency	cycles	4::1::8	
		FX-divide latency	cycles	35::5::55	
S_{10}	FP Latency	FPU latency	cycles	5::1::9	5
		FP-divide latency	cycles	25::5::45	
S_{11}	L/S Latency	Load/Store latency	cycles	3::1::7	5
S_{12}	Memory Latency	Main memory latency	cycles	70::5::115	10

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

input set to obtain 100 million instructions per benchmark program. Systematic validation was performed to compare the sampled traces against the full traces to ensure accurate representation [6]. Our benchmark suite is representative of larger suites frequently used in the microarchitectural research community [12]. Although specific conclusions of our design space studies may differ with additional benchmarks, we do not leverage any particular benchmark feature in model formulation and believe our framework may be generally applied to other workloads with similar accuracy.

2.3 Design Space

We demonstrate spatial sampling and regression modeling for the design space of Table 1. Parameters within a group are varied together to avoid fundamental design imbalances. The range of values considered for each parameter group is specified by a set of values, $S_1, \ldots, S_{12}$. The Cartesian product of these sets, $S = \prod_{i=1}^{12} S_i$, defines the entire design space. The cardinality of this product is $|S| = \prod_{i=1}^{12} |S_i| = 9.38E+08$, 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.

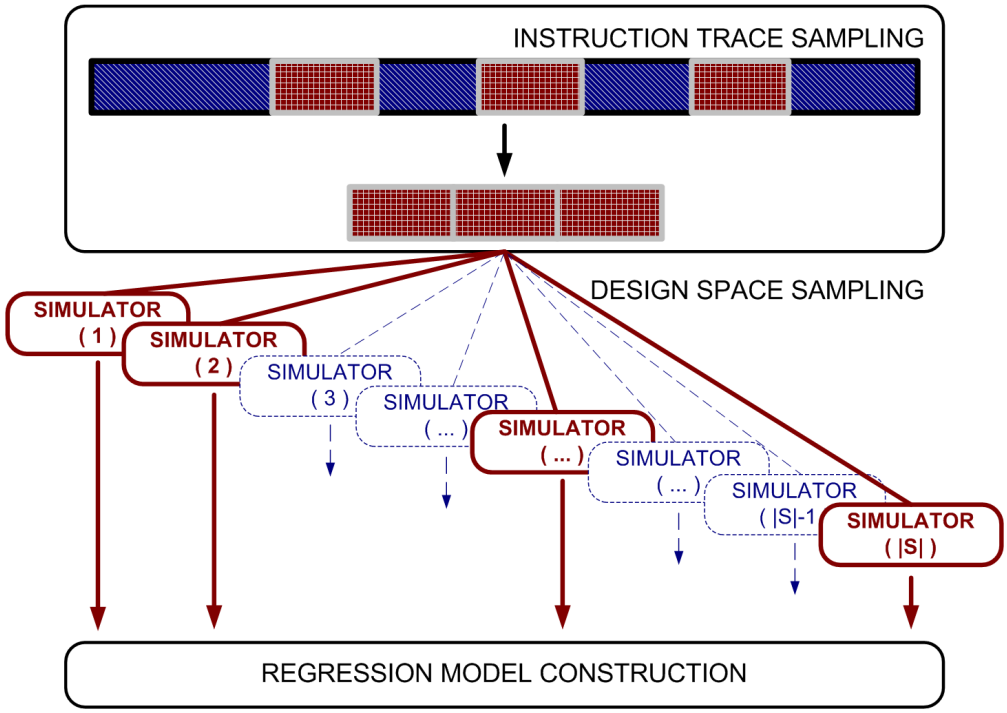

Figure 1: Simulation Paradigm. Temporal and spatial sampling reduces per simulation costs and number of required simulations.

3 Spatial Sampling

Spatial sampling, the second part of the 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. We first describe the synergies between spatial sampling and existing techniques in temporal sampling. Temporal and spatial sampling are orthogonal techniques that reduce the overall simulation costs of design space exploration by reducing costs per simulation and number of required simulations, respectively. We propose sampling points uniformly at random from the design space, outlining advantages of this approach with respect to complexity and modeling. Lastly, we survey alternative sampling strategies, highlighting their relative strengths and weaknesses.

3.1 Spatial and Temporal Synergies

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 [2]. Nussbaum, *et al.*, examine similar approaches for superscalar and symmetric multiprocessor simulation [11]. Both profile benchmarks to construct smaller, synthetic benchmarks with similar character-

istics. Sherwood and Wunderlich separately propose techniques to identify representative simulation points within an instruction trace to reduce the total number of instructions simulated [13, 17].

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 the number of potential simulations increases exponentially with the number of design parameters. This exponential increase is currently driven by the design of multi-threaded, multi-core architectures. Furthermore, as architectures differentiate and target various design metrics (*e.g.*, single-thread latency, aggregate throughput, energy efficiency), the community will observe increasing diversity in the set of viable and interesting designs (*e.g.*, simpler, less aggressive cores), thereby requiring a more thorough exploration of comprehensive design spaces with a larger number of tunable parameters.

We must therefore supplement temporal sampling with *spatial sampling*, a technique that samples points from the design space for simulation. This approach recognizes that sampling must occur in the design space to control exponentially increasing design space sizes. 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, we are able to consider many design parameters simultaneously and each parameter may assume a large number of different values.

3.2 Uniformly Random Sampling

We propose sampling design *uniformly at random* (UAR) from the design space S. This approach provides observations drawn from the full range of parameter values. An arbitrarily large number of values may be included each parameter's range since the number of simulations is decoupled from parameter resolution. Furthermore, sampling UAR does not bias simulated data toward particular designs. Uniform sampling will produce, 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 sampling UAR 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 a design point is unobserved is unrelated to the performance, power, or configuration of the design.

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 may prevent data collection for very low performance architectures and the "missingness" of a configuration is correlated with its

performance. In this case, the "missingness" is non-ignorable and we must formulate an additional model to predict whether a design point can be observed by the simulator. By sampling UAR from the design space, we ensure the observations are MCAR and avoid such modeling complications.

We will construct regression models using $n = 1,000$ design space samples. Each sampled design is simulated for every benchmark. Simulator reported performance and power numbers will provide data necessary for regression model construction. Although we use samples obtained uniformly at random from the design space, we also survey a number of alternative sampling strategies .

3.3 Alternative Sampling Strategies

Several other sampling strategies have been applied to increase the predictive accuracy of machine learning models (*e.g.*, neural networks) for the microarchitectural design space. Although we consider spline-based regression models, these strategies are broadly applicable. These techniques 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. Emphasized samples are weighted to increase their influence during model training. Weighted sampling might 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 might be used to construct localized models from samples collected uniformly at random. This approach may be necessary if regions of interest are unknown prior to sampling [8].

- **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 may improve accuracy. These 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 [3].

- **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 [7].

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. 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 guarantees sample properties that are only approximated by uniform at random 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 models constructed with these samples. We have found, however, sampling UAR provides the basis for sufficiently accurate models and comprehensive design optimization.

4 Regression Modeling

This section provides the theoretical background for relevant statistics and regression theory, illustrates the implementation of these techniques using the R statistical computing package, and interprets the resulting data and figures. 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 before and after model specification prunes unnecessary and ineffective predictors to improve model efficiency. Specifically, we consider the following derivation process for regression modeling:

- **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.
- **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.
- **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.
- **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.
- **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. Lastly, predictive ability is assessed by performance and power predictions on 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 uniformly at random 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 trainining data to improve sample resolution or extend design space boundaries (*e.g.*, large range of parameter values) 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 (*i.e.* large training sets, comprehensive design spaces).

4.1 Preliminaries

We use R, a open-source software environment for statistical computing, to script and automate statistical analyses [16]. Within this environment, we use the Hmisc and Design packages implemented by Harrell [5]. Sources, binaries, and documentation for R may be obtained via CRAN, the "Comprehensive R Archive Network", at http://www.r-project.org/. Manuals providing a broad introduction to R commands are available online at R-project website (www.r-project.org). This tutorial will discuss commands and features as they are introduced for use in regression.

4.1.1 Implementation

We begin by loading the necessary packages and libraries for regression and graphics support. The `Hmisc, Design` packages provide support for non-linear regression modeling and the `lattice` library provides support for producing many of the graphics in this tutorial. 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' headers 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`. In practice,

our data sets are assumed to contain simulator reported performance and power, design parameter values from Table 1, and application characteristics (*e.g.*, cache miss rates).

```
## Load Libraries and Data
    library(Hmisc, T); library(Design, T); library(lattice);
    data_model.df <- read.table("data_model.txt", sep="\t", header=TRUE);
    data_valid.df <- read.table("data_valid.txt", sep="\t", header=TRUE);

## Data Description
    describe(data_model.df);
    dd <- datadist(data_model.df); ## uses information about data set distribution
    options(datadist='dd');        ## to set options for other Design, Hmisc functions
```

4.1.2 Analysis

Invoking the `describe` function on a data frame provides summary statistics of its variables. The following excerpt of ouput from `derive` summarizes the performance data, `bips` and indicates 2000 observations, none missing and 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 0.3572. In this case, the mean is close to the median suggesting a symmetric distribution. Lastly, five outliers at both extremes are listed (`lowest, highest`).

```
bips
      n missing  unique    Mean     .05     .10     .25     .50     .75     .90     .95
   2000       0    2000  0.7641  0.2610  0.3572  0.4966  0.7122  0.9771  1.2595  1.4178

lowest : 0.1140 0.1192 0.1198 0.1225 0.1281
highest: 2.1061 2.1349 2.2984 2.4647 2.8090
```

4.2 Hierarchical Clustering

4.2.1 Theory

Data clustering classifies N data elements into clusters based on 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 based on 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 one N-element cluster is obtained

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 modeler 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 since correlated responses will likely scale with the modeled response.

Pruning the number of 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 as they reduce the number of sampled observations required for model formulation. A number of studies in which models are validated on independent data sets have shown a fitted regression model is likely reliable (no over-fitting) when the number of samples is twenty times the number of model terms [5].

```
## 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();
```

4.2.2 Implementation

We perform clustering with `varclus` on the potential predictors and responses, specifying variables and the data frame where the variables are defined. The $\sim$ operator specifies a relationship between $x \sim y + z$ where x is the response and y, z are the predictors. For `varclus`, a sum of terms on the right hand side of the relationship 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. We relate performance to these parameters and characteristics using the ~ relationship operator. Lastly, we specify the data frame in which all of these variables are

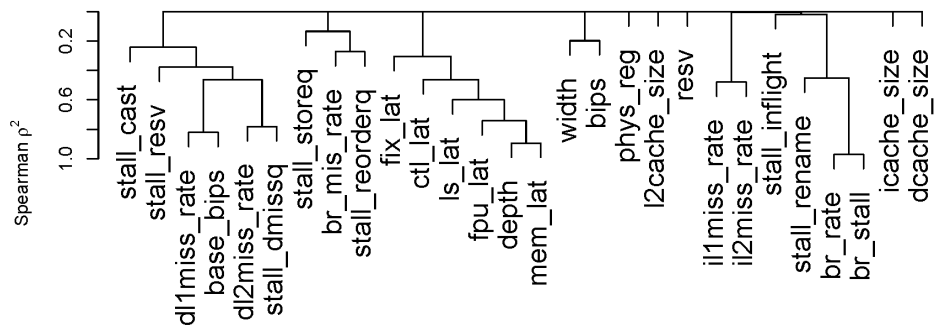

Figure 2: Hierarchical Clustering. The level at which clusters connect indicates their degree of similarity. Spearman ρ^2 is a rank-based measure of correlation.

defined (`data=data_model.df`). Alternatively, we could have specified each variable with the `$` operator (*e.g.*, `data_model.df$depth`).

Storing the clustering results to `v` allows us to print the numerical results of the similarity matrix. The `print(v)` command will produce a $n \times n$ matrix where n is the number of predictors and each entry contains pairwise correlation coefficients between variables. The correlations between variables are more easily observed in a clustering figure. Such a figure is generated by creating a trellis device with the `trellis.device` function, specifying the figure format, file name, and figure dimensions. Close the device with the `dev.off` function.

4.2.3 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 meet indicates their degree of similarity. L1 and L2 misses due to instruction cache accesses are highly correlated. We find the absolute number of L2 cache misses from the instruction probes to be negligible and 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 since we scale the original functional unit latencies with depth. Since final latencies are a function of original latencies and pipeline depth, we choose to keep these original latency variables. Including both predictors enables us to differentiate the performance impact of individual functional unit latency changes from the global latency changes as depth varies. However, if these effects are later determined to be insignificant, we should remove these extra latency parameters to improve model efficiency. Similarly, we keep both d-L1 cache miss rate and the baseline performance predictors to differentiate cache performance from global performance.

4.3 Association Analysis

4.3.1 Theory

Scatterplots qualitatively represent the association between variables. Such plots are useful for examining association between predictors and the response, revealing potential non-monotonicity or non-linearity. Scatterplots could quickly identify more significant predictors by showing, for example, a clear monotonic relationship with the response. Conversely, plots that exhibit low response variation despite a changing predictor value might suggest predictor insignificance. Overall, scatterplots allow the modeler to understand the parameter space quickly at a high level.

4.3.2 Implementation

The `summary` function takes a relationship specified by the $\sim$ operator. For example, $bips \sim depth + width + phys\_reg + resv$ 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.

```
## 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();
```

4.3.3 Analysis

Invoking the `print` function produces the data table of Figure 3L relating predictor intervals to the response. For example, pipeline depth takes a value between 9 and 15 FO4 delays per stage for 1213 of 4000 samples. The average performance for these designs is 0.865 billion instructions per second. Figure 3R visualizes this data with scatterplots, illustrating strong monotonic relationships between performance and pipeline dimensions. The register file size appears to have a significant, but non-linear, relationship with performance. In contrast, the number of entries in reservation stations seems to have a negligible performance impact.

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	0.8316163
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

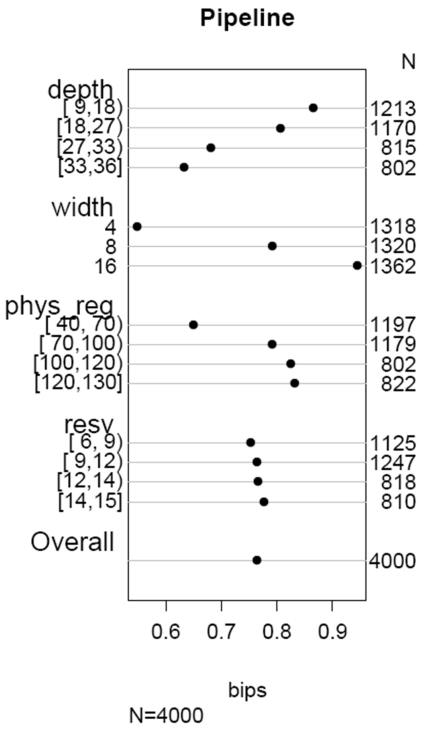

Figure 3: Association Analysis. Table summarizes ranges of parameter values (second column), number of samples in each range (third column), and average performance of these samples (fourth column). Scatterplot visualizes this data.

4.4 Correlation Analysis

4.4.1 Theory

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 by Equation (1) where X,Y are random variables with expectations μ_x,μ_y and standard deviations σ_x,σ_y.

$$\rho = \frac{E((X-\mu_X)(Y-\mu_Y))}{\sigma_X \sigma_Y} \tag{1}$$

In cases where the distribution of X, Y are unknown, non-parametric statistics may be more robust. In particular, we use the Spearman rank correlation coefficient ρ_{sp} that can quantify association independently of variable distribution. The computationally efficient approximation only requires 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 x_i's rank in X, y_i's rank in Y. Spearman rank correlation computes a coefficient using the N differences in rank.

$$\rho_{sp} = \frac{\sum_{i=1}^{N} X_i Y_i}{\sqrt{\sum_{i=1}^{N} X_i^2 \sum_{j=1}^{N} Y_j^2}} \approx 1 - 6 \sum_{i=1}^{N} \frac{d_i^2}{N(N^2-1)} \tag{2}$$

4.4.2 Implementation

Given a predictor-response relationship, the `spearman2` function computes the squared Spearman rank correlation coefficient. Specifically, we compute correlation of performance (left hand side) against each predictor (right hand side) using the `~` relationship operator. For example, `spearman2(bips~depth, data=data_model.df)` computes the Spearman rank correlation coefficient 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 + base_bips),
                    data = data_model.df);
    print(sp);
    trellis.device("pdf", file="./spearman_plot.pdf");
    plot(sp);
    dev.off();
```

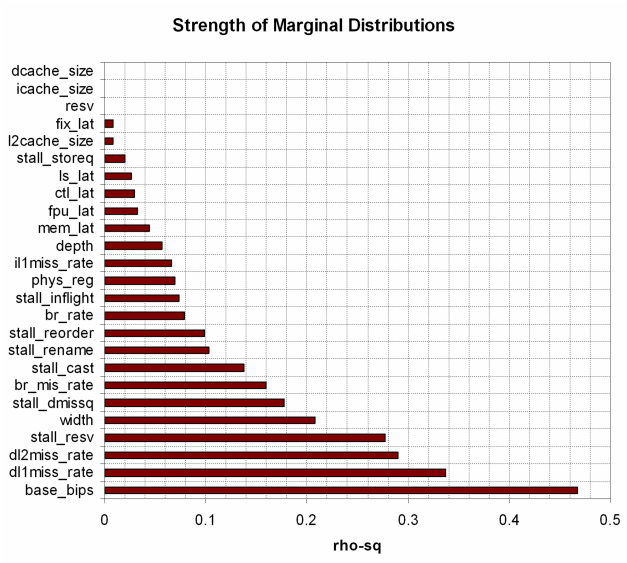

Figure 4: Correlation Analysis. Plots squared spearman rank correlation coefficients for each parameter, characteristic.

4.4.3 Analysis

Figure 4 plots the non-monotonic generalization of the Spearman rank correlation coefficient for each predictor variable. This information will guide our choice in 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 are more effective since they consider only the microarchitectural impact on performance.

4.5 Model Specification

We apply regression modeling techniques to obtain empirical estimates for metrics of interest. 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 potentially non-linear relationships.

4.5.1 Theory

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 $x_i = x_{i,1}, \ldots, x_{i,p}$ 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 Equation (3). Mathematically, β_j may be interpreted as the expected change in y_i per unit change in the predictor variable $x_{i,j}$. The e_i are assumed independent random variables with zero mean and constant variance; $E(e_i) = 0$ and $Var(e_i) = \sigma^2$.

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

Transformations f and $g = g_1, \ldots, g_p$ 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 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 of the predicted responses given by the model from the actual observed responses. Thus, least squares finds 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, $\hat{\beta}_j$, are estimates of the coefficients in Equation (3).

$$S(\beta_0, \ldots, \beta_p) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \tag{4}$$

Interaction: In some cases, the effect of two predictors $x_{i,1}$ and $x_{i,2}$ on the response cannot be separated; the effect of $x_{i,1}$ on y_i depends on the value of $x_{i,2}$ and vice versa. The interaction between two predictors may be modeled by constructing a third predictor $x_{i,3} = x_{i,1}x_{i,2}$ to obtain $y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,1} x_{i,2} + e_i$. Modeling predictor interactions in this manner makes it difficult to interpret β_1 and β_2 in isolation. After simple algebraic manipulation to account, we find $\beta_1 + \beta_3 x_{i,2}$ is the expected change in y_i per unit change in $x_{i,1}$ for a fixed $x_{i,2}$.

Non-Linearity: Basic linear regression models often assume the response behaves linearly in all predictors. This assumption is often too restrictive 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.

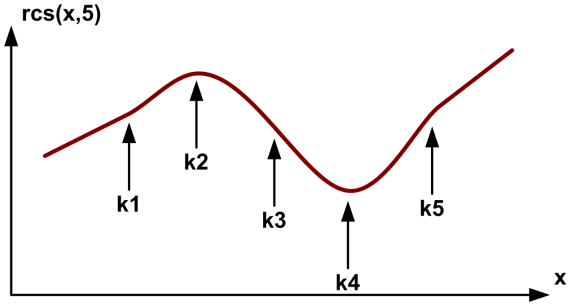

Figure 5: Restricted Cubic Spline. Domain of x divided using five knots. Polynomial interior, linear tails.

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. 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. A restricted cubic spline on x with k knots $t_1, \ldots, t_k$ is given by Equation (5) where $j = 1, \ldots, k-2$ [15].

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_{k-1} x_{k-1} \tag{5}$$

$$x_1 = x \tag{6}$$

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

We use restricted cubic splines since linear splines may be inadequate for complex, highly curved relationships. Splines of higher order polynomials may offer better fits [5]. 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. Cubic splines may have poor behavior in the tails before the first knot and after the last knot [15]. Restricted cubic splines that constrain the function to be linear in the tails are often better behaved.

Figure 5 illustrates schematically a restricted cubic spline with five knots and linear tails. The choice and position of knots are variable parameters when specifying non-linearity 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 [15]. In practice, five knots or fewer are generally sufficient for restricted cubic splines. Fewer knots may be required for small data sets. 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.

4.5.2 Implementation

We specify a regression model, predicting performance from various microarchitectural design parameters. Restricted cubic splines are specified by the `rcs` function that takes the predictor and 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 for restricted interactions between cubic splines by removing doubly non-linear 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 stalls affect, in turn, instruction throughput gains from pipelining. Thus, their joint impact on performance 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 (*e.g.* 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 (demonstrated in the next section) suggests this high-level representation is sufficient.

```
## 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) ~ .);
```

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 based on 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 ".". 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 vary (*e.g.*, depth). Both `sqrt` and `log` transformatins are standard in the statistics literature and are typically applied to more tractably and effectively analyze data with large values in absolute terms.

4.6 Assessing Fit

We define and assess model fit using three approaches. We first quantify the degree to which our model captures the variability in the underlying training data. Ideally, the model and simulator predict metrics of interest with the same variance. We then check the model for systematic bias to ensure robustness. Ideally, model error is random and independent of the predicted design. For example, a biased and less robust model might consistently over-predict performance for particular pipeline depths. Lastly, we assess model accuracy by comparing model predictions against detailed simulation using an indepedent validation data set. Collectively, these analyses give users confidence in the robustness and accuracy of the model.

4.6.1 Theory

The model's fit to the sampled simulation data used in formulation is quantified with the *multiple correlation statistic*, R^2, in Equation (10). This statistic quantifies regression error (SSE) as a fraction of total error (SST). From the equation, R^2 will be zero when 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 one may indicate over-fitting, a situation in which the model's worth is exaggerated and future observations may not agree with the modeled predictions predicted values. Over-fitting typically occurs when too many predictors are used to estimate relatively small data sets.

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

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

$$R^2 = 1 - \frac{SSE}{SST} \tag{10}$$

The model should also be examined to ensure predictions exhibit no systematic bias based on an analysis of residuals in Equation (11). These residuals are per sample differences between predicted and observed performance in the training set illustrated in Figure 6. In particular, we should validate the following assumptions to ensure model robustness:

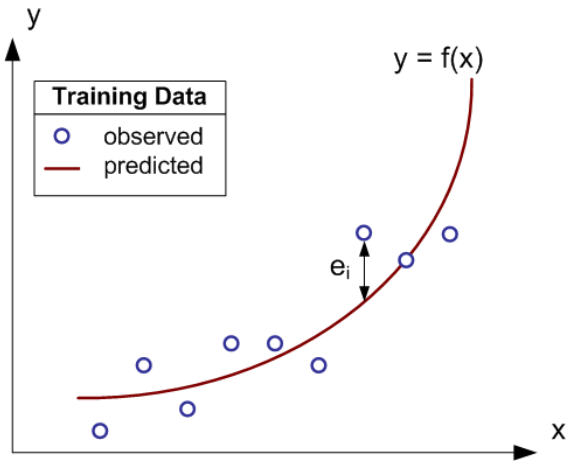

Figure 6: Residual Schematic. Differences between observed and predicted values in training data.

1. the residuals are not correlated with the predicted response
2. the residuals' randomness should be 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 since such plots may reveal systematic deviations from randomness. The third assumption is usually 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.

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

Lastly, we obtain 100 additional randomly selected points from the design space and compare simulator-reported metrics against regression-predicted metrics. The error distribution for these validation points may be visualized using boxplots. 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

1. horizontal lines at median and upper, lower quartiles
2. vertical lines drawn up/down from upper/lower quartile to most extreme data point within 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 vertical lines

3. circles for each outlier

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

4.6.2 Implementation

The R^2 statistic is included in the model summary obtained by applying the `print` function to the formulated models (`f` and `g`). Residuals are plotted against model predictions for the training set. This data is obtained from a fitted model by the `resid` and `fitted` functions, respectively. Residuals are plotted against the fitted values (*i.e.*, 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 twenty elements (`nx=20`). This function plots the median, lower and upper quantiles of the residuals for these groups of twenty observations. The grouping is necessary if there are too many observations for a standard scatterplot.

We validate assumption 3 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 performs the ranking and generation of normal samples to produce a plot that should appear linear if the residuals follow a normal distribution. The `qqline` function draws a line through the 25-th and 75-th percentile of the residuals to aid this analysis.

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

## Residual Analysis: (1) scatterplot and (2) quantile-quantile plot
    trellis.device("pdf", file="./residf.pdf", width=6, height=4);
    xYplot(resid(f) ~ fitted(f), method='quantile', nx=20,
           xlab=``Fittted Values'', ylab=''Residuals'');
    dev.off();

    trellis.device("pdf", file="./qqnormf.pdf", width=6, height=4);
    qqnorm(resid(f));
    qqline(resid(f));
    dev.off();
```

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. Note that 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 tab-delimited file with appropriate column names. The error distribution is visualized by boxplots constructed

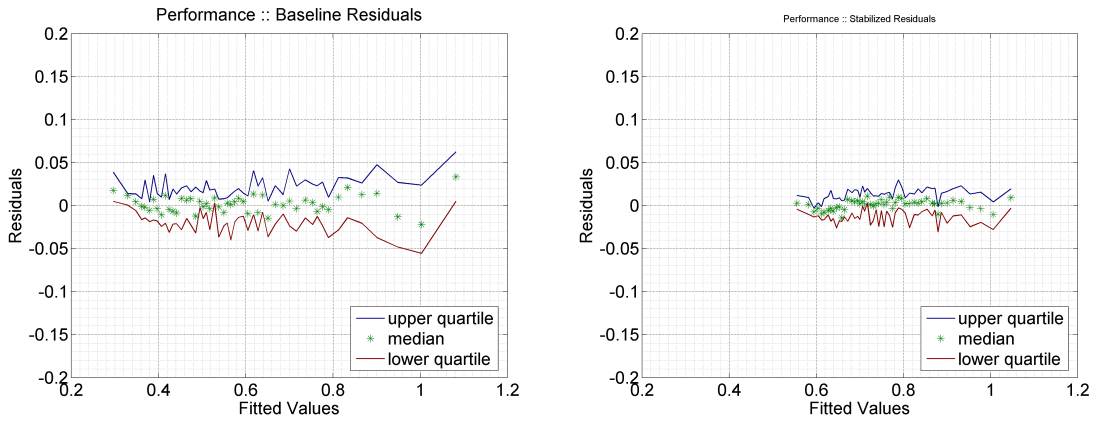

Figure 7: Residual Analysis. Scatterplots of residuals before and after square root transformation. Residuals of an unbiased model should appear randomly and independently distributed around zero.

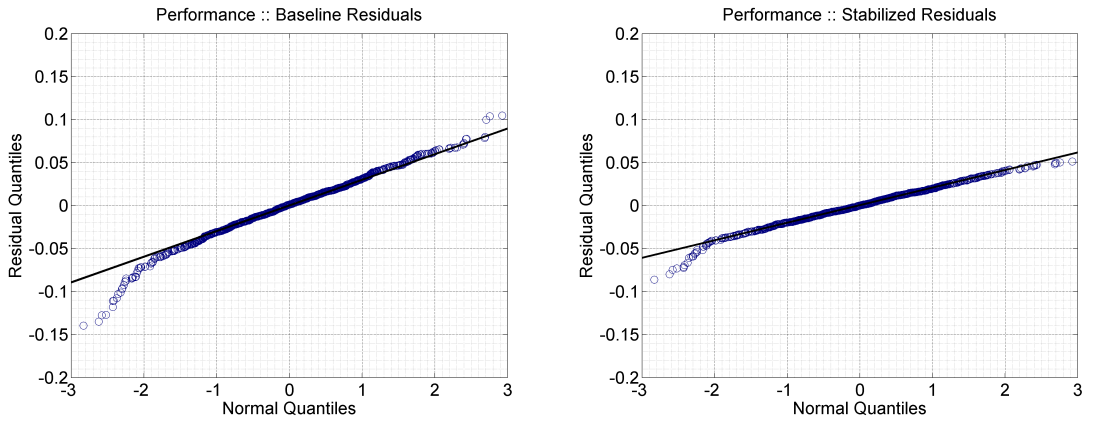

Figure 8: Residual Analysis. Quantile-Quantile plots of residuals before and after square root transformation. Plots for an unbiased model used appear linear.

with the `bwplot` function.

```
## Model Prediction
    o <- data_valid.df\$bips;
    p <- (predict(object=f, newdata=data_valid.df))^2;
    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();
```

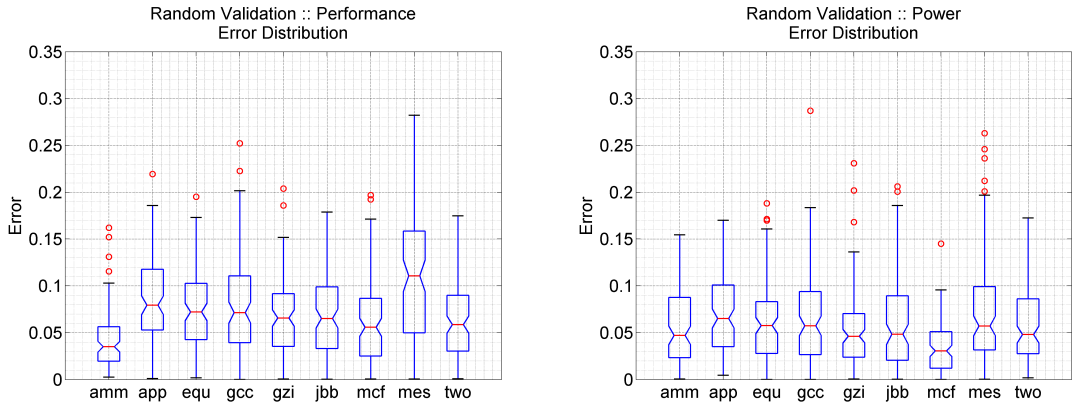

Figure 9: Error Distributions. Boxplots capture quartiles of model errors when predicting performance and power of validation set.

4.6.3 Analysis

The 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. Over-fitting should not be a concern since the number of model terms is much smaller than the number of samples. Figure 7L suggests a correlation between residuals and fitted values for a non-transformed performance metric. Residuals are biased positive for the smallest and largest fitted values. Figure 7R indicates the standard variance stabilizing square root transformation on performance reduces the magnitude of these correlations. This transformation also causes residuals to follow a normal distribution more closely as indicated by the linear trend in Figure 8.

Figure 9 indicates the performance model achieves median errors ranging from 3.5 percent (ammp) to 11.1 percent (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.

5 Design Optimization

The proposed simulation paradigm effectively combines spatial sampling and 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 while 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.

- **Design Space Characterization:** The computational efficiency of regression models enable the complete performance and power characterization of design spaces with hundreds of thousands of points. Exhaustively evaluating the models for every point in the space, we are able to identify quickly design bottlenecks and trends as design parameters vary [9].

- **Pareto Frontier Analysis:** The complete design space characterization enables the construction of a pareto frontier. This frontier is comprised of designs that maximize performance for a given power budget or minimize power for a given performance target [9].

- **Parameter Sensitivity Analysis:** Regression models enables us to consider all parameters simultaneously when performing parameter sensitivity analyses. For example, most prior pipeline depth studies held non-depth parameters constant and considered the effects of varying depth around a particular baseline design. In contrast, regression models enable us to 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 [9].

- **Heterogeneous Multicore Design:** The models enable comprehensive optimization for identifying effective core design compromises given a workload set. The models are used to identify per workload optima (*i.e.* architectures maximizing a metric of interest). A clustering analysis on these optima produce design compromises for workloads requiring similar resources at the microarchitectural level. These compromises may form the basis for core design in heterogeneous multiprocessor architectures targeting these workloads [9].

- **Topology Visualization:** Microarchitectural designs occupy high-dimensional spaces. Regression models enable the visualization of performance and power topologies using two-dimensional projections of the design space. Practically, these projections produce two-dimensional contour maps for all pairs of design parameters in the space, illustrating non-linearities or non-monotonicities. These visualizations aid bottleneck analysis while motivating the need for non-linear models.

- **Topology Roughness Metrics:** Given visualizations of design topologies, we may assess relative topology roughness subjectively by comparing contour maps. Roughness metrics supplement these contour maps, quantifying the range and variability of these contours. Furthermore, these metrics extend to higher dimensions and may be applied beyond two-dimensional contours (*e.g.*, Equation (13) for d-dimensional roughness). In the microarchitectural context, $f(x)$ is a performance or power function and $x_1, \ldots, x_d$ are design parameters. The function $f(x)$ is approximated by regression models while the derivatives and integrals may be approximated by differences and sums.

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

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

- **Topology Optimization:** Optimization by 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 [4], using regression models for these comparisons significantly increases the heuristic's computational efficiency.

Thus, regression models enable new capabilities in design space characterization and optimization. These capabilities produce more comprehensive variants of traditional studies or enable new studies not possible with current usage patterns for microarchitectural simulators. Collectively, these capabilities motivate the new simulation paradigm based on spatial sampling and regression modeling to more effectively utilize simulator cycles.

6 Conclusion

This tutorial demonstrates the successful application of a new simulation paradigm for microarchitectural design evaluation, combining spatial sampling from comprehensive design spaces and statistical inference. Specifically, this paradigm enables (1) a more comprehensive understanding of the design space by (2) selectively simulating a modest number of designs from that space, and (3) more efficiently leveraging that simulation data using techniques in statistical inference. We propose sampling designs uniformly at random from the design space. We also review statistical analyses required for constructing robust and efficient spline-based regression models.

Detailed simulation will continue to play a significant role in microarchitectural design. These simulators enable early stage 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 current simulator usage patterns are highly inefficient. Spatial sampling and statistical inference reduces 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.

Acknowledgements

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

References

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

[2] L. Eeckhout, S. Nussbaum, J. Smith, and K. DeBosschere. Statistical simulation: Adding efficiency to the computer designer's toolbox. *IEEE Micro*, Sept/Oct 2003.

[3] E.Ipek, S.A.McKee, B. de Supinski, M. Schulz, and R. Caruana. Efficiently exploring architectural design spaces via predictive modeling. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2006.

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

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

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

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

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

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

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

[11] S. Nussbaum and J. Smith. Modeling superscalar processors via statistical simulation. In *International Conference on Parallel Architectures and Compilation Techniques*, Sept 2001.

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

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

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

[15] C. Stone and C. Koo. Additive splines in statistics. In *Statistical Computing Section ASA*, 1985.

[16] R. D. Team. *R Language Definition*.

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

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

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