Statistical Inference for Efficient Microarchitectural Analysis

A dissertation presented
by

Benjamin Chi-Chung Lee

to
The School of Engineering and Applied Sciences
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of

Computer Science

Harvard University
Cambridge, Massachusetts
May 2008

Thesis advisor

David M. Brooks

Author

Benjamin Chi-Chung Lee

Statistical Inference for Efficient Microarchitectural Analysis

Abstract

The transition to multiprocessors expands the space of viable core designs and requires sophisticated optimization over multiple design metrics. However, microarchitectural design space exploration is often inefficient and ad hoc due to the significant computational costs of hardware simulators. Long simulation times cause designers to subjectively constrain the design space considered. However, by pruning the design space with intuition before a study, the designer risks obtaining conclusions that simply reinforce prior intuition, thereby limiting the study's value. Addressing these fundamental challenges in microarchitectural analysis becomes increasingly urgent as the semiconductor industry moves into new domains where tried and tested intuition becomes less effective.

This dissertation presents the case for statistical inference in microarchitectural design, proposing a simulation paradigm that (1) defines a comprehensive design space, (2) simulates sparse samples from that space, and (3) derives inferential regression models to reveal salient trends. These regression models accurately capture performance and power associations for comprehensive multi-billion point design spaces. Moreover, they are capable of thousand's of predictions per second.

Used as computationally efficient surrogates for detailed simulation, regression models enable previously intractable analyses of performance and power. Leverag-

ing model efficiency, this dissertation demonstrates qualitatively new capabilities by using pareto frontiers to identify power-efficient designs, contour maps to visualize bottlenecks, and roughness metrics to quantify non-monotonicity in design topologies.

Furthermore, inferential models enable qualitatively new capabilities in optimization for emerging design priorities. Not only do these models answer prior questions far more quickly, they answer new questions previously intractable with detailed simulation. This dissertation implements robust optimization techniques to assess multiprocessor heterogeneity and microarchitectural adaptivity, quantifying trends and limits in performance and power efficiency from these design paradigms. The capabilities from inference scale to multi-billion point design spaces, giving designers the holistic view necessary to successfully implement the transition to multiprocessors.

Contents

List of Figures

List of Tables

Citations to Previously Published Work

Portions of this dissertation have appeared in the following conference proceedings:

Benjamin C. Lee, David M. Brooks. "Efficiency trends and limits from comprehensive microarchitectural adaptivity." ASPLOS-XIII: International Conference on Architectural Support for Programming Languages and Operating Systems. Seattle, WA, March 2008.

Benjamin C. Lee, David M. Brooks. "Roughness of microarchitectural design topologies and its implications for optimization." HPCA-14: International Symposium on High-Performance Computer Architecture. Salt Lake City, UT, February 2008.

Benjamin C. Lee, David M. Brooks. "Illustrative design space studies with microarchitectural regression models." HPCA-13: International Symposium on High-Performance Computer Architecture. Phoenix, AZ, February 2007.

Benjamin C. Lee, David M. Brooks. "Accurate and efficient regression modeling for microarchitectural performance and power prediction." ASPLOS-XII: International Conference on Architectural Support for Programming Languages and Operating Systems. San Jose, CA, October 2006.

Acknowledgments

Many people have contributed to my professional and personal growth. I thank David Brooks for his invaluable guidance and support as my doctoral advisor. I appreciate his experience and ability to evaluate all contingencies. Most importantly, I learned from David the importance of balancing methodology and application.

Also at Harvard University, I thank Mike Smith, Gu-Yeon Wei, and Patrick Wolfe for serving on my qualifying committee. Their early feedback on my work established a strong foundation for the rest of my doctoral research. I also thank Greg Morrisett and Margo Seltzer for serving on my dissertation committee. Their unique perspectives on my work, in particular, and computing research, in general, enriched both my dissertation and my vision for future research.

For first introducing me to research, I thank Jim Demmel and Kathy Yelick at the University of California, Berkeley. They helped me build a foundation in rigorous performance analysis and solid research as an undergraduate in the Berkeley Benchmark and OPtimization (BeBOP) group. I also thank Rich Vuduc, then a graduate student at Berkeley, for his patience and guidance early in my career. Jim, Kathy, and Rich prepared me well for the rigors of graduate school and I am in their debt.

For enhancing my graduate studies with practical experience, I thank Bronis de Supinski and Martin Schulz for hosting my internship at Lawrence Livermore National Laboratory. Similarly, I thank Hong Wang and Jamison Collins for hosting my internship at Intel Corporation. These experiences broadened my view of the computing landscape. I learned more about the realities of computing in a few months than I could have ever feasibly learned in my years of graduate study.

For enriching my graduate school experience, I thank my colleagues at Harvard.

I especially thank Glenn Holloway for his invaluable technical support and, perhaps more importantly, the occasional Maxwell Dworkin kitchen conversation. I also thank Kevin Brownell, Meeta Gupta, Alex (Xiaoyao) Liang, Mike Lyons, Krishna Rangan, and VJ Reddi for their camaraderie, joining me to convert what was originally a machine room into a vibrant lab. I also thank members of the VLSI group, Ankur Agrawal, Hayun Chung, Mark Hempstead, Wonyoung Kim, Andrew Liu, and Ruwan Ratnayake for their fellowship and discussion of seemingly endless tape-outs.

Most importantly, I thank my family for their love and support. My father always has ready words of inspiration through which he expresses pride in my accomplishments. My mother has an infinite capacity for learning about the details of my life as only a mother can. My brother, of whom I am immensely proud, has been and will be an invaluable sounding board as I enter the next stage of my career and life.

Dedicated to my parents, Martin and Peggy,
and my brother, Franklin.

Chapter 1

Introduction

Contents

Technology trends drive increasing microarchitectural design and metric diversity, which increases the difficulty and cost of effective design analysis. Mitigating these costs is increasingly urgent as the computer industry moves into previously unexplored domains where designer intuition is less effective. This dissertation proposes a new paradigm for efficient design space exploration, reducing design evaluation costs by several orders of magnitude while drastically increasing the information content from microarchitectural simulation. Relying on statistical inference, the paradigm enables qualitatively new capabilities in design analysis and optimization.

1.1 Technology Trends

Moore's Law provides increasing transistor densities and, consequently, more abundant resources to microarchitectural designers [55].[1] With this abundance, however, comes responsibility. Designers must deliver performance in a cost-effective manner, whether by controlling power dissipation, area costs, or design complexity. Power, in particular, is a first-order design constraint, necessary for controlling thermal effects in high-performance systems or extending battery life in embedded architectures.

Figure 1.1 schematically illustrates the microprocessor core and the large number of tunable design parameters that impact performance and power. Instructions are fetched from the instruction cache, buffered, and decoded to determine the operations they implement. Instruction operands are dynamically renamed to mitigate write hazards. Dispatched instructions are queued and sent to various functional units. The number of functional units (*i.e.*, superscalar width) and the number of logic stages in the datapath (*i.e.*, pipeline depth) impact performance by determining the number of instructions flowing through the datapath. Power increases with width and depth as additional structures are added and the core's operating frequency increases, respectively. Sizes for various supporting queues, tables, and arrays also impact performance and power. Lastly, the datapath is supported by a cache hierarchy where size and associativity impact performance and power.

Navigating the inherent tensions between additional resources and their associated power costs, designers turn to chip multiprocessors. Comprised of multiple mi-

[1] Moore's Law describes technology advances that double transistor density on integrated circuits every twelve to eighteen months.

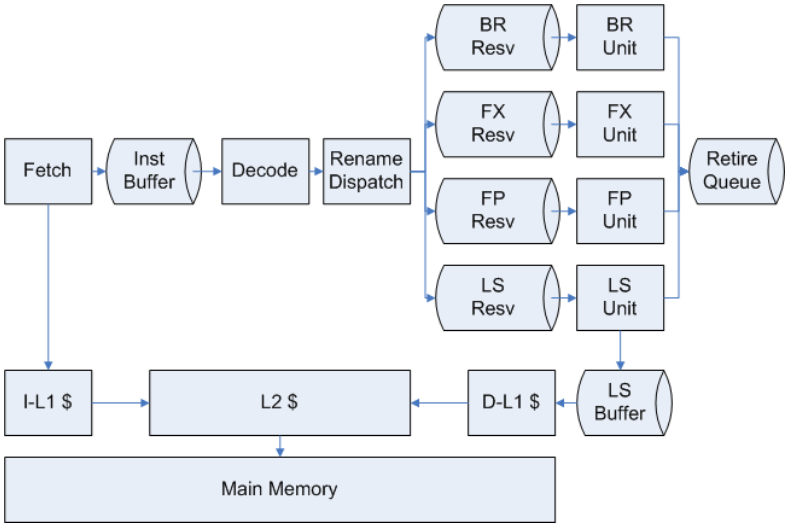

Figure 1.1: **Microprocessor Core**. A datapath fetches and executes instructions. The datapath is supported by a memory hierarchy with multiple levels of cache.

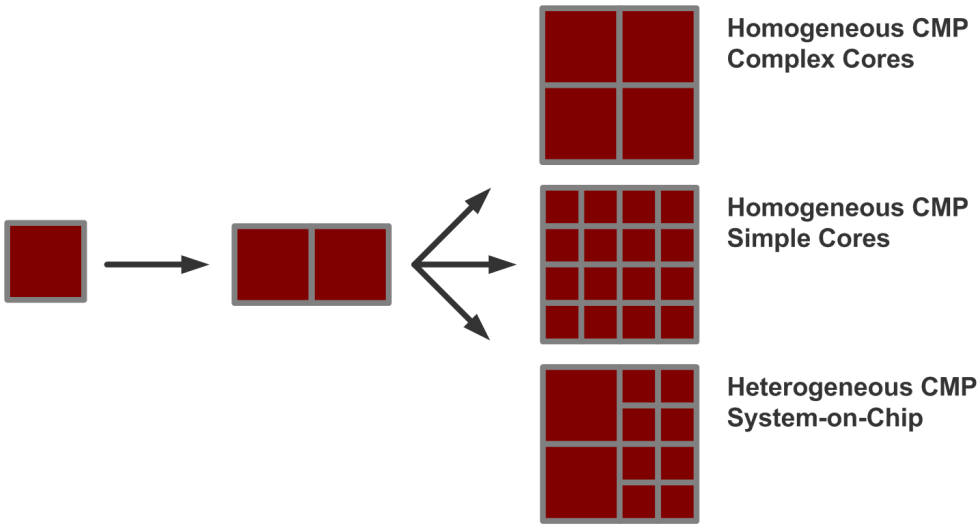

Figure 1.2: **Transition to Chip Multiprocessors**. Previously, designers considered uniprocessors. At present, designers consider multi-core architectures. In the future, designers choose among several diverse trajectories: homogeneous multi-core with cores of varying size and complexity or heterogeneous multi-core/system-on-chip architectures.

croprocessor cores on a single chip, multiprocessors deliver performance with greater power efficiency than uniprocessors. As designers move into the multiprocessor domain, however, we observe a number of possible trajectories as shown in Figure 1.2. Homogeneous multiprocessors with a relatively small number of large, complex cores naturally extend past and present design paradigms. For example, the IBM POWER5 is a dual-core chip-multiprocessor with relatively wide datapaths and abundant per-core resources [64]. Also likely, however, are multiprocessors with a large number of small, simple cores that deliver high aggregate throughput at the expense of per-core latency. For example, the Sun UltraSPARC T2 implements relatively small in-order cores emphasizing throughput across a large number of threads [50]. Finally, heterogeneous multiprocessors may enhance performance and power efficiency with some combination of large cores for general-purpose computing and small cores for special-purpose acceleration. For example, the IBM Cell Broadband Engine combines a general-purpose PowerPC architecture accelerated by smaller synergistic processing elements for single-instruction multiple data (SIMD) acceleration [33]. Thus, the transition to multiprocessors drives increasing *design diversity* as the industry considers microprocessor cores for a broad range of emerging microarchitectural design priorities.

Simultaneously, microprocessor core optimization is increasingly complex. As the industry identifies market segments and differentiates their hardware offerings for these segments, designers must optimize for an array of metrics such as single-threaded latency, aggregate throughput, power, temperature, and area. Although all of these metrics are important, different market segments assign them differ-

ent relative priorities. For example, high-performance computing might emphasize single-threaded latency whereas commercial transaction processing might emphasize aggregate throughput. Thus, the transition to multiprocessors must also account for increasing *metric diversity* as the industry considers the different design trajectories for different market segments.

Increasing design and metric diversity requires more robust analysis and optimization. Designs considered infeasible in the uniprocessor space become viable in the multiprocessor domain. Scalable and comprehensive design space exploration is required to assess the relative merits of many disparate designs occupying very different parts of the space. This scalability must also extend to optimization across diverse metrics. Exhaustive search to identify optima, while tractable for small design spaces, inherently lacks scalability and more sophisticated optimization heuristics are needed. Microarchitectural design space exploration has always been an expensive combinatorial optimization problem. Collectively, however, technology trends increase the difficulty of analysis and optimization, exposing fundamental weaknesses in modern microarchitectural simulators.

1.2 Simulation Challenges

Simulation is the tool of choice in microarchitectural analysis as both academic and industrial researchers leverage its software extensibility to quickly propose and evaluate new design features. Cycle-accurate microarchitectural simulators provide detailed insight into application behavior for a broad range of microprocessor core configurations. These simulators track instructions as they progress through a simulated

microarchitectural pipeline to record resource utilization statistics. Given these statistics, simulators estimate metrics such as performance and power.

However, detailed simulation is expensive and its cost consists of two components: cost per simulation and the number of required simulations. Cost per simulation is often tractable when analyzing a particular design; simulating a modest trace of 100 million instructions requires tens of minutes. However, cost does not scale with the number of simulations required for design space exploration. The design space size and, consequently, the number of potential simulations scale exponentially with the number of design parameters. Thus, simulation costs quickly become intractable for any comprehensive analysis of a broad and diverse design space.

Given the costs of simulation, microarchitectural design space exploration is often inefficient and ad hoc. Designers circumvent these simulation challenges by constraining the design space using intuition or experience. However, by pruning the design space with intuition prior to a study, designers risk obtaining conclusions that simply reinforce prior intuition and may not generalize to the broader space. Such an approach to analysis will become increasingly ineffective as the industry moves into new domains, characterized by significant design and metric diversity, where designer intuition is less mature.

1.3 Simulation Paradigm

To address these fundamental simulation challenges, this dissertation proposes a simulation paradigm with three components: comprehensive design spaces, spatial sampling, and statistical inference (Chapter 2).

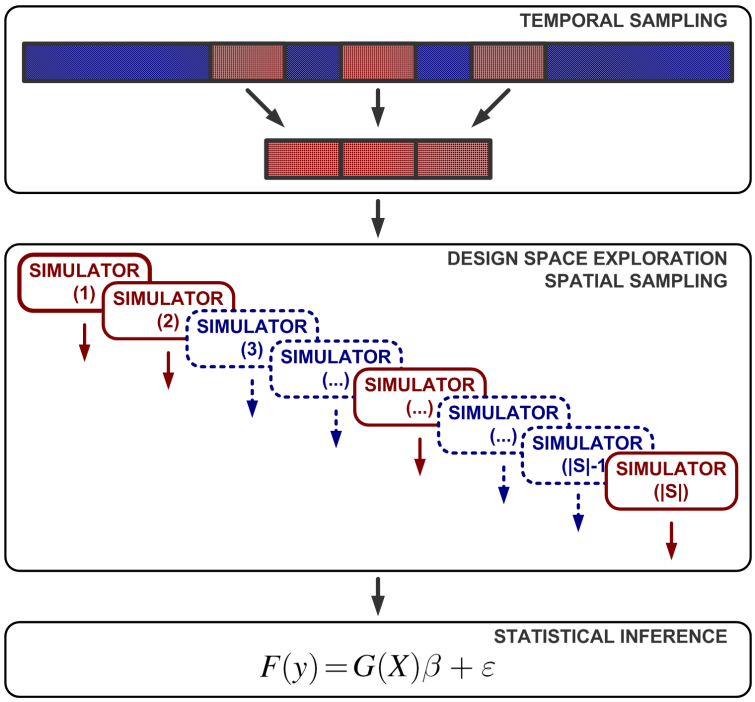

Figure 1.3: **Simulation Paradigm**. Temporal and spatial sampling reduces per simulation costs and number of required simulations, respectively. Statistical inference reveals broader trends from spatial samples.

1. **Comprehensive Design Spaces:** Designers specify a large, high-resolution space that considers many design parameter simultaneously.

2. **Spatial Sampling:** Designers sparsely sample and simulate designs from the space, thereby decoupling design space size from the number of simulations.

3. **Statistical Inference:** Designers efficiently leverage simulation data by constructing inferential models from sparse samples to predict metrics of interest.

Each component of this paradigm targets fundamental limitations in current simulation methodology as illustrated in Figure 1.3.

A comprehensive design space addresses increasing design diversity. The shift to multiprocessors makes a larger fraction of the design space viable for implementation. To most effectively identify the optimal core size and complexity for future multiprocessors, designers must take a holistic view of diverse design options. Unlike studies constrained using intuition, a comprehensive design analysis makes possible the discovery of unexpected trends or optima. Furthermore, by considering many parameters simultaneously, designers expose interactions between parameters and ensure bottlenecks are truly removed, not simply shifted from one parameter to another parameter outside a limited study's scope.

Spatial sampling, the process of selectively simulating points from a comprehensive space, reduces the number of simulations required for design space exploration. Traditionally, designers mitigate simulation costs with temporal sampling, reducing the number of instructions simulated to control the time required for any one design simulation [16, 58, 59, 62, 70]. However, temporal sampling does not impact the number of simulations required for design space exploration. Spatial sampling, in contrast, enables designers to specify and to explore a comprehensive design space knowing that detailed simulation for every design point is no longer required.

Given sparsely sampled and simulated designs, inferential models reveal broader trends and salient details. In particular, data from sparsely collected simulations train spline-based regression models to predict metrics of interest, such as performance and power. Both model training and evaluation may be expressed as matrix operations (*i.e.*, linear solve and linear combination) and are, therefore, computationally efficient. Thus, inference provides efficient surrogates for detailed simulation;

the models capture the input-output relationships of simulators. Instead of relying on further simulation for design studies, this dissertation proposes relying on efficient models for accurate inference within these studies.

1.4 Qualitatively New Capabilities

Statistical inference is computationally efficient, allowing designers to evaluate inferential models instead of running detailed simulations for comprehensive design space exploration. This efficiency enables qualitatively new capabilities in design characterization (Chapter 3) and optimization (Chapter 4). Not only can designers answer prior questions more quickly, they can answer new questions previously intractable with simulation.

Design characterization has long been constrained by the microarchitectural curse of dimensionality. High dimensionality and expensive simulation resulted in sensitivity analyses constrained in scope and lacking generality. This dissertation demonstrates the pitfalls of such an approach and demonstrates more rigorous alternatives made possible by the proposed simulation paradigm. In particular, inferential models provide performance and power estimates to fully characterize hundreds of thousands of designs and to quickly identify pareto frontiers from this characterization.

Similarly, the proposed simulation paradigm trivializes construction costs for contour maps, useful for revealing bottlenecks and comparing workload behavior across the design space. Given a plethora of inexpensively constructed contour maps, this dissertation computes roughness metrics that quantify observed non-linearities and non-monotonicities. These metrics reduce the subjectivity and qualitative judgment

currently required from designers using contour maps. Ranking contours by roughness, designers can focus attention on the roughest contours, which likely contain the most interesting compromises or optima. Further illustrating the capabilities of inferential models, these roughness metrics require numerical integrals and derivatives computed on empirically constructed performance and power models.

This dissertation also illustrates new capabilities in robust optimization enabled by statistical inference. Exhaustive search, which evaluates every design to identify an optimum, is tractable in simulation only for highly constrained design spaces. In contrast, designers relying on inferential models can easily estimate design metrics for spaces with hundreds of thousands of points to identify optima. For more comprehensive spaces with million's or billion's of points, this dissertation illustrates robust heuristic optimization, which includes techniques such as gradient ascent and genetic algorithms. These heuristics iteratively traverse the design topology, estimating design metrics as they step toward an optimum. Replacing detailed simulation with inferential models within the iterative loop enables a far greater number of heuristic trials, iterations per trial, and metric estimates per iteration, thereby improving heuristic effectiveness.

Furthermore, this dissertation combines inference with robust optimization to understand emerging design priorities. In particular, trends and limits in performance and power efficiency are quantified for two fundamental design schemes: multiprocessor heterogeneity and microarchitectural adaptivity. Heterogeneous multiprocessors deliver greater efficiency by using multiple and different core designs on a single chip. The degree of heterogeneity and the design of each core determine efficiency gains.

Adaptive microarchitectures deliver greater performance and power efficiency by over-provisioning on-chip resources and providing these resources only during application intervals that require them. Thus, adaptivity enhances performance while localizing associated power costs. Comprehensive analyses of heterogeneity and adaptivity, prohibitively expensive using detailed simulation and made possible by inferential models, provide fundamental insight into potential efficiencies from these design schemes.

1.5 Summary of Contributions

We contribute to methodology proposing a simulation paradigm that leverages statistical inference to address fundamental challenges in microarchitectural simulation. Chapter 2 examines:

- **Spatial Sampling:** We propose sampling, sparsely and uniformly at random, points from a design space to reveal trends and trade-offs in microarchitectural design metrics, such as performance and power. [42]. (Section 2.1)
- **Statistical Inference:** We use spline-based regression models, trained from comprehensive but sparsely simulated spaces, as accurate and efficient surrogates for detailed simulation [42]. (Sections 2.2–2.3)

We then apply the computational efficiency of statistical inference to enable qualitatively new capabilities in characterizing microarchitectural performance and power. In particular, Chapter 3 examines:

- **Comprehensive Characterization:** We construct comprehensive pareto fron-

tiers and contour maps for large, previously intractable design spaces with hundreds of thousands of points [43, 45]. (Sections 3.3–3.2).

- **Roughness Metrics:** We quantify the roughness of performance and power topologies using numerical derivatives and integrals, using these roughness metrics to identify interesting topologies for designer attention [45]. (Section 3.4)

We combine statistical inference with robust optimization, demonstrating qualitatively new capabilities in performance and power optimization for emerging design priorities. Chapter 4 examines:

- **Robust Optimization:** We show iterative optimization heuristics are more effective with more iterations, which are most tractably provided by inferential models within the iterative loop [45]. (Section 4.1)

- **Multiprocessor Heterogeneity:** We quantify the trends and limits of performance and power efficiency from multiprocessor heterogeneity by combining inferential models with heuristic clustering [43]. (Section 4.2)

- **Microarchitectural Adaptivity:** We quantify trends and limits of performance and power efficiency from microarchitectural adaptivity by combining inferential models with heuristic optimization [44]. (Section 4.3)

Collectively, this dissertation applies best-known practices in statistical inference and optimization to fundamentally address microarchitectural simulation costs. These methodologies enable the robust and comprehensive analysis required for a successful transition to the era of chip multiprocessors.

Chapter 2

Statistical Inference

Contents

This chapter presents a simulation paradigm for microarchitectural design evaluation and optimization, countering simulation costs attributed to exponentially increasing design space sizes. As shown in Figure 1.3, the proposed approach (1) provides a more comprehensive understanding of the design space by (2) selectively simulating a modest number of designs from that space and (3) effectively leveraging simulation data using statistical inference.

Within this paradigm, robust and effective statistical inference reveals salient trends from a broadly defined, sparsely sampled design space. These techniques modestly reduce detail for substantial gains in speed and tractability. Even in scenarios where collecting extensive measurement data is feasible, efficient analysis of this data often lends itself to statistical modeling. These models typically require an initial data set for model formulation or training. Once trained, the model responds to predictive queries by leveraging correlations in the original data for inference. Thus, inference increases simulation efficiency by increasing the information content from any given number of simulations.

Regression models implement inference in a cost effective manner, formulating models from observed data by numerically solving a system of linear equations and predicting unobserved data by evaluating a linear system. Highly optimized numerical linear algebra libraries lead to computationally efficient models, enabling thousand's of predictions per second. Predicting design metrics (e.g., performance and power) as a function of tunable design parameters (e.g., pipeline depth and cache sizes), regression equations are efficient surrogates for cycle-accurate simulation. As surrogates, models enable a broad range of analyses without further simulation.

This chapter surveys statistical regression theory and details its application to the microarchitectural design space. We construct predictive regression models using a statistically rigorous framework:

- **Spatial Sampling:** We define a large, comprehensive design space and sparsely sample, uniformly at random, designs for detailed simulation. Each simulated sample maps a set of microarchitectural design parameters to a set of design metrics. These samples train regression models, which express performance and power as functions of design parameters. (Section 2.1)

- **Model Derivation:** We consider a general class of regression models that estimates a *response* as a linear combination of *predictors* with random error. We take a statistically rigorous approach to construct these regression models, beginning with exploratory data analysis to determine the significance of each design parameter. When defining the regression models' functional form, we account for interactions between predictors and non-linear predictor-response relationships. (Section 2.2)

- **Model Evaluation:** After fitting the regression model, we evaluate its fit to the training data, check its residuals for randomness and normality assumptions, and test its accuracy for independently collected validation data. (Section 2.3)

Collectively, this statistically rigorous approach provides efficient and robust regression models, which address fundamental limitations in current simulators.

2.1 Spatial Sampling

Spatial sampling decouples design space size from the number of simulations required to expose trends by simulating only 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 overall simulation costs of design space exploration by reducing costs per simulation and number of required simulations, respectively. For spatial sampling, we propose drawing points uniformly at random from the design space and outline the advantages of this approach with respect to complexity and modeling. Lastly, we survey alternative sampling strategies, highlighting relative strengths and weaknesses.

2.1.1 Spatial and Temporal Synergies

Efforts to control simulation costs have focused primarily on *temporal sampling*. These techniques extract representative samples from instruction traces in the time domain, reducing the costs per simulation by reducing the number of instructions simulated [16, 58, 59, 62, 70]. 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 comprehensive design space. This limitation often constrains design space exploration since the number of simulations increases exponentially with the number of design parameters.

We must supplement temporal sampling with *spatial sampling*, a technique that samples points from the design space for simulation to control exponentially increasing design space sizes. Spatial sampling also mitigates the inefficiencies of traditional

simulation techniques that sweep design parameter values, exhaustively simulating all points defined within a tightly constrained space. By decoupling space size from the number of simulations, spatial sampling enables the study of larger, higher resolution design spaces. Specifically, we consider many design parameters simultaneously and let each parameter assume one of many different values.

2.1.2 Uniformly Random Sampling

We propose sampling designs *uniformly at random* (UAR) from the space S containing $|S|$ points. This approach provides observations drawn from the full range of parameter values. Each parameter may assume one of an arbitrarily large number of possible values since parameter resolution is decoupled from the number of required simulations. 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 the microarchitectural context, the fact a design point is missing is unrelated to the performance, power, or configuration of the design.

In contrast, *informative missing* describes the case where elements are more likely missing if their responses are systematically higher or lower. For example, simulator limitations may prevent data collection for very low performance architectures and

"missingness" of a configuration is correlated with its performance. In this case, "missingness" is non-ignorable and we need an additional model to predict whether a design point can be simulated. By sampling UAR from the design space, we ensure observations are MCAR and avoid such modeling complications.

We construct regression models using sparsely collected design space samples. Each sampled design is simulated for every workload of interest. Simulator reported performance and power numbers will provide data necessary for constructing regression models. Although we use samples obtained uniformly at random from the design space, a number of alternative sampling strategies may also apply.

2.1.3 Alternative Sampling Strategies

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

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

- **Regional sampling** also emphasizes samples from particular design regions given samples from the broader space. Instead of using a continuous range of weights, this approach specifies a region of interest and excludes undesired samples during model training (effectively binary weights). Regional sampling

might be used to construct localized models from samples collected uniformly at random from the entire space. This approach may be necessary if regions of interest are unknown prior to sampling but become known after exploratory data analysis [42].

- **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 error-prone samples are iteratively added to the training set, with each iteration choosing a sample with large error variance and most different from those already added [18].

- **Latin hypercube sampling and space-filling** seek to maximize design space coverage. Hypercube sampling guarantees each parameter value is represented in the sampled designs. Space-filling metrics are used to select the most uniformly distributed sample from the large number of hypercube samples that exist for any given design space [35].

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

that are only approximated by uniform at random sampling, but such a guarantee increases sampling complexity. Collectively, these sampling strategies provide options for improving the accuracy of models constructed with these samples. We find, however, sampling UAR is sufficient for accurate models and comprehensive design optimization.

2.2 Model Derivation

This section provides the theoretical background for relevant statistical regression theory and demonstrates its application to microarchitectural performance and power modeling. Model derivation in statistical inference requires empirical data and the transfer of domain knowledge from user to model. Empirical data is necessary to construct a surrogate for the underlying generator of this data. Model construction begins with exploratory data analysis, which guides the user toward significant relationships between predictors (model inputs) and responses (model outputs). The user combines significance analyses with domain-specific knowledge to specify the model's functional form. The derivation approach is summarized in four steps:

- **Hierarchical Clustering:** Clustering examines correlations between candidate predictors and exposes redundancy among them. Pruning redundant predictors controls model size, thereby reducing risk of over-fitting and improving model efficiency during training and prediction.

- **Association Analysis:** Scatterplots qualitatively capture trends in predictor-response relationships, revealing the degree of non-monotonicity or non-linearity.

Scatterplots with little response variation as predictor values change may suggest predictor insignificance and enable further pruning.

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

- **Model Specification:** Domain-specific knowledge is used to specify predictor interactions. 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 more significantly impact model accuracy. Given the model's functional form, the method of least squares determines regression coefficients.

This derivation assumes two sets of data are available: a sizable training set and a smaller validation set. Data sets are sampled uniformly at random from the design space and evaluated with detailed microarchitectural simulation for representative workloads (Appendices A–C). To simplify the exposition, this chapter focuses on twelve design parameters spanning a space of nearly one billion points (Table 2.1). Regression models are constructed using up to 4,000 training points for each of nine SPEC benchmarks (Table 2.2). Section 2.5 discusses derivation extensibility, demonstrating comparable accuracy with fewer training samples, larger design spaces, and non-SPEC workloads.

	Set	Parameters	Measure	Range	$\|S_i\|$
S_1	Depth	Depth	FO4	9::3::36	10
S_2	Width	Width	issue b/w	2,4,8	3
		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 Purpoes (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 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	Main Memory	Main Memory Latency	cycles	70::5::115	10
S_9	Control Latency	Branch Latency	cycles	1,2	2
S_{10}	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_{11}	Floating-Point Latency	FPU Latency	cycles	5::1::9	5
		FP-Divide Latency	cycles	25::5::45	
S_{12}	Memory Latency	Load/Store Latency	cycles	3::1::7	5

Table 2.1: **Design Space I**. Used for initial model derivation and proof of concept. $p = 12$, $|S| = 9.4\text{E+}8$.

SPEC CPU 2000	
ammp	Simulates molecular dynamics
applu	Solves parabolic/elliptic partial differential equations (PDE's)
equake	Simulates seismic wave propagation
gcc	Compiles C programs
gzip	Performs compression
mcf	Performs combinatorial optimization
mesa	Provides 3-D graphics library support
twolf	Simulates circuit place and route
SPEC JBB 2000	
jbb	3-tier Java business server

Table 2.2: **Benchmarks**.

2.2.1 Hierarchical Clustering

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. The heuristic is initialized by assigning each element to its own cluster, producing N single-element clusters. Each iteration merges the most similar pair of clusters into a single cluster 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 clusters.

Hierarchical clustering reveals potential redundancy in the data set. If multiple predictors are highly correlated and are classified into the same cluster, a single representative predictor may capture the cluster's impact on the response. Similarly,

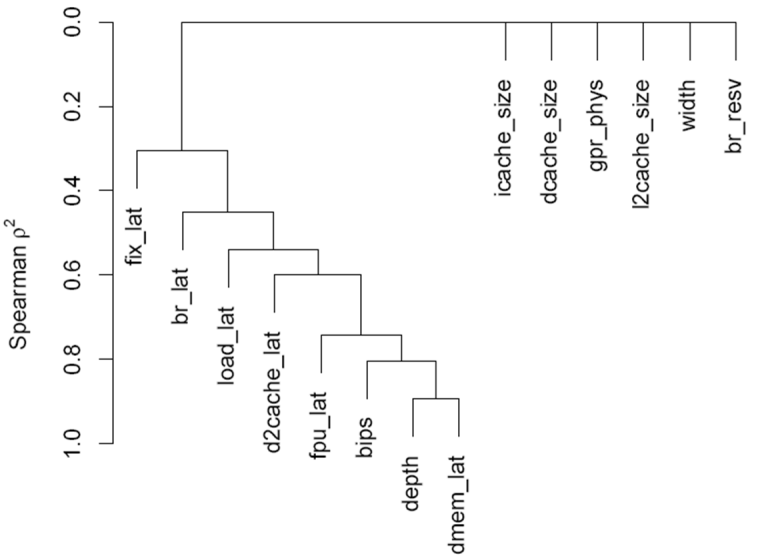

Figure 2.1: **Hierarchical Clustering**. The level at which clusters connect indicates their degree of similarity. Spearman ρ^2 is a rank-based measure of correlation. Example for design space of Table 2.1 and nine SPEC benchmarks of Table 2.2.

if multiple responses are highly correlated, a single representative response may be modeled since the other correlated responses will likely scale with the chosen response. Pruning predictors is important to control model size by controlling the number of predictors and model terms. Smaller models are preferable as they reduce the number of sampled training points required for model training.

Figure 2.1 presents an example of hierarchical clustering. Correlation coefficients are used as a similarity metric and larger ρ^2 indicates greater correlation. The level at which clusters meet indicates their degree of similarity. This design space considers effects from circuit-level tuning for each functional unit by varying the unit's latency (measured in cycles). For example, a more efficient arithmetic-logic unit (ALU) design may require one cycle less than that required by a baseline design. Independently, we vary pipeline depth assuming logic for each functional unit may be perfectly divided

across pipeline stages. For example, an ALU may implement 3 pipeline stages (*e.g.*, latency of 3 cycles) with 18 FO4 delays per stage.[1] If we consider a shallower pipeline with 27 FO4 delays per stage, each stage becomes 50 percent longer and logic will require 67 percent the original number of stages. The same ALU logic originally implemented with 3 stages may now be implemented with 2 stages ($2 = 3 \times 18/27$). Thus, pipeline depth is highly correlated with functional unit latency since deeper pipelines imply a larger number of shorter cycles for each unit. Including both latency and depth predictors allows us to differentiate the performance impact of changes to individual functional unit latencies from changes to global latency as depth varies. However, the clustering analysis suggests these effects are insignificant and we should remove these extra latency parameters to improve model efficiency.

2.2.2 Association Analysis

Scatterplots qualitatively capture the association between variables. Such plots illustrate associations between predictors and the response, revealing non-monotonicity or non-linearity. Scatterplots quickly reveal 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.

Figure 2.2 presents an example association analysis. Figure 2.2L summarizes the association between performance and four microarchitectural predictors: pipeline

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

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

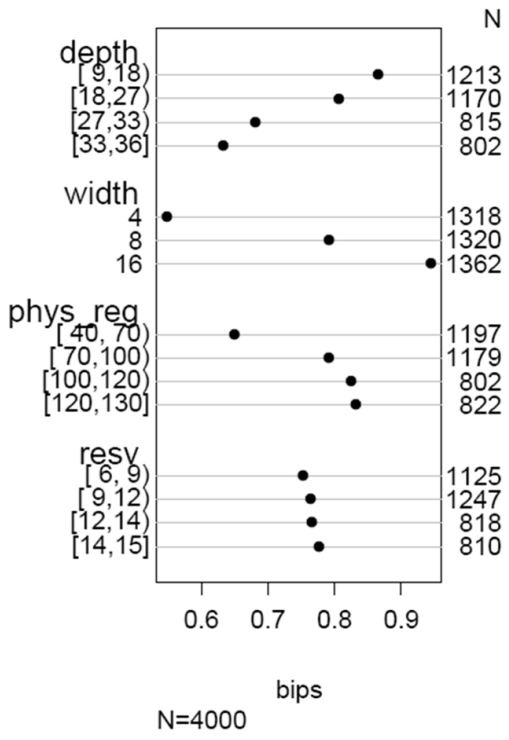

Figure 2.2: **Association Analysis**. Table summarizes ranges of parameter values (second column), number of samples in each range (third column), and average performance of these samples (fourth column) (L). Scatterplot visualizes this data (R). Example for design space of Table 2.1 and nine SPEC benchmarks of Table 2.2.

depth, superscalar width, register file size, and reservation station size. Predictor ranges are divided into intervals and the average performance response is reported for each interval. For example, pipeline depth takes a value between 9 and 18 FO4 delays per stage for 1213 of 4000 samples. The average performance for these designs is 0.865 billion instructions per second.

Figure 2.2R 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; we observe diminishing marginal returns in performance as we increase the file size. Performance increases by 21.7 percent from interval [40,70) to interval [70,100), but only increases by 4.4 percent from interval [70,100) to [100,120). Finally, the number of entries in reservation stations has a relatively small performance impact.

2.2.3 Correlation Analysis

In addition to qualitative associations between performance and design parameters, we also quantify association strength using correlation coefficients. Pearson's correlation coefficient between two random variables is computed by Equation (2.1) where X,Y are random variables with expectations μ_x,μ_y and standard deviations σ_x,σ_y.

$$\rho = \frac{E((X-\mu_X)(Y-\mu_Y))}{\sigma_X \sigma_Y} \tag{2.1}$$

Pearson's correlation coefficient assumes X, Y follow Normal distributions and differences between X, Y can be meaningfully compared. Such assumptions are often too restrictive. For example, the second assumption is difficult to justify when consider-

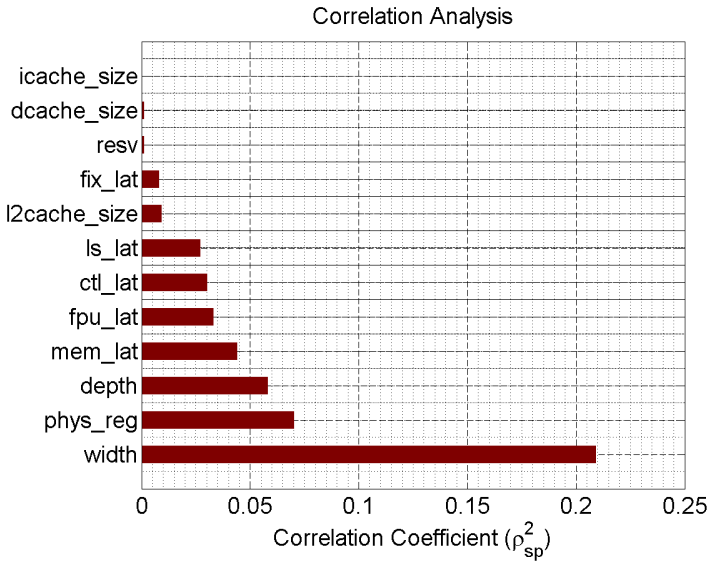

Figure 2.3: **Correlation Analysis**. Plots squared Spearman rank correlation between microarchitectural parameters and performance. Example for design space of Table 2.1 and nine SPEC benchmarks of Table 2.2.

ing differences between performance and pipeline depth measured in instructions per second and FO4 delays per stage, respectively.

In cases where the distributions 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 requires only d_i, the difference in ordinal rank of x_i in X and y_i in Y. Supposing $X = (x_1, x_2, \ldots, x_N)$ and $Y = (y_1, y_2, \ldots, y_N)$, we first compute x_i's rank in X, y_i's rank in Y. We then compute the Spearman rank correlation using the N differences in rank as shown by Equation (2.2).

$$\rho_{sp} = \frac{\sum_{i=1}^{N} X_i Y_i}{\sqrt{\sum_{i=1}^{N} X_i^2 \sum_{j=1}^{N} Y_j^2}} \approx 1 - 6 \sum_{i=1}^{N} \frac{d_i^2}{N(N^2-1)} \tag{2.2}$$

Spearman's rank correlation coefficient assumes ranks capture equi-distant positions

on the measured variable. In particular, the distance between values ranked R_i, R_{i+1} is equal to the distance between values ranked R_j, R_{j+1} for any $1 \leq i, j < N$. This assumption holds for our training data, which is sampled uniformly at random from the full range of possible values for each design parameter.

Figure 2.3 presents an example correlation analysis, plotting the squared correlation between microarchitectural predictors and performance. This information influences the degree of non-linear flexibility used to specify the regression model. Greater flexibility likely improves fit and fit is more important for significant design parameters with high ρ^2_{sp}. Lack of fit for predictors with high ρ^2_{sp} will have a greater negative impact on accuracy. For microarchitectural predictors, a lack of fit will be more consequential (in descending order of importance) for width, register file size, depth, functional unit latencies, L2 cache size, and reservation stations.

2.2.4 Model Specification

We apply a general class of models in which a response is modeled as a linear combination of predictor variables plus random noise. Within this modeling framework, we consider more advanced techniques to account for potentially non-linear relationships.

Formulation: For a large universe of interest, suppose we have a subset of n observations for which values of the response and predictor variables are known. Let Y denote a vector of observed responses and X denote a matrix of predictors for those responses. Each sample i from this universe has a response y_i and p predictors $x_{i,1}, \ldots, x_{i,p}$. As shown in Equation (2.3), let β denote regression coefficients that describe the response as a linear function of predictors plus random error ε. Mathe-

matically, β_j may be interpreted as the expected change in y per unit change in the predictor variable x_j. The error ε is assumed to be independent random Normally distributed variables with zero mean and constant variance: $\varepsilon_i \sim N(0, \sigma^2)$.

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} x_{11} & \dots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{np} \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_n \end{bmatrix} \quad \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

In the microarchitectural context, the response Y is a metric of interest. This dissertation focuses on two metrics: performance measured in billion's of instructions per second (*bips*) and power dissipation measured in Watts (*watts*). Predictors X are design parameters, such as pipeline depth or L2 cache size, as described in Table 2.1. Regression models predict design metrics as a linear combination of design parameters after accounting for any non-linear transformations.

$$Y = X\beta + \varepsilon \tag{2.3}$$

$$F(Y) = G(X)\beta_G + \varepsilon \tag{2.4}$$

Transformations F and G may be applied to response and predictors, respectively. If G increases the number of terms in the model, the number of regression coefficients in β_G increases accordingly. We find a square-root transformation on the response $F(Y) = [\sqrt{y_1}, \dots, \sqrt{y_n}]^T$ is effective for reducing error variances if Y is the performance metric. Similarly a log transformation $F(Y) = [log(y_1), \dots, log(y_n)]^T$ effectively captures exponential trends if Y is the power metric. These square-root

and log transformations are common strategies for improving model fit by mitigating a non-constant error variance, which violates our assumption $\varepsilon_i \sim N(0, \sigma^2)$. We also consider cubic splines G that transform predictors into piecewise cubic polynomials as described later in this section. The spline transformations improve model fit by accounting for non-linear response-predictor relationships.

$$S(\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \tag{2.5}$$

Least squares identifies the best-fitting model for a set of training data. This method finds coefficients β to minimize $S(\beta)$, the sum of squared errors between the predicted responses (given by model) from the actual observed responses (given by simulator). $S(\beta)$ may be minimized by solving a system of $p+1$ partial derivatives of S with respect to β_j, $\delta S(\beta)/\delta\beta_j = 0$ for $j \in [0, p]$. The solutions to this system are estimates for β and may often be expressed in closed form. Statistical properties of the solution form the basis for significance testing using common techniques, such as T-tests or F-tests.

Interaction: Consider a response y and two predictors x_1, x_2. In some cases, the effects of x_1 and x_2 cannot be separated; the effect of x_1 on y depends on the value of x_2 and vice versa. The interaction between two predictors may be modeled by constructing a third predictor $x_1 x_2$ to obtain the model of Equation (2.6). Such product terms capture interactions between x_1, x_2 since the impact of one parameter, expressed as a partial derivative, is a function of the other.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \varepsilon \tag{2.6}$$

$$\frac{\delta y}{\delta x_1} = \beta_1 + \beta_3 x_2 \tag{2.7}$$

Interactions complicate direct interpretations of regression coefficients. The partial derivative $\delta y/\delta x_i$ equals the regression coefficient β_i only in a model without interactions. Splines will further complicate the interpretation of coefficients. For these more sophisticated models, the partial derivative $\delta y/\delta x_i$ is the most accurate way to assess the impact of x_i on y.

We draw on domain-specific knowledge to specify interactions. Pipeline depth interacts with cache sizes; a smaller L2 cache leads to additional memory hazards in the pipeline. These stalls, in turn, affect instruction throughput gains from pipelining as shown in Equation (2.8). Thus, their joint impact on performance must be modeled. Expressing this interaction in the notation of Equations (2.6)–(2.7), let y be performance, x_1 be pipeline depth, and x_2 be L2 cache size. Equation (2.8) indicates speedup from pipelining increases with depth and cache size (due to fewer memory stalls). Thus, $\delta y/\delta x_1$ should increase with x_2, which implies $\beta_3 > 0$.

$$Speedup = \frac{Depth}{1 + Stalls/Inst} \tag{2.8}$$

In a similar fashion, we expect pipeline width to interact with the register file. The register file must supply operand data to all functional units for any given superscalar width. We also expect interactions between adjacent levels in the cache hierarchy. An inclusive cache hierarchy and locality effects suggest L1 and L2 cache size interactions. 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 this high-level representation is sufficient (Section 2.3).

Non-Linearity: Several techniques for capturing non-linearity may be applied to improve accuracy. The simplest technique is a polynomial transformation on predic-

tors suspected of having a non-linear correlation with the response. However, polynomials have undesirable peaks and valleys. 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.

$$G(x) = [1 \quad x \quad (x-k_1)_+ \quad (x-k_2)_+ \quad (x-k_3)_+] \tag{2.9}$$

$$\begin{aligned} y &= G(x)\beta_G \\ &= \beta_0 + \beta_1 x + \beta_2(x-k_1)_+ + \beta_3(x-k_2)_+ + \beta_4(x-k_3)_+ \end{aligned} \tag{2.10}$$

Spline functions are piecewise polynomials used in curve fitting. The function is divided into intervals each defining continuous polynomials joined at endpoints called *knots*. The number of knots varies depending on the amount of available data for fitting the function, but more knots generally leads to better fits. A linear spline (*i.e.*, piecewise linear function) on x with three knots at k_1, k_2, and k_3 is given by Equation (2.10) where $(u)_+ = u$ if $u > 0$ and $(u)_+ = 0$ otherwise.

$$G(x) = \left[1 \quad x \quad x^2 \quad x^3 \quad (x-k_1)_+^3 \quad (x-k_2)_+^3 \quad (x-k_3)_+^3\right] \tag{2.11}$$

$$\begin{aligned} y &= G(x)\beta_G \\ &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4(x-k_1)_+^3 + \beta_5(x-k_2)_+^3 + \beta_6(x-k_3)_+^3 \end{aligned} \tag{2.12}$$

Linear splines may be inadequate for complex, highly curved relationships. Splines of higher order polynomials may offer better fits and cubic splines have been found particularly effective [25]. 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. For example, a cubic spline on x with three knots is given by Equation (2.12).

$$
\begin{aligned}
G(x) &= [1 \quad x_1 \quad x_2 \quad \ldots \quad x_{m-1}] && (2.13)\\
x_1 &= x \\
x_{j+1} &= (x-k_j)_+^3 - (x-k_{m-1})_+^3(k_m-k_j)/(k_m-k_{m-1}) \\
&\quad +(x-k_m)_+^3(k_{m-1}-k_j)/(k_m-k_{m-1}) && (2.14)\\
y &= G(x)\beta && (2.15)\\
&= \beta_0+\beta_1x_1+\beta_2x_2+\ldots+\beta_{m-1}x_{m-1}
\end{aligned}
$$

Cubic splines may have poor behavior in the tails before the first knot and after the last knot [66]. Restricted cubic splines that constrain the function to be linear in the tails are often better behaved and have the added advantage of fewer terms relative to cubic splines. A restricted cubic spline on x with m knots $k_1, \ldots, k_m$ is given by Equation (2.15) where $j = 1, \ldots, m-2$ [13].

Figure 2.4 illustrates schematically a restricted cubic spline with five knots and linear tails. The choice and position of knots are tunable 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 [25]. Recommended equally spaced quantiles are shown in Table 2.3. In practice, five knots or fewer are generally sufficient for restricted cubic splines [65]. 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.

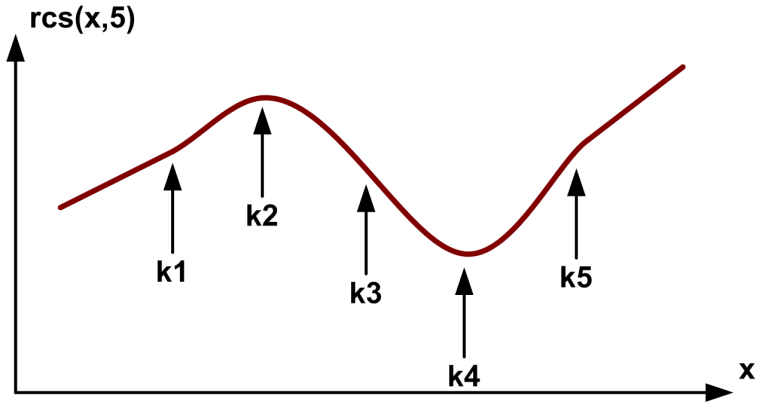

Figure 2.4: **Restricted Cubic Spline**. Range of x divided using five knots with polynomial interiors, linear tails.

K	Quantiles						
3			0.1000	0.5000	0.9000		
4			0.0500	0.3500	0.6500	0.9500	
5		0.0500	0.2750	0.5000	0.7250	0.9500	
6	0.0500	0.2300	0.4100	0.5900	0.7700	0.9500	
7	0.0250	0.1833	0.3417	0.5000	0.6583	0.8167	0.9750

Table 2.3: **Recommended Knot Placement**. K knots are placed at fixed quantiles of the data [25].

In the microarchitectural context, we specify restricted cubic splines with varying flexibility achieved by varying knot counts. The correlations between predictors and the response from Section 2.2.3 guide our choice in the number of knots. Predictors with stronger correlations, and therefore more important to model accuracy, are assigned a greater number of knots. For example, predictors with stronger relationships (*e.g.* pipeline depth, register file size) will use four knots while those with weaker relationships (*e.g.* cache sizes, reservation station sizes) will use three knots. A predictor must exhibit more than three unique values to use cubic splines. Despite its strong correlation with performance, superscalar width does not take a sufficient number of unique values to apply splines and we consider its linear effects only.

In practice, we apply restricted cubic spline transformations before specifying interactions. As shown in Equation (2.14), splines significantly expand the number of terms for each predictor x. This expansion is problematic when we express interactions using products of predictors and attempt to multiply sums of cubic binomials. Such interactions will rapidly increase model size and training costs. To control these effects when using restricted cubic splines G_1 and G_2, we consider interactions of the form $x_1 G_2(x_2)$ and $x_2 G_1(x_1)$, thereby ignoring doubly non-linear terms and reducing the size of these product terms [25].

$$E[\hat{Y}] = E[\hat{X}\beta + \varepsilon] = E[\hat{X}\beta] + E[\varepsilon] = \hat{X}\beta \tag{2.16}$$

Prediction: A model is trained with least squares to determine regression coefficients β. A set of design queries or predictions are defined by $\hat{X}$ and evaluated with Equation (2.16). The expected response $E[\hat{Y}] = \hat{X}\beta$ is efficiently obtained by a matrix-vector multiply. This result follows from observing the additive property

of expectations, the expectation of a constant $\hat{X}\beta$ is $\hat{X}\beta$, and the random errors are assumed to follow a normal distribution $\varepsilon \sim N(0, \sigma^2)$.

2.3 Model Evaluation

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 captures the same variations reported by simulators. We then check the model for systematic bias to ensure robustness. Ideally, model error is random and independent of predicted metric values. For example, a biased and less robust model might consistently over-predict performance for low-performance designs. Lastly, we assess model accuracy by comparing model predictions against detailed simulation using an independent validation data. Collectively, these analyses give users confidence in the robustness and accuracy of the model.

2.3.1 Evaluating Fit

The model's fit to training data is quantified with the *multiple correlation statistic* R^2 in Equation (2.19). 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. 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. Over-fitting typically occurs when too many predictors in X are used to estimate relatively small data sets.

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.17}$$

$$SST = \sum_{i=1}^{n}\left(y_i - \frac{1}{n}\sum_{i=1}^{n} y_i\right)^2 \tag{2.18}$$

$$R^2 = 1 - \frac{SSE}{SST} \tag{2.19}$$

The median R^2 values across the benchmark suite are 0.94 and 0.99 for performance and power, respectively, for the design space of Table 2.1 and nine SPEC benchmarks of Table 2.2. 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 [25]. In this example, we use 4,000 training samples to construct models with approximately 80 terms, thereby limiting risk of over-fitting. Furthermore, despite the close fit to training data, Section 2.3.3 suggests our models are accurate not only for training data but also for separately collected validation data.

2.3.2 Evaluating Bias

Model residuals are examined to ensure predictions are unbiased. These residuals, defined in Equation (2.20) and illustrated in Figure 2.5, are per sample differences between predicted and observed responses in the training set. In particular, we validate the following assumptions to ensure model robustness:

1. residuals are not correlated with predicted response

2. residuals have a normal distribution with zero mean and constant variance

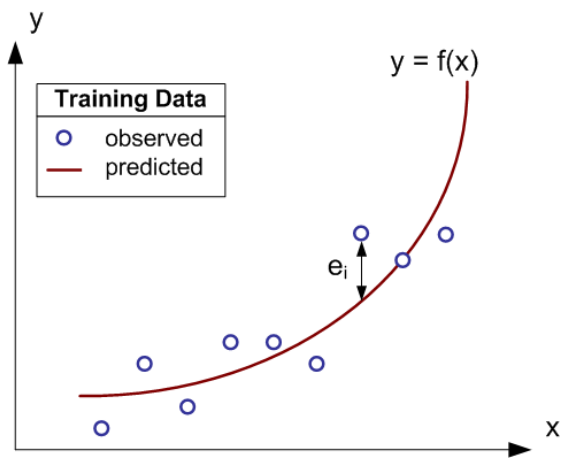

Figure 2.5: **Residual Schematic**. Differences between observed and predicted values in training data.

$$\varepsilon_i = y_i - \beta_0 - \sum_{j=0}^{p} \beta_j x_{ij} \tag{2.20}$$

The first assumption is typically validated with scatterplots of residuals against predicted responses since such plots may reveal systematic deviations from randomness. For example, Figure 2.6 plots the median, lower, and upper quartiles of performance residuals before and after transforming the *bips* response. Before the transformation, Figure 2.6L indicates significant correlations between residuals and fitted values; residuals are biased positive for the smallest and largest fitted values. We apply a square-root transformation on performance, a typical variance stabilizing technique to mitigate the magnitude of these correlations. Figure 2.6R shows stabilized residuals.

The second 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 $\varepsilon^{(1)}, \ldots, \varepsilon^{(n)}$, obtaining n ranked samples from the Normal

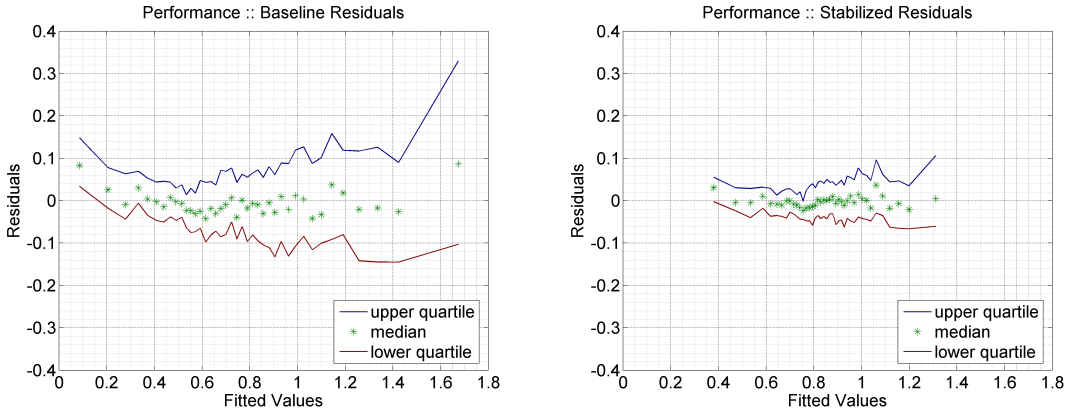

Figure 2.6: **Residual Randomness**. Scatterplots of residuals before and after square-root transformation. Residuals of an unbiased model should appear randomly and independently distributed around zero. Example for design space of Table 2.1 and nine SPEC benchmarks of Table 2.2.

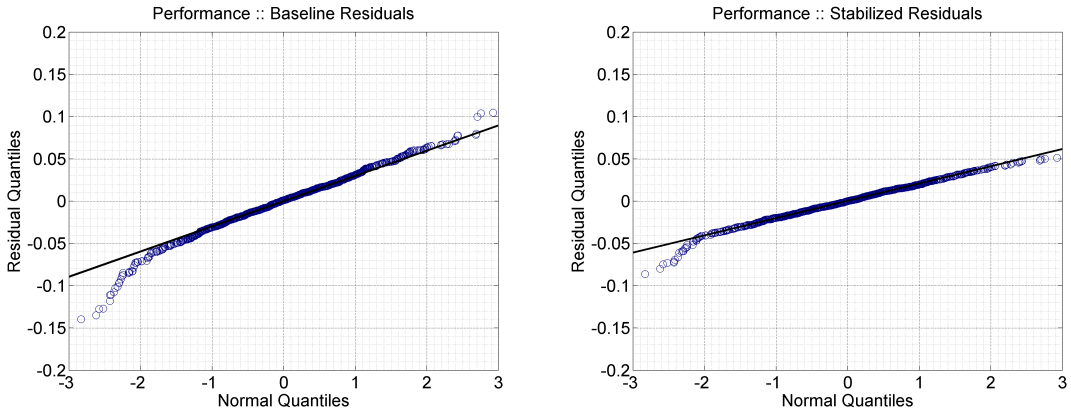

Figure 2.7: **Residual Normality**. Quantile-Quantile plots of residuals before and after square-root transformation. Plots for an unbiased model should appear linear. Example for design space of Table 2.1 and nine SPEC benchmarks of Table 2.2.

distribution $s^{(1)}, \ldots, s^{(n)}$, and producing a scatterplot of $(\varepsilon^{(i)}, s^{(i)})$ that should appear linear if the residuals follow a Normal distribution. Variance stabilization also causes the residuals to follow a Normal distribution more closely as shown in Figure 2.7. The observed linear trend indicates the residuals follow the Normal distribution when Normal and residual quantiles are plotted on the x-axis and y-axis, respectively.

2.3.3 Evaluating Accuracy

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 is 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×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

Figure 2.8L indicates the performance model achieves median errors ranging from 3.7 percent (ammp) to 11.0 percent (mesa) with an overall median error across all benchmarks of 7.2 percent. Figure 2.8R indicates power models are slightly more

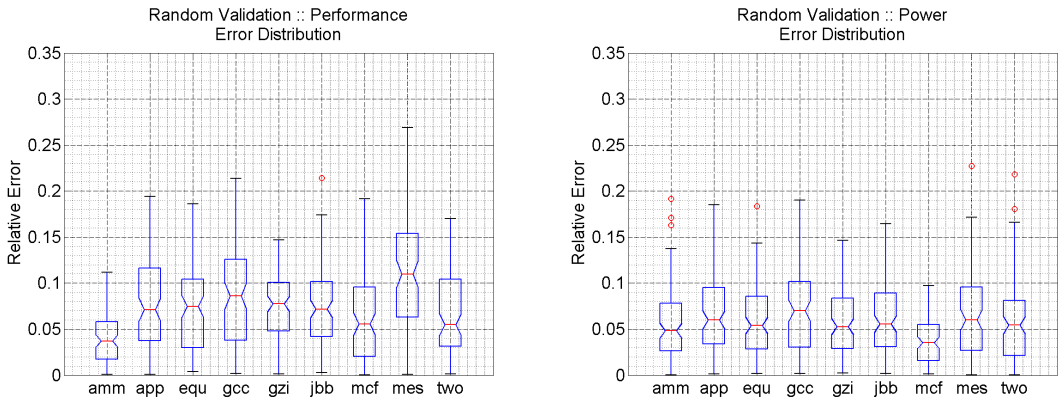

Figure 2.8: **Model Error Distributions**. Boxplots capture error distributions for performance (L) and power (R) predictions on validation set of 100 random designs. Example for design space of Table 2.1 and nine SPEC benchmarks of Table 2.2.

accurate with median errors ranging from 3.5 percent (mcf) to 7.0 percent (gcc) and an overall median of 5.4 percent. These error rates are sufficient for early stage design analysis and optimization.

2.3.4 Alternative Modeling Strategies

Other modeling strategies might be applied to the microarchitectural design space. These methodologies differ in complexity and effectiveness.

- **Decision trees** partition the space of designs into subspaces based on the significance of predictors. For example, if predictor x_1 is the most significant predictor of a response y, the space of designs will be partitioned into subspaces based on the values of x_1 so that different subspaces contain designs with different values of x_1. Each subspace is recursively partitioned based on the values of its most significant predictor to create a tree-representation of a partitioned design space. Leaves in the tree are subspaces that contain similar designs x

and similar responses y. Predictions are obtained by traversing the tree, following branches based on the partitioning of the design parameters until a leaf is reached and the average response for the leaf's designs is used to estimate the response [51, 52].

- **Regression trees** are similar to decision trees except leaves estimate the response with regression models fit to designs within the leaf. Intuitively, regression trees may be shallower than decision trees since regression models can take incompletely partitioned nodes and infer any trends that would arise from a further, more complete partitioning. Thus, regression models can handle design heterogeneity at a leaf whereas decision trees are most effective when reporting the average from a space of homogeneous designs at a leaf [53].

- **Instance-based learning** predicts a response as a weighted average of neighboring responses. Weights are a function of distance and kernel width. When using larger kernel widths, more distant neighbors contribute to the weighted average. Various kernels might be used to compute the weights as a function of distance, but a Gaussian weighting function is a typical example [51, 54].

- **Neural Networks** are a class of machine learning models that map predictors to a response using a network of neurons, simple processing elements, connected by weighted edges. As in regression, data is required to train the edge weights to construct the model. Predictions are obtained by feeding data into the network and computing weighted sums as data propagates through the network [51].

Predictions from inference trees and neural networks are likely more expensive than those from spline-based regression since they require graph traversals to arrive at leaves or output neurons for every prediction. In contrast, spline-based regression first performs non-linear transformations for the splines and then performs a matrix-vector multiply where the number of matrix rows equals the number of desired predictions. Prediction times are more scalable when model evaluation is expressed as linear algebra instead of graph traversals.

Spline-based regression is more sophisticated and effective than kernel regression since it fits non-linear trends instead of computing a simple average. However, the kernel regression idea of constructing models with neighboring responses extends naturally to spline-based regression by using the weighted or regional sampling strategies described in Section 2.1.3. However, distance calculations to identify neighbors in high-dimensional spaces is expensive. Such costs may be justified if more localized models predict responses more accurately.

Neural networks differ from spline-based regression in three broad categories: automation, transparency, and efficiency. As a machine learning technique, neural networks are constructed automatically from training data by using gradient ascent to identify network edge weights that minimize training residuals. In contrast, spline-based regression requires user feedback during correlation and significance testing as well as domain-knowledge to specify predictor interactions. However, spline-based regression produces a more transparent model since users define interactions and the degree of non-linearity in the model's functional form. Neural networks are often treated as a black-box to generate predictions. Lastly, regression models are more

computationally efficient since they are trained by solving linear systems and evaluated with matrix-vector multiplies. In contrast, neural networks are trained with heuristic optimization, such as gradient ascent, and evaluated by traversing the network, which translates into less efficient nested, weighted sums.

2.4 Related Work

Simulation is the technique of choice for evaluating microarchitectural designs and enhancing simulator efficiency has been an active research area. Related work takes one of four broad categories: temporal sampling, parameter significance testing, and design evaluation using empirically or analytically derived models.

2.4.1 Temporal Sampling

Sherwood, *et al.*, propose SimPoint to identify representative instructions [62]. SimPoint identifies phases from a workload, clusters these phases, and takes phases in cluster centroids as representative of the original workload during microarchitectural simulation. By reducing sizes of instruction traces, SimPoint reduces costs per simulation. Wunderlich, *et al.*, propose SMARTS to identify the number of instructions needed for a representative subset of the original workload [70]. The number of samples is chosen to achieve user-specified confidence intervals when estimating design metrics, such as performance. Both SimPoint and SMARTS extract instruction segments from the original trace to capture broader application behavior.

Eeckhout, *et al.*, study statistical profiling for workloads used in microarchitectural simulation [16]. Nussbaum, *et al.* examine similar statistical approaches for

simulating superscalar and symmetric multiprocessors [58]. Oskin, *et al.*, also use statistical profiles to reduce the size of instruction streams input to simulators [59]. These researchers observe detailed simulations for specific benchmarks are not feasible early in the design process. Instead, profiling produces relevant program characteristics, such as instruction mix and data dependencies between instructions. A smaller synthetic benchmark then replicates these characteristics.

Introducing sampling and statistics into simulation reduces accuracy in return for gains in speed and tractability. While researchers in instruction sampling and synthetic benchmarks suggest this trade-off for simulator inputs (*i.e.*, workloads), we propose this trade-off for simulator outputs (*i.e.*, performance and power results). As observed in Section 2.1.1, temporal and spatial sampling should be applied jointly to reduce costs per simulation and number of simulations, respectively.

2.4.2 Parameter Significance Testing

Yi, *et al.*, identify critical, statistically significant microarchitectural design parameters using Plackett-Burman matrices to design optimal multi-factorial experiments [71]. They suggest fixing all non-critical parameters to reasonable constants and performing extensive simulations that sweep a range of values for the critical parameters. By designing experiments more intelligently, designers use simulations more effectively and reveal more about the design space. Similarly, we identify statistically significant design parameters with clustering, association, and correlation analyses, using these parameters to construct regression models. Instead of further simulation, we rely on regression models to explore the design space.

Joseph, *et al.*, derive performance regression models using stepwise regression, an automatic and iterative approach to adding and dropping terms from a model depending on measures of significance [34]. They use these models for significance testing only and do not actually predict performance. Although commonly used, stepwise regression has several problems cited by Harrell [25]: (1) R^2 values are biased high, (2) standard errors of regression coefficients are biased low leading to falsely narrow confidence intervals, (3) p-values are too small, and (4) regression coefficients are biased high.

2.4.3 Empirical and Analytical Modeling

Ipek, *et al.*, and Joseph, *et al.*, separately predict microarchitectural performance with artificial neural networks (ANN's) trained by gradient descent and evaluated by nested weighted sums [18, 35]. Ipek, *et al.*, use sigmoid activation functions whereas Joseph, *et al.*, uses radial basis functions with similar accuracy. Dubach, *et al.*, reduce ANN training costs for new, untrained applications by expressing their performance as a linear combination of performance predictions for existing, previously modeled applications [10]. Training the weights in this linear model is less expensive than training completely new application-specific models.

Comparing neural networks and spline-based regression models, we find similar accuracy but also find trade-offs in efficiency and automation [46]. Regression model construction requires more rigorous statistical analysis while neural network construction is automated; the network is often treated as a black box. Regression models are likely more computationally efficient than neural networks. Regression models

are constructed by solving linear systems and evaluated by multiplying matrices and vectors. In contrast, neural networks are constructed with gradient ascent and evaluated with nested weighted sums as data propagates through multi-layer networks. Differences in computational efficiency are essentially differences in numerical linear algebra for regression and heuristics for neural networks.

In contrast to empirical models, analytical models capture first-order design trends by encapsulating designers' prior intuition and understanding of the design space. Hartstein, *et al.*, present a first-order model for analyzing pipeline depth that illustrates opposing design trends: greater instruction-level parallelism decreases the optimal depth while fewer pipeline stalls increases the optimal depth [26]. Noonburg, *et al.*, measure application parallelism with trace-driven simulation [57]. These measures of parallelism are combined with analytical expressions of microarchitectural capabilities to estimate performance. Karkhanis, *et al.*, construct analytical models to estimate performance by penalizing idealized steady-state performance with miss events from the branch predictor or cache hierarchy measured with fast, functional simulation [36].

We construct empirical models for integrated functional and performance simulators. In contrast, analytical models separate functionality and performance, typically measuring program characteristics using fast trace-driven functional simulations. Analytical models use these characteristics to quickly estimate performance. Separating functional simulation and performance models improves the speed of performance estimation. However, despite simplifying assumptions, analytical models are often difficult to construct given the complexity of modern superscalar, out-of-order pipelines.

In contrast, models constructed empirically, with statistical inference or machine learning, can capture the complexity encapsulated by detailed, cycle-accurate simulators of modern designs. Furthermore, empirical models are more likely to scale with complexity as designers consider design domains with less mature intuition.

2.5 Summary

This chapter details our approach to statistical inference, a core component of a simulation paradigm that reduces the number of simulations for design space exploration. Empirically constructed regression models are comprehensive, sparsely trained, accurate, and efficient.

Statistical inference is comprehensive and effectively captures performance and power trends across large design spaces. Although this chapter describes models constructed with 4,000 samples from a billion-point space, the methodology extends to spaces with up to 240 billion designs and 500 simulated samples (Table B.4). Such a design space includes all the standard parameters of out-of-order, superscalar architectures. This scalability provides a holistic view of the design space. However, these models are not intended to replace simulators. As regression models reveal interesting regions in the design space, further simulation may be required within these regions to expose further detail or to construct a refined regional model.

Comprehensive regression models are inexpensive to construct. This chapter discusses 4,000 samples from a space of nearly one billion designs, but we find 500 simulations sufficient (effectively one simulation for every two million designs). Costs drop to one simulation for every 500 million designs for our space of 15 parameters

Figure 2.9: **Model Error Distributions**. Boxplots capture error distributions for performance (L) and power (R) predictions on validation set of 100 random designs. Example for design space of Table B.4 and benchmarks of Table 2.2.

and 240 billion points. Since regression models are constructed once and used repeatedly, the costs of simulating these sparse samples are quickly amortized over a broad range of design analyses and optimizations. Furthermore, these samples are collected uniformly at random, minimizing user effort in configuring simulations.

Despite the sparsely simulated training samples, models are accurate. We illustrate median errors between 5 and 7 percent for our space of one billion points and observe comparable errors for our space of 240 billion points (Figure 2.9). Such accuracy is sufficient to guide designers to interesting regions of the design space. Furthermore, these errors are typically much smaller than estimated performance improvements and/or power savings in early stage design optimization.

Lastly, regression models are efficient. Users evaluate regression equations for performance and power estimates. After non-linear transformations, these estimates are expressed as matrix-vector multiplication, which is capable of leveraging highly optimized numerical linear algebra routines. This efficiency translates into thousand's

of performance or power estimates per second. In contrast, cycle-accurate simulation time for a single design is measured in tens of minutes. The rest of this dissertation leverages this computational efficiency to enable qualitatively new capabilities in design analysis and optimization.

Chapter 3

Characterizing Performance and Power Topologies

Contents

Microarchitectural design characterization is often insufficient and ad hoc due to the significant computational cost of modern simulator infrastructure. Simultaneously, robust characterization is increasingly important as performance is rarely considered in isolation and the inclusion of power metrics introduces non-monotonicities and interesting design compromises. Robust characterization must take a holistic view of the space, guiding the designer to performance and power efficient regions. Revealing efficiency trends and exposing bottlenecks, high-level characterization is a necessary prerequisite to effective optimization. After finding regions of interest, designers might further refine the analysis to extract additional detail or accuracy.

Leveraging the efficiency of inferential models, we may consider comprehensive design spaces several orders of magnitude larger than those tractable in detailed simulation. Instead of simulating a few hundred designs, we may now evaluate inferential models for hundreds of thousands of designs. This more complete understanding is critical as Moore's Law provides increasingly abundant microarchitectural resources and designers must use these resources to deliver performance in a power efficient manner. Furthermore, this strategy addresses fundamental limitations in current approaches to sensitivity analysis. Instead of constraining studies and shifting bottlenecks from one parameter to another parameter outside the scope of study, we consider the entire design space and effectively remove bottlenecks.

We evaluate performance in billion's of instructions per second (*bips*) and power

in Watts (w). Efficiency is typically measured in $bips/w$. We also consider $bips^3/w$, a more rigorous alternative, equivalent to the inverse energy delay-squared product [6, 23]. This voltage invariant metric is derived from the cubic relationship between power and voltage-frequency (V, f). Since $w \propto V^2 f$ and $V \propto f$, $w \propto f^3$ and $f^3/w \propto k_0$ where k_0 is some constant. If $f \propto bips$, then $bips^3/w \propto k_1$ where k_1 is some constant and the metric is invariant as voltage and frequency change. A $bips^3/w$ maximizing design is optimal regardless of any voltage and frequency scaling. Informally, this metric emphasizes performance over power such that efficiency gains from a percentage increase in performance will be greater than those from a percentage decrease in power.

This chapter considers a series of increasingly sophisticated characterization techniques. Each technique evaluates regression models to estimate performance and power. These estimates are combined to consider efficiency in the following design space analyses:

- **Pareto Frontiers:** We comprehensively characterize a design space, constructing a regression-predicted pareto frontier in the power-delay space. This frontier consists of designs that minimize delay for a given power budget or minimize power for a given delay target. (Section 3.2)

- **Contours for Visualizing Topologies:** We construct contour maps to visualize the design topology, surveying the practical applications of these maps in bottleneck analysis and workload characterization. (Section 3.3)

- **Metrics for Quantifying Roughness:** We define and compute metrics to quantify the roughness of performance and power topologies. We examine

the link between roughness and contour maps, validating roughness metrics graphically by ensuring contours observed with greater non-linearity or non-monotonicity are quantified rougher. (Section 3.4)

These techniques further designer understanding of the superscalar, out-of-order microarchitecture. More importantly, they establish a rigorous foundation for characterizing less intuitive design spaces as we consider emerging design domains with significant design and metric diversity.

3.1 Parameter Sensitivity

Before considering more comprehensive techniques for design space characterization, we first summarize the current approach to assessing parameter sensitivity. Designers typically take a parameter of interest and sweep a range of values in simulation while fixing all other parameters to a constant baseline value. Such constrained studies may simply shift bottlenecks from the parameter of interest to another parameter outside the scope of study, thereby producing results that may not generalize to the broader design space. Furthermore, the constrained scope of study will limit the discovered performance gains or power savings.

3.1.1 Pitfalls of One-Dimensional Sensitivity

The microarchitectural design space is defined by tunable parameters that impact performance and power. The compromise between these metrics for a given parameter X_i may be expressed as its sensitivity:

$$S_{Xi}(x) = \left| \left(\frac{\delta Perf / \delta X_i}{Perf} \right) \times \left(\frac{\delta Power / \delta X_i}{Power} \right)^{-1} \right|_{x=(\tilde{x}_1,\ldots,\tilde{x}_p)} \tag{3.1}$$

where $Perf$, $Power$, and their partial derivatives are evaluated at a particular design point $x = (\tilde{x}_1, \ldots, \tilde{x}_p)$. Sensitivity is the absolute magnitude of the percentage change in performance for a percentage change in power. Sensitivities for an optimized design should be balanced so that marginal power costs of performance from all tunable parameters are equal [49, 75]. Computing these derivatives with respect to each parameter, designers identify each parameter's performance and power trade-off at a particular design point. High sensitivity indicates parameters from which significant performance gains are possible with modest power costs. Sensitivities must be recomputed after optimizing a parameter since relative sensitivities likely change from point to point in the space. Thus, sensitivity is a metric for ranking parameter significance at any given design.

Figure 3.1 presents parameter sensitivities for ammp and mcf evaluated at a design point resembling the IBM POWER4 (Table 3.1). Ammp performs well at the baseline and there are few opportunities to further tune performance. Observing superscalar width is most sensitive and L1 cache sizes are least sensitive, any additional tuning should first enhance superscalar width and then re-assess sensitivity. In contrast, sensitivities for mcf indicate opportunities to tune the cache hierarchy by increasing L1 data cache size. Intuitively, this analysis suggests the L1 data cache provides a much greater performance benefit for every percentage change in power cost while large L2 caches deliver performance in a relatively power inefficient manner.

One-dimensional optimization simply shifts sensitivities and efficient tuning should

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

Table 3.1: **POWER4 Baseline**. Superscalar, out-of-order microarchitectural design resembling the IBM POWER4.

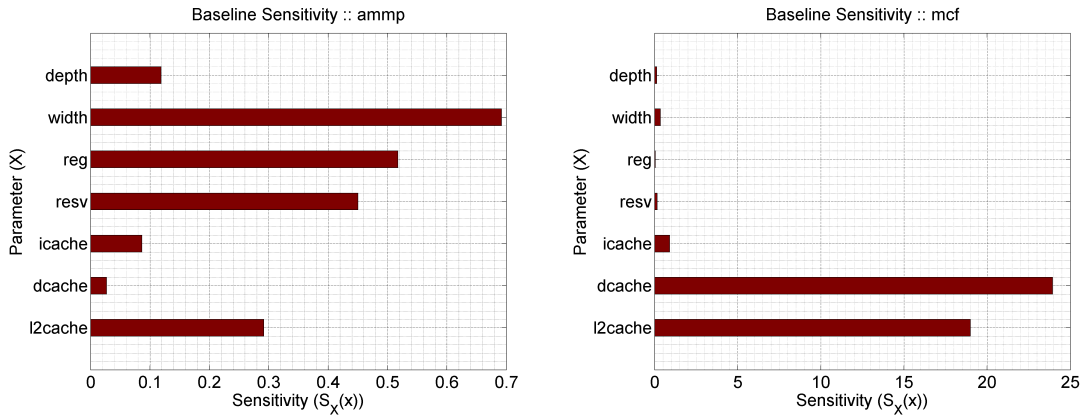

Figure 3.1: **Sensitivity at Baseline**. Parameter sensitivity for ammp (L) and mcf (R) computed at a baseline design resembling the IBM POWER4 design (Table 3.1).

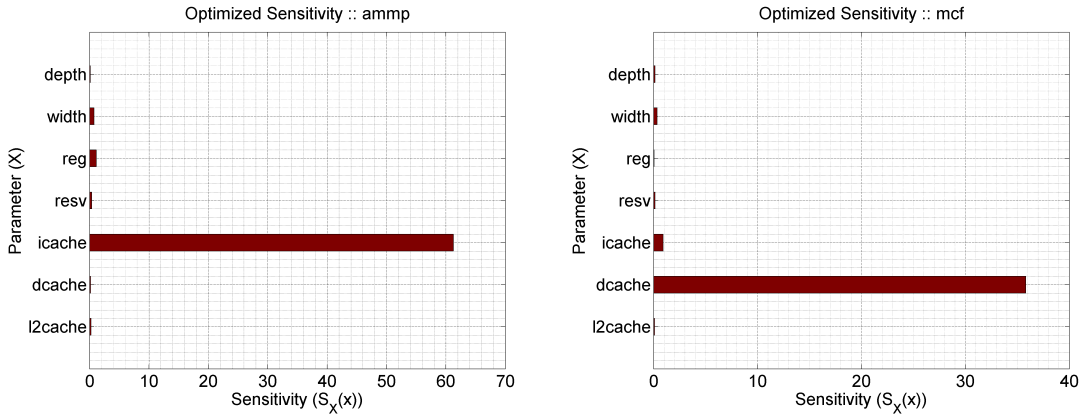

Figure 3.2: **Sensitivity after Single Parameter Optimization**. Parameter sensitivity for ammp (L) and mcf (R) computed after optimizing the most sensitive parameters, superscalar width for ammp and L1 data cache for mcf.

occur at higher dimensions by considering all design parameters simultaneously. Figure 3.2 shows sensitivities of designs after tuning the most sensitive parameters for ammp and mcf. For ammp, we increase issue bandwidth from 4 to 8 since superscalar width is most sensitive. However, all non-width parameters remain constant at POWER4-like values leading to a new bottleneck in the L1 instruction cache. Furthermore, this width optimization is ineffective and generates a net $bips/w$ efficiency loss, increasing performance and power by 40.7 and 62.8 percent, respectively. A similar optimization for mcf illustrates that a doubling of L1 data cache size from 32 to 64 KB relieves L2 cache sensitivity, but has no appreciable impact on performance and power efficiency. Thus, one-dimensional optimization simply shifts sensitivities and tuning must occur at higher dimensions, considering all design parameters simultaneously. This point is well understood by designers in principle, but single parameter sensitivity has become the status quo owing to the computational costs of detailed simulation for multi-dimensional alternatives.

	Set	Parameters	Measure	Range	$\|S_i\|$
S_1	Depth	depth	FO4	9::3::36	10
S_2	Width	width	issue b/w	2,4,8	3
		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	KB	16::2x::256	5
S_6	D-L1 Cache	d-L1 sache size	KB	8::2x::128	5
S_7	L2 Cache	L2 cache size	MB	0.25::2x::4	5

Table 3.2: **Design Space II**. Used for design characterization and optimization where regression models are evaluated exhaustively for every point in the space. $p = 7$, $|S| = 3.8\text{E+}5$.

3.1.2 Case Study of Pipeline Depth

Pipeline depth is a particularly important design parameter. Parameter studies of depth convinced the microprocessor industry to moderate pipelining; deep pipelines were inefficient, delivering performance with high power costs [26, 28, 74]. However, prior pipeline studies consider various depths while holding most other design parameters at constant values to avoid the simulation costs of varying multiple parameters simultaneously. Thus constraining the space may lead to narrowly defined studies with conclusions that may not generalize. Regression models enable a more complete characterization of pipeline depth by varying all parameters simultaneously. A more comprehensive depth analysis ensures observed trends are not an artifact of the constant baseline values to which other parameters are held.

SPEC CPU 2000	
ammp	Simulates molecular dynamics
applu	Solves parabolic/elliptic partial differential equations (PDE's)
equake	Simulates seismic wave propagation
gcc	Compiles C programs
gzip	Performs compression
mcf	Performs combinatorial optimization
mesa	Provides 3-D graphics library support
twolf	Simulates circuit place and route
SPEC JBB 2000	
jbb	3-tier Java business server

Table 3.3: **Benchmarks**.

Pipeline depth is specified by the number of fan-out-of-four (FO4) inverter delays per pipeline stage. FO4 delay is defined as the delay of one inverter driving four copies of an equally sized inverter. When logic per pipeline stage is measured in terms of FO4 delay, deeper pipelines have smaller FO4 delays. We first apply the approach of earlier studies as a reference and then enhance these studies by allowing all parameters to vary simultaneously. Specifically, for an average of the nine SPEC benchmarks in Table 3.3, we compare and contrast the following approaches:

- **Original Analysis:** Consider the POWER4-like baseline architecture of Table 3.1, predicting $bips^3/w$ efficiency as depth varies and all other design parameters are held constant at baseline values.

- **Enhanced Analysis:** Consider the comprehensive design space of Table 3.2, predicting $bips^3/w$ efficiency as all parameters vary simultaneously.

The line plot of Figure 3.3 presents predicted efficiency relative to the $bips^3/w$ maximizing baseline design in the constrained original analysis. This analysis sweeps

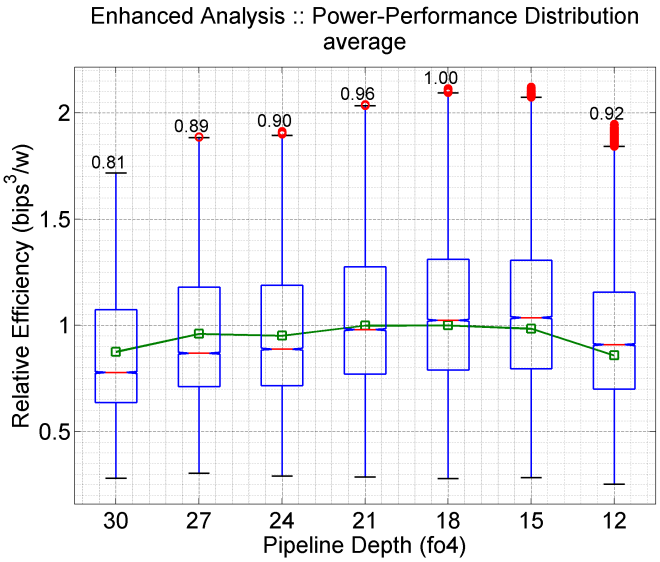

Figure 3.3: **Pipeline Depth Analysis**. $bips^3/w$ for original (line plot) and enhanced (boxplots) analyses. Efficiency relative to $bips^3/w$ optimum in original analysis at 18 FO4.

pipeline depth holding all other parameters fixed. 18 FO4 delays per stage is optimal for an average of the benchmark suite and efficiency is normalized to this optimum in Figure 3.3. Although choosing the deepest (12 FO4) or shallowest (36 FO4) pipeline will achieve only 85.9 or 87.6 percent of this optimal efficiency, respectively, the models suggest a plateau around the optimum and not a sharp peak.

The superimposed boxplots of Figure 3.3 show the $bips^3/w$ distribution of all the designs that implement each pipeline depth in the enhanced analysis. In a space of 375,000 points and ten pipeline depths, there are 37,500 designs represented in each boxplot. From the $bips^3/w$ quartiles, the boxplot for 18 FO4 indicate 75, 50, and 25 percent of these designs achieve efficiency of at least 79, 102, and 131 percent of the original $bips^3/w$ optimum. The maxima of these boxplots constitute a potential bound on $bips^3/w$ efficiency achievable in this design space with up to 2.1x improvements at

the optimal 18 FO4 pipeline depth. These bounding architectures are characterized by wide pipelines as well as larger queue and register file sizes. Thus, higher efficiency is achieved when width and register file sizes vary with depth.

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

Predicted efficiency penalties for sub-optimal depths are also more significant for the bounding architectures. In the original analysis, the $bips^3/w$ maximizing depth is 15-18 FO4 and the sub-optimal 30 FO4 design achieves 88 percent of the optimal efficiency, incurring a 12 percent efficiency penalty as shown by the line plot. The numbers above each boxplot in Figure 3.3 quantify each bounding architecture's efficiency relative to that of the $bips^3/w$ maximizing bound architecture at 18 FO4. While the bounding architectures are also most efficient at 15 to 18 FO4 in the enhanced analysis, the sub-optimal 30 FO4 design achieves only 81 percent of the optimal efficiency and incurs a 19 percent penalty (greater than the 12 percent penalty in the original analysis). This trend is observed for all depths shallower than the optimal 18 FO4. Since bounding architectures are characterized by wider pipelines, choice of depth

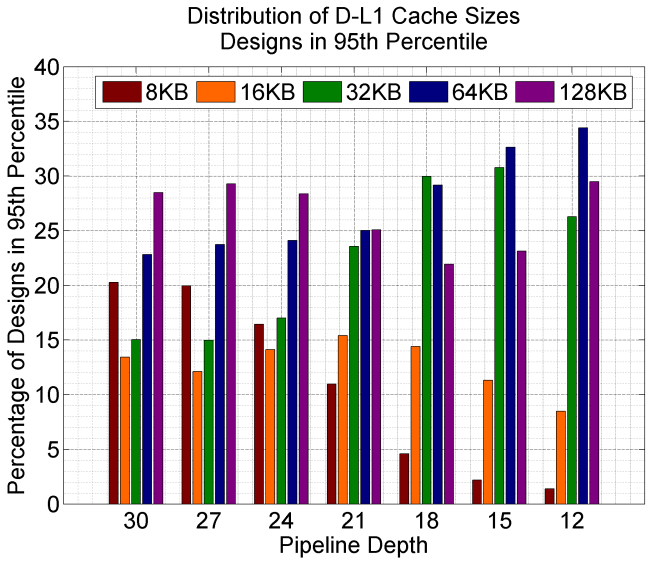

Figure 3.4: **Pipeline and Cache Sizes**. Distribution of d-L1 cache sizes for designs in 95th percentile.

becomes more significant and sub-optimal depths incur greater efficiency penalties. For the average across our benchmark suite, wide pipelines with shallow depths result in greater design imbalances and power-performance inefficiencies.

Figure 3.4 presents the distribution of L1 data cache sizes in the most efficient designs at each depth. In particular, we take the 37,500 designs at each depth and consider designs in the 95-th percentile (*i.e.*, 1,875 designs in the top 5 percent of each depth's boxplot). Small 8 KB data caches are observed for 20.3 percent of top designs at 30 FO4 while such caches are optimal for only 1.4 percent of top designs at 12 FO4. The percentage of top designs with larger 64 KB caches increases from 22.8 to 34.4 percent with deeper pipelines. Thus, small caches are increasingly viable at shallow pipelines while top designs often have large caches at deep pipelines, confirming our intuition that deeper pipelines favor larger caches to mitigate the

increased costs of cache misses. Given these interactions, cache design parameters should vary simultaneously with pipeline depth.

As shown in the discussion of one-dimensional sensitivity and the case study for pipeline depth, efficiency is maximized when we simultaneously consider multiple parameters. As illustrated in our case study, our regression models enable this more holistic view of parameter sensitivity. Although sensitivity is the standard approach to performance and power analysis, regression models also enable more sophisticated analysis of these metrics.

3.2 Pareto Frontiers

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

Regression models enable a complete characterization of the microarchitectural design space. In particular, we exhaustively evaluate a design space containing 375,000 points (Table 3.2). Such a characterization reveals all trade-offs between a large number of design parameters simultaneously compared to an approach that relies on per parameter sensitivity analysis. Given this characterization, we construct Pareto frontiers. While we cannot explicitly validate the regression-predicted Pareto frontier

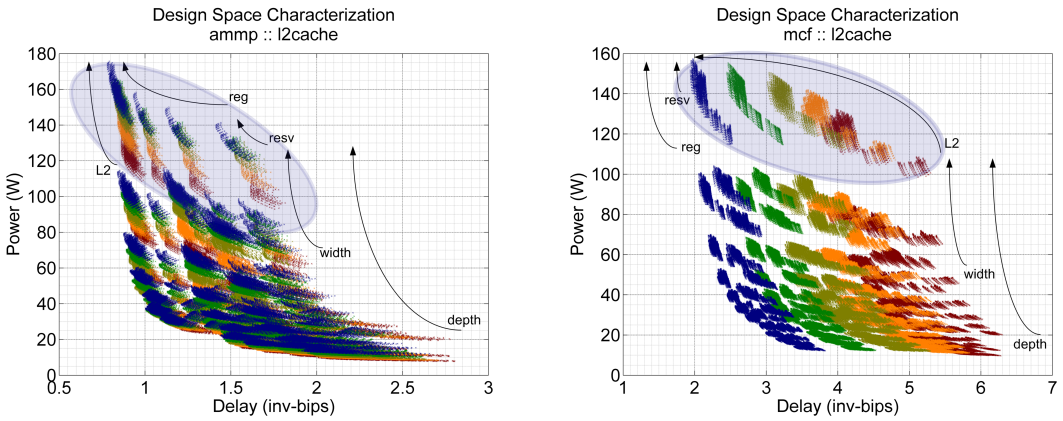

Figure 3.5: **Design Characterization**. Regression-predicted delay, power for 375,000 designs of Table 3.2 running representative SPEC benchmarks ammp (L) and mcf (R). Arrows indicate trends as parameter values change. Colors map to L2 cache sizes.

against a hypothetical frontier found by exhaustive simulation, the former is likely close to the latter given the accuracy observed in validation.

3.2.1 Characterizing the Design Space

Figure 3.5 plots predicted delay (inverse performance) and power by exhaustively evaluating the regression models for representative benchmarks. The design space is characterized by several overlapping clusters of similar designs. Each cluster contains designs with a particular pipeline depth-width combination. For example, the shaded mcf cluster is comprised of designs with depths of 12 FO4 and widths of 8 functional units. This cluster minimizes delay at the greatest power cost with delays ranging from 1.9 to 5.3 seconds and power ranging from 100 to 160 watts.

The arrows of Figure 3.5 identify power-delay trends as a particular resource size increases. Consider the shaded 12 FO4, 8-wide design clusters for ammp and

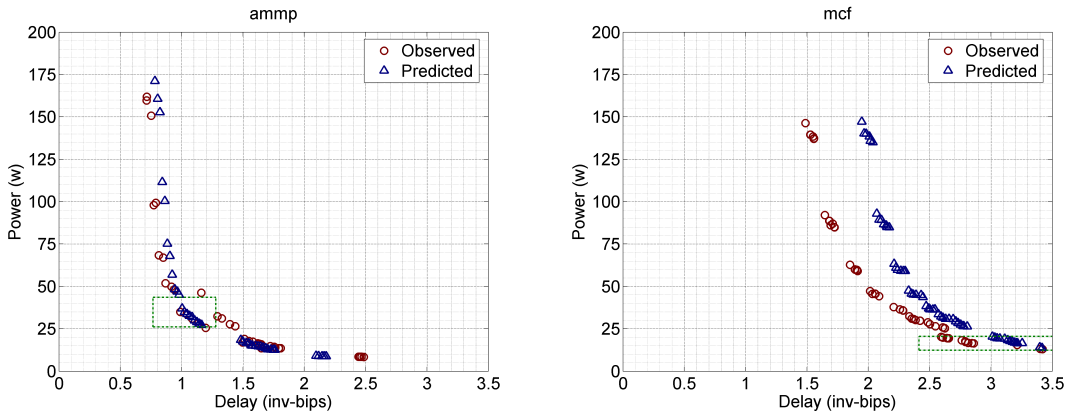

Figure 3.6: **Pareto Frontier**. Pareto optima for 375,000 designs of Table 3.2 running representative SPEC benchmarks ammp (L) and mcf (R). The green boxes illustrate a region within 25 percent of the $bips^3/w$ optimal delay and power from Table 3.4.

mcf. Mcf performance benefits from larger caches with delay shifting from 5.3 to 1.9 seconds as L2 cache size shifts from 0.25 to 4 MB. In contrast, ammp achieves limited performance benefits with delay shifting from 1.0 to 0.8 seconds as L2 cache size increases by the same amount. Ammp also appears to exhibit greater instruction level parallelism, effectively utilizing additional physical registers and reservation stations to reduce delay from approximately 1.8 to 0.8 seconds compared to mcf's reduction of 2.5 to 2.0 seconds.

3.2.2 Identifying the Pareto Frontier

Figure 3.6 plots regression predicted Pareto optima. These optima minimize delay for a given power budget or minimize power for a given delay target. Given exhaustively predicted power and delay characteristics, the frontier is constructed by discretizing the range of delays and identifying the design that minimizes power for each delay in a range of delay targets. These designs are Pareto optimal with respect to the

	Depth	Width	Reg	Resv	I-$ (KB)	D-$ (KB)	L2-$ (MB)	Delay Model	Power Model
ammp	27	8	130	12	32	128	2	1.0	35.9
applu	27	8	130	15	16	8	0.25	0.8	39.6
equake	27	8	130	15	64	8	0.25	1.2	41.5
gcc	15	2	70	9	16	8	1	1.2	44.1
gzip	15	2	70	6	16	8	0.25	0.8	24.2
jbb	15	8	80	12	16	128	1	0.6	80.9
mcf	30	2	70	6	256	8	4	3.5	12.9
mesa	15	8	80	13	256	32	0.25	0.4	86.9
twolf	27	8	130	15	128	128	2	1.1	34.5

Table 3.4: **Efficient Pareto Optima**. $bips^3/w$ maximizing designs for nine SPEC benchmarks of Table 3.3.

regression models, but may not be the same optima obtained via a hypothetical exhaustive simulation of the space.

Although Pareto optima are useful for particular delay targets or power budgets, not all Pareto optima are efficient with respect to $bips^3/w$, the inverse energy delay-squared product.[1] We compute this efficiency metric for each design on the Pareto frontier and identify the most efficient designs for each benchmark in Table 3.4. The $bips^3/w$ optimal design for ammp is located at 1.0 seconds and 35.9 watts in the delay-power space, the knee of the Pareto optimal curve. Similarly, the mcf $bips^3/w$ optimum is located at 3.5 seconds and 12.9 watts. Overall, these optima are drawn from diverse design regions motivating comprehensive space exploration. Although Table 3.4 indicates these optima occupy very different parts of the design space, they reside in very similar regions of the power-delay space. Most of the optima are located between 0.5 and 1.5 seconds, 25 and 50 watts.

[1] $bips^3/w$ is a voltage invariant power-performance metric derived from the cubic relationship between power and voltage [6, 23].

3.2.3 Validating the Pareto Frontier

Figure 3.6 superimposes regression-predicted Pareto frontiers with simulations of points on those frontiers, suggesting good relative accuracy. Regression effectively captures the delay-power trends of the Pareto frontier. As performance prediction is slightly less accurate than power prediction, however, differences are characterized by horizontal shifts in delay. Performance model accuracy is the limiting factor for more accurate Pareto frontier prediction across all benchmarks in our suite. Mcf highlights these trends, but its absolute performance error is more an exception than a common case. Ammp is more representative of accuracy for the broader benchmark suite.

Figure 3.7 presents error distributions from predicting the performance and power of pareto optima. The median performance error ranges from 4.3 percent (ammp) to 15.6 percent (mcf) with an overall median of 8.7 percent. Similarly, the median power error ranges from 1.4 percent (mcf) to 9.5 percent (applu) with an overall median of 5.5 percent. These error rates are consistent with the median error rates of 7.2 and 5.4 percent observed in the validation of random designs (Figure 2.8), suggesting predictions for Pareto optima are as accurate as those for the overall design space.

In practice, not all Pareto optima are interesting and viable designs. The high power or high delay designs located at the frontier extrema are not particularly interesting due to unfavorable power and delay trade-offs. For the majority of benchmarks, we find our models more accurate for more interesting points near the $bips^3/w$ optimum of Table 3.4. Figure 3.8 presents restricted error distributions when considering only Pareto optima exhibiting delay and power within 25 percent of the $bips^3/w$ optimal delay and power (boxes of Figure 3.6). Comparing complete and restricted error

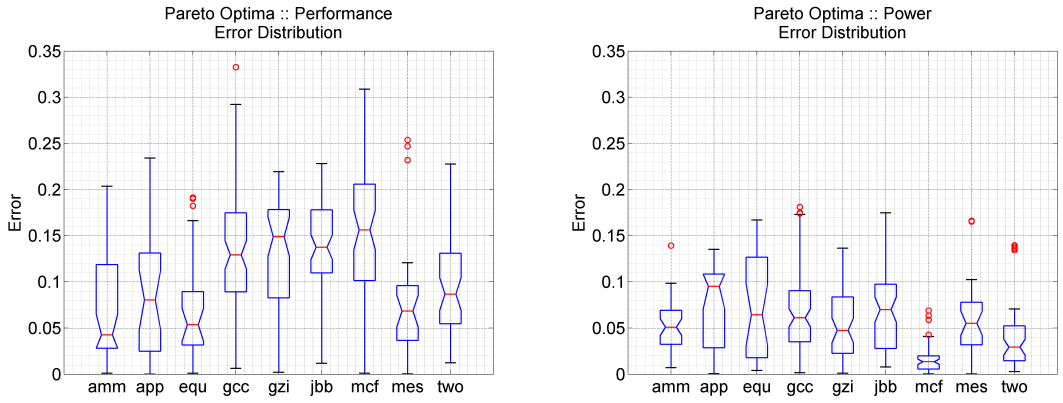

Figure 3.7: **Complete Pareto Frontier Accuracy**. Boxplots capture error distributions for performance (L) and power (R) predictions for complete set of Pareto optima. Pareto frontiers constructed for design space of Table 3.2 and nine SPEC benchmarks of Table 3.3.

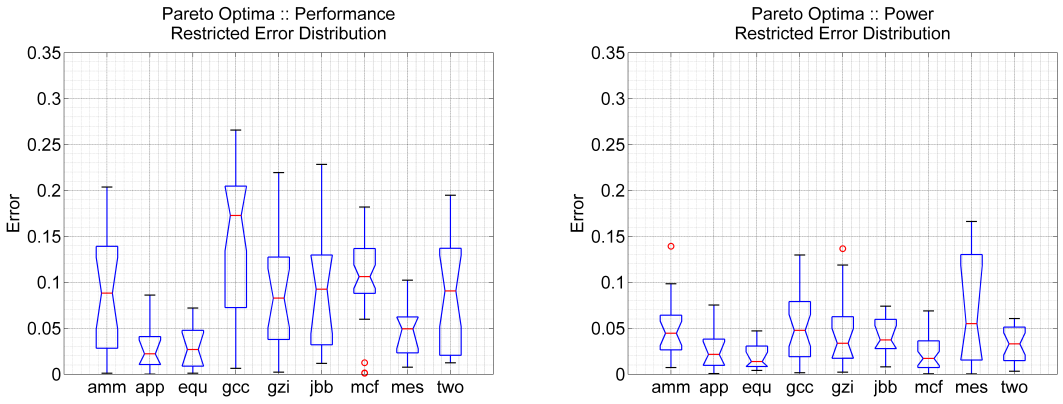

Figure 3.8: **Restricted Pareto Frontier Accuracy**. Boxplots capture error distributions for performance (L) and power (R) predictions on restricted subset of Pareto optima exhibiting delay and power within 25 percent of $bips^3/w$ optima in Table 3.4. Pareto frontiers constructed for design space of Table 3.2 and nine SPEC benchmarks of Table 3.3.

distributions of Figures 3.7–3.8, we find the median and interquartile range decrease for a majority of benchmark errors as we examine only the region around the $bips^3/w$ optimum.

The differing error distributions in Figures 3.7–3.8 motivate future work on hierarchical modeling schemes in which high-level models are first constructed for a comprehensive design space to identify regions of interest around particular optima. Further detail and accuracy may be achieved by performing constrained spatial sampling and constructing localized regression models for regions of interest. Such a scheme overcomes the models' potential regional biases and may further reduce model error as we shift emphases from the complete design space to particular subspaces.

3.3 Contours for Visualizing Topologies

Contour maps enable efficient and comprehensive visualizations of microarchitectural performance and power topologies. A large number of these maps may be quickly generated using performance and power estimates from regression models. Once constructed, contour maps reveal microarchitectural bottlenecks and enable workload comparisons based on microarchitectural resource requirements. After demonstrating these applications of contour maps, we quantify contour roughness and ensure our roughness metrics corroborate graphically observed roughness.

3.3.1 Contour Maps

Contour maps are constructed by exhaustively evaluating regression models for performance and power. Two-dimensional projections of the design space lay the foun-

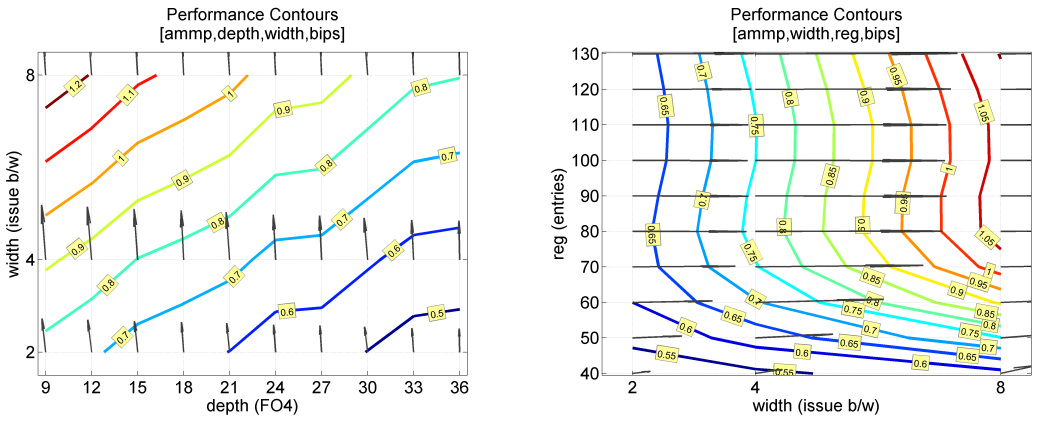

Figure 3.9: **Performance Contours**. Contour maps of SPEC ammp *bips* for depth, width (L) and register file, width (R).

dation for analysis in higher dimensions. These maps illustrate the topology for a metric of interest, revealing non-linearities that may arise from non-monotonic trends (hills, valleys) or diminishing marginal returns from additional resources (plateaus).

The visualization also identifies a path to optimality from any initial design point. Figure 3.9L presents ammp's performance contours for a two-dimensional projection of pipeline depth and superscalar width from the seven-dimensional design space of Table 3.2, revealing a clear path to optimal performance in the direction of deeper, wider pipelines. This particular space favors balanced pipeline dimensions, indicating depth and width should increase together to most effectively deliver performance. While we plot contours that span a continuous range of width values, in practice, we are likely to examine only feasible design points in the contour maps (*e.g.*, 2-, 4-, 8-wide designs).

Gradients produce vector fields that point in the direction of increasing values. We compute these vector fields using numerical gradients and visualize them using

arrows in the contour maps. Furthermore, contour levels closely spaced together imply steeper slopes since smaller microarchitectural changes are required to achieve the same performance or efficiency increase. Thus, closely spaced contour levels result in larger gradient magnitudes. These vector fields complement the contour maps and reveal paths to design optima more quickly.

3.3.2 Bottleneck Analysis

The path to optimality reveals the changing source of bottlenecks for the metric of interest. Figure 3.9R presents ammp's performance contours for a two-dimensional projection of the register file and superscalar width. We observe two distinct regions divided horizontally by the 80-entry register file. The ammp benchmark experiences significant register pressure when running on designs in the lower region of this map. Performance in this lower region is most effectively optimized by increasing both register file sizes and superscalar width. In contrast, we observe diminishing marginal returns in performance from larger register file sizes in the upper region of the map. This is illustrated by vertical contour levels in which changing register file sizes cannot lead to higher performance. The register file is no longer a bottleneck in this region and other design parameters should be analyzed.

Power metrics are easily included into the analysis. Power contours alone are less interesting as power often increases monotonically with increases in microarchitectural resources. Figure 3.10 considers the same projections of Figure 3.9 using a $bips/w$ efficiency metric. Figure 3.10L examines trade-offs for pipeline dimensions, suggesting this design space favors shallower, narrower pipelines once the optimization metric

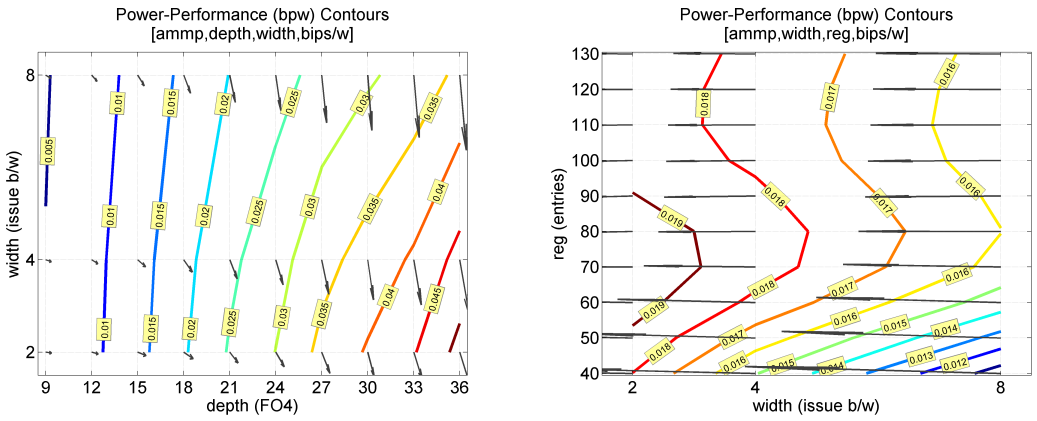

Figure 3.10: **Power-Performance Contours**. Contour maps of SPEC ammp *bips/w* for depth, width (L) and register file, width (R).

accounts for the power costs of deeper, wider pipelines. Pipeline depth is the primary efficiency bottleneck for the more aggressive designs as illustrated by the vertical contour levels in the 9 to 15 FO4 region. As we consider increasingly shallow pipelines, superscalar width begins to limit efficiency and we see a more pronounced trend toward fewer instructions issued per cycle (*i.e.*, issue bandwidth from 8 to 2) in the 24 to 36 FO4 region.

Figure 3.10R illustrates performance and power trade-offs for the register file and superscalar width. The *bips/w* efficiency metric strongly favors narrower pipelines. This metric more clearly identifies the optimal register file size between 60 and 90 entries based on the power costs of over-provisioning this resource. In contrast, the performance analysis of Figure 3.9R favors more than 80 entries and does not capture the power costs that make 130-entry register files unattractive.

The gradients and vector fields are consistent with the high level observations from the contour maps. For example, perfectly vertical or horizontal gradients in

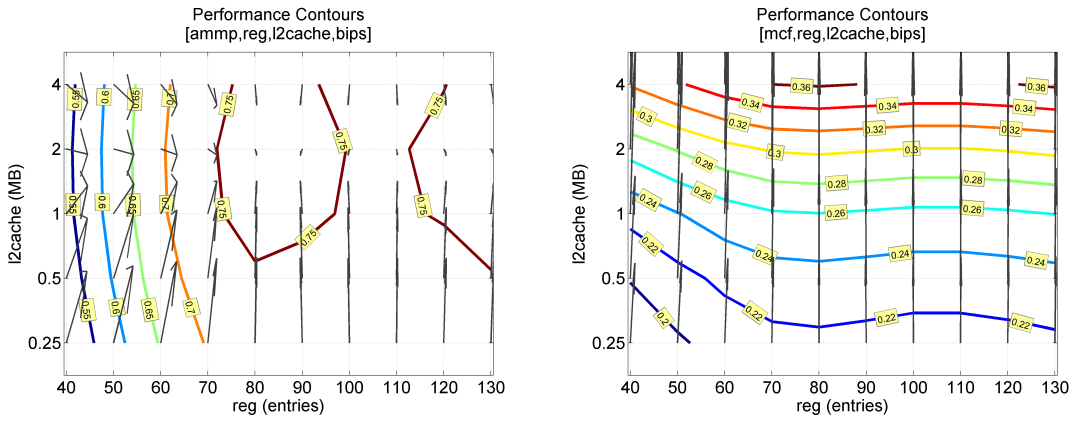

Figure 3.11: **Performance Contours**. Contour maps of SPEC ammp (L) and mcf (R) for L2 cache, register file.

the two dimensions indicate $\delta f/\delta x_1 \approx 0$ or $\delta f/\delta x_2 \approx 0$, respectively. In Figure 3.9R, for example, we observe $\delta Perf/\delta reg \approx 0$ for register files with more than 80 entries, indicating diminished register pressure. In contrast, a large partial derivative $\delta Perf/\delta width >> 0$ indicates significant performance changes as superscalar width varies. The gradients of Figure 3.9L and Figure 3.10L include both vertical and horizontal components, indicating non-zero sensitivities for both parameters in the two-dimensional projection.

3.3.3 Workload Characterization

Bottleneck analysis with contour maps enables workload comparisons based on their microarchitectural resource requirements. Figure 3.11 illustrates such a comparison between ammp and mcf performance by examining demands on the register file and L2 cache. As observed earlier, the register file is a bottleneck for ammp. The vertical contour levels indicate negligible performance advantages from larger caches until

register pressure has been relieved with at least 80 entries. Once the register file contains 80 entries, performance is enhanced by cache sizes larger than 1 MB. In contrast, the horizontal contour levels for mcf reveal significant performance advantages from larger caches, indicating the workload is relatively memory bound. We also observe more subtle indicators that 80-entry register files are optimal for mcf. The vertical dips in the horizontal contour levels all occur at 70 or 80 physical registers, indicating greater performance at this register file size for any L2 cache size.

Microarchitecture independent workload characterizations examine and assess performance through fundamental program characteristics, such as instruction mix and register traffic. However, contour maps enable a more direct analysis based on computational resource requirements. For example, the number of independent instructions within an instruction window is a microarchitecture independent metric to assess instruction level parallelism (ILP) [60]. In contrast, a contour map of pipeline dimensions would assess ILP based on the optimality of various depths and widths, revealing a workload's characteristics and its direct implications for microarchitectural design. Similarly, contour maps of the L2 cache would reveal a workload's memory intensity while exposing implications for cache design.

3.4 Metrics for Quantifying Roughness

The roughness of microarchitectural performance and power topologies has direct implications for regression modeling. Rough topologies require greater model flexibility in the form of additional spline knots. Rough regression models imply an underlying topology at least as rough as the derived model. These effects are most easily

observed in contours where non-linearity and non-monotonicity are evident.

In the context of visualization, roughness metrics quantify the relative roughness between contours. Rough contours imply more challenging inputs to optimization heuristics as hills and valleys in the topology increases the likelihood of heuristics converging to local sub-optima. While contour maps are feasible for low-dimensional analysis, roughness metrics extend to higher dimensions and provide a more complete assessment of a topology. By quantifying topology roughness that would otherwise be assessed subjectively with contour maps, these metrics lay the foundation for robust modeling and optimization in the presence of rough topologies.

We apply the roughness metrics proposed by Green and Silverman for smoothed, non-parametric regression [24]. Equation (3.2) defines a measure of roughness R_1 for a one-dimensional function $f(x)$. This measure of roughness is unaffected by the addition of a constant or linear function since R_1 depends on the second derivative. Furthermore, this definition has a basis in mechanical engineering; if a thin piece of flexible wood is bent to the shape of $f(x)$, the leading term in the strain energy is proportional to R_1. Equation (3.3) is a two-dimensional extension of the one-dimensional definition. Intuitively, R_2 captures roughness since the second derivatives in R_2 are large if the function $f(x_1, x_2)$ exhibits high local curvature. As in R_1, the bending energy of a thin plate is, to first order, proportional to R_2.

$$R_1 = \int_x \left(\frac{\delta^2 f}{\delta x^2}\right)^2 dx \tag{3.2}$$

$$R_2 = \int_{x_2}\int_{x_1} \left\{\left(\frac{\delta^2 f}{\delta x_1^2}\right)^2 + 2\left(\frac{\delta^2 f}{\delta x_1 x_2}\right)^2 + \left(\frac{\delta^2 f}{\delta x_2^2}\right)^2\right\} dx_1 dx_2 \tag{3.3}$$

$$R_d = \int_{x_d}\cdots\int_{x_1} \sum \frac{m!}{v_1!\ldots v_d!}\left(\frac{\delta^m f}{\delta {x_1}^{v_1}\ldots\delta {x_d}^{v_d}}\right)^2 dx_1\ldots dx_d \tag{3.4}$$

Equation (3.4) provides a more general d-dimensional metric based on m-th derivatives, where $2m > d$ [24]. The sum within the integral is over all non-negative integers $v_1, \ldots, v_d$ such that $v_1 + \ldots + v_d = m$. For example, this particular work considers a seven-dimensional design space and computes R_7 based on 4-th derivatives. As in the one- and two-dimensional cases, $R_d = 0$ only if $f(x_1, \ldots, x_d)$ is a polynomial of total degree less than m. Throughout, we report relative roughness rankings since the absolute roughness values are not known to have an intuitive physical interpretation in the microarchitectural context.

3.4.1 Numerical Approximations

Derivatives are approximated using numerical gradients. These approximations use centered differences at the interior and one-sided differences at design space boundaries. A three-point estimate using centered differences computes the slope of a secant line through $(x^* - h, f(x^* - h))$ and $(x^* + h, f(x^* + h))$ as shown by Equation (3.5). Since this approximation is possible only at the interior of the space, the one-sided approximation of Equation (3.6) is needed for derivatives evaluated at the boundaries.

$$\left.\frac{\delta f}{\delta x}\right|_{x=x^*} \approx \frac{f(x^* + h) - f(x^* - h)}{2h} \tag{3.5}$$

$$\left.\frac{\delta f}{\delta x}\right|_{x=x^*} \approx \frac{f(x^* + h) - f(x^*)}{h} \tag{3.6}$$

Integrals are approximated with Riemann sums as shown for one and two dimensions in Equations (3.7)–(3.8). Riemann sums divide the domain of x into n intervals of equal width $\Delta x = \frac{max[x] - min[x]}{n}$ and identify approximation points $x_1^*, x_2^*, \ldots, x_n^*$ such that x_i^* lies in the i-th interval.

$$\int_x f(x)dx \approx \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x \tag{3.7}$$

$$\int_{x_2}\int_{x_1} f(x_1,x_2)dx_1dx_2 \approx \lim_{m,n\to\infty} \sum_{i=1}^{m}\sum_{i=j}^{n} f(x_{1,i,j}^*, x_{2,i,j}^*)\Delta x_1\Delta x_2 \tag{3.8}$$

This approximation is valid if f is a continuous, positive function defined over the domain of x. In the microarchitectural context, f is a regression model for performance, power, or efficiency. The function f is continuous because models are constructed with piecewise cubic polynomials using smooth, continuous knot connections. The function is positive because values for our design metrics are positive.

The approximation points $x_1^*, \ldots, x_n^*$ correspond to the defined resolution of design parameters in our space. The n points for parameter x_i correspond to the number of values in the parameter range of x_i ($|S_i|$ in Tables B.2–B.4). Although f is a regression model that can be evaluated at arbitrarily high resolution, accuracy is constrained by the resolution of the design space used to construct the model. The limits in Equations (3.7)–(3.8) suggest higher resolution design spaces with parameter values observed at finer granularity lead to more accurate approximations.

3.4.2 Roughness and Regression

The functional mapping between designs and metrics would ideally be provided by the simulator. Due to computational costs of extensive simulation, we approximate these functions with regression models formulated from sparsely simulated design space samples. This approximation may smooth non-linear, non-monotonic trends in the design space. Assessing the roughness of the underlying microarchitectural design

topology through regression models captures only high-level roughness since more detailed roughness may be obscured by smoothing in least squares fitting. Thus, we should treat these roughness metrics computed with regression models as conservative estimates of the true design space roughness.

Piecewise cubic splines provide the flexibility needed to capture non-linear trends in the design space. Regression models that most utilize this flexibility for a non-linear design topology are likely trying to capture more difficult trends, resulting in greater modeling error. To assess these effects, Figures 3.12–3.13 plot the nine SPEC benchmarks of Table 3.3 in the error-roughness space for performance and power. We consider median and maximum errors across 100 randomly collected validation points. We then correlate these errors against seven-dimensional roughness. Both error and roughness are expressed relative to their maxima across the nine benchmarks.

Figure 3.12 correlates roughness with median and maximum performance errors, illustrating a positive relationship. For example, ammp and jbb's performance topologies are least and most rough, respectively. Median and maximum jbb errors are 91 and 89 percent greater than those for ammp. The trendline for the nine benchmarks is positive with correlation coefficients of 0.33, 0.38 between roughness and median, maximum errors. Similarly, Figure 3.13 indicates mcf and jbb's power topologies are least and most rough, respectively. Jbb roughness translates into median and maximum errors 58 and 67 percent greater than those for mcf. Power roughness correlations are non-trivial with coefficients of 0.42, 0.48 for median, maximum errors.

Given this relationship between roughness and model accuracy, the model specification may be optimized based on its quantified roughness. A rough model likely

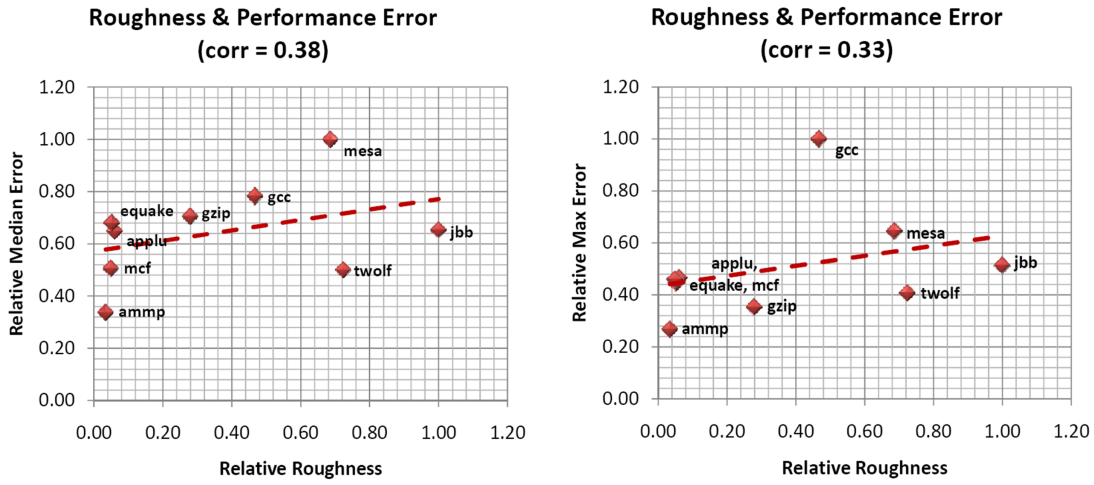

Figure 3.12: **Roughness and Performance Error**. Plots roughness against median (L) and maximum (R) regression errors for performance. Roughness and error are relative to maximum across nine SPEC benchmarks of Table 3.3. Trendlines indicate positive correlations.

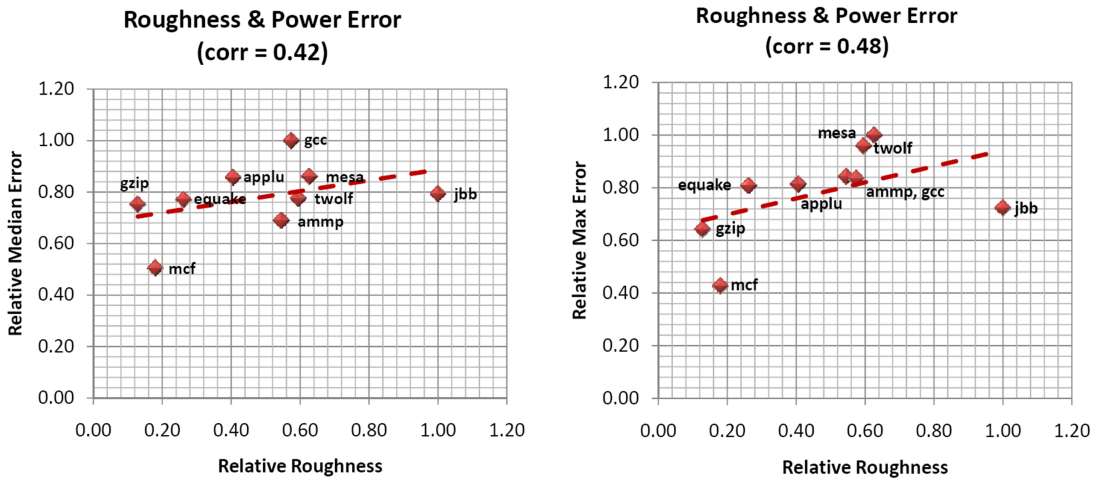

Figure 3.13: **Roughness and Power Error**. Plots roughness against median (L) and maximum (R) regression errors for power. Roughness and error are relative to maximum across nine SPEC benchmarks of Table 3.3. Trendlines indicate positive correlations.

reflects the underlying roughness of the design space. Design space roughness may be more accurately captured by increasing the knot count, thereby increasing the flexibility of the model specification. While the approach of Chapter 2 uses a single model specification across all benchmarks, optimizing models based on topology roughness requires application-specific knot counts.

3.4.3 Roughness and Contours

Given the computational speed of regression models, designers can quickly generate a large number of contour maps. However, designers will also need strategies to sift through this plethora of visualized data. Prior work demonstrated analysis with microarchitectural contours, but did not provide any objective mechanism for focusing designer attention [59]. In contrast, we propose metrics to identify interesting contours. We assess the effectiveness of our roughness metrics by computing R_2 for contour maps and validating against graphically observed roughness.

This analysis reveals two contributors to larger roughness values: range and variability. Range is the difference between minimum and maximum metric values in a given contour plot. Variability is curvature in the topology, manifested in contour non-linearity or non-monotonicity. Empirically, we observe a significant range component in the roughness metrics. For example, we find R_2(Figure 3.10L) $>$ R_2(Figure 3.10R) and R_2(Figure 3.11R) $>$ R_2(Figure 3.11L) due to range differences.

For a more thorough analysis, we construct all two-dimensional contours from the seven-dimensional design space and rank them by R_2 roughness. In a seven-dimensional space, each benchmark is visualized using $\binom{7}{2} = 21$ contours. For each

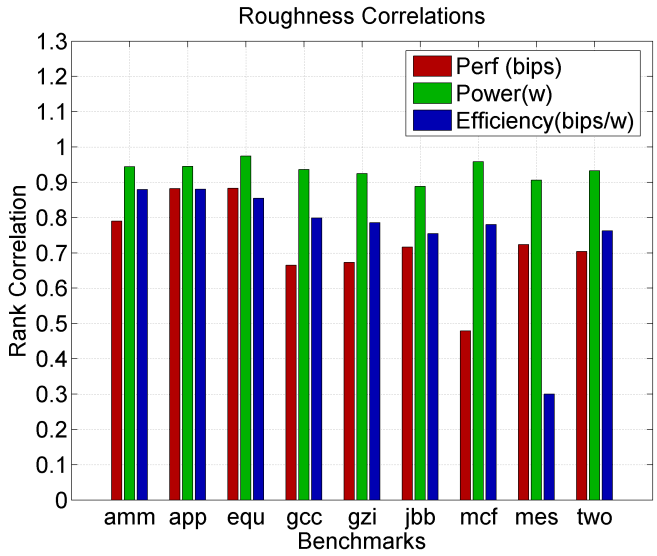

Figure 3.14: **Contour Roughness-Range Correlation**. Correlation between topology roughness and range of performance, power, and *bips*/*w* efficiency values in contour maps. Range is computed by dividing the maximum contour value by the minimum contour value.

benchmark, we correlate the roughness and range of the 21 contours. Figure 3.14 illustrates large correlation coefficients (most exceeding 0.65), suggesting a strong relationship between the range and roughness of contour maps for various benchmarks and design metrics.

We isolate contributions from variability by standardizing all contours to comparable ranges. We subtract the mean and divide by the standard deviation to produce similar ranges primarily between -3.0 and 3.0, the dominant range of a standard Normal distribution. Figure 3.15 presents standardized *bips*/*w* contours of varying roughness from the $\binom{7}{2} = 21$ possible two-dimensional contours. We observe qualitative differences in non-linearity and non-monotonicity as we progress from most rough to least rough contours. These trends are representative of two-dimensional contours for

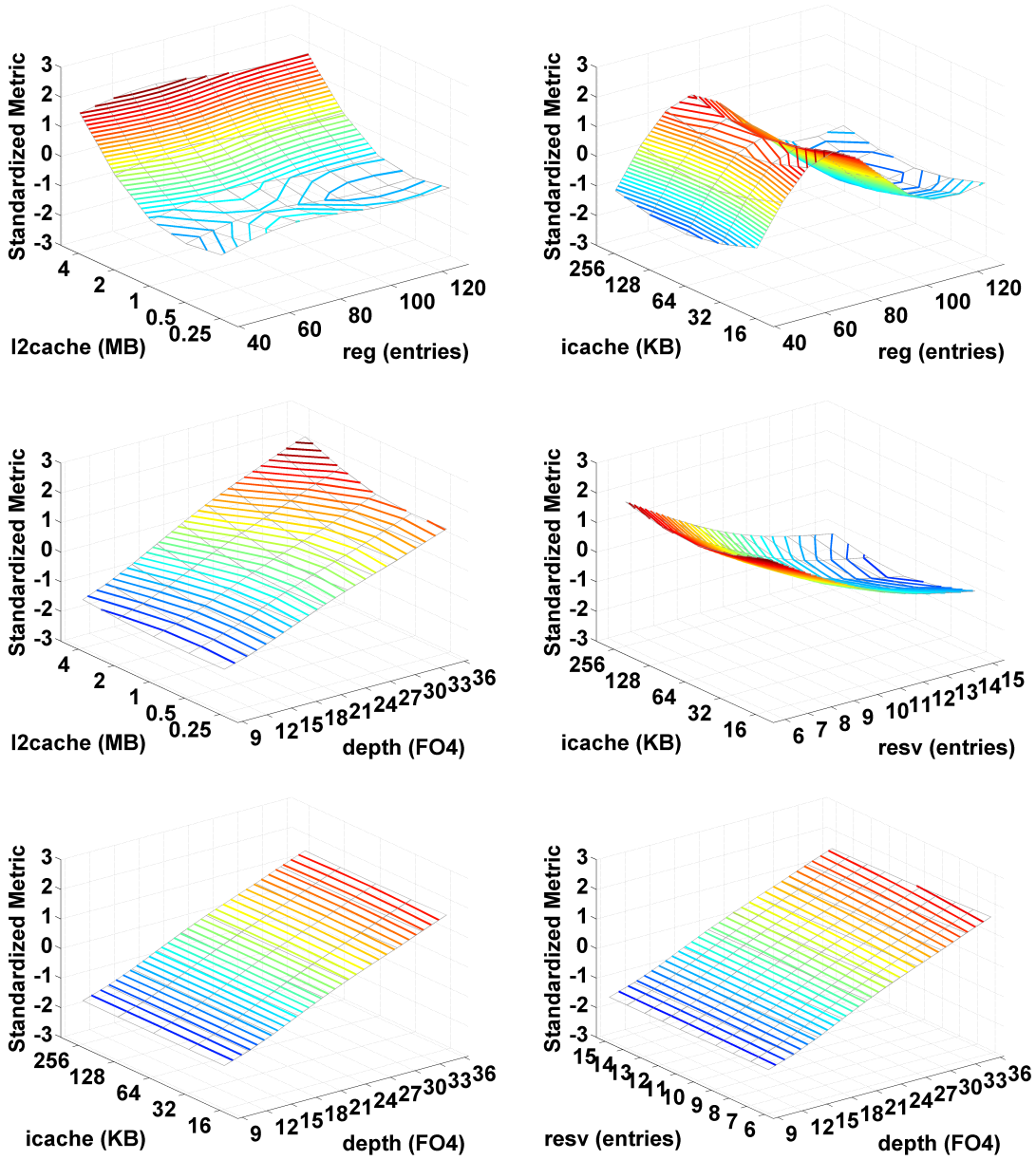

Figure 3.15: **Roughness and Observed Contour Variability**. Standardized *bips*/*w* contours for *mcf*. Ranking the $\binom{7}{2}$ contours in order of decreasing roughness, we present contours ranked most rough (1st, 2nd of 21), moderately rough (10th, 11th of 21) and least rough (20th, 21st of 21) from left to right, top to bottom.

other workloads, with rankings corroborating graphically observed roughness. Hills, valleys, and plateaus are frequently observed for rough contours while regular vector fields and vertical, horizontal contour levels dominate smooth contours.

This comparison of roughness metrics to observed topologies in the easily visualized two-dimensional contours qualitatively validates our measures of non-linearity. Contours with high roughness rankings do indeed appear rough. We thus build confidence in the proposed metrics, especially when applying them to high-dimensional topologies that cannot be effectively visualized. Thus, roughness metrics enable quantitative and objective comparisons of contour range and variability.

This quantitative assessment of roughness eliminates much of the subjectivity in contour analysis and provide a rigorous mechanism for focusing designer attention on the most significant parameters and interesting topologies. This focus is especially important since contours are often considered for low k-dimensional projections of a p-dimensional space. When p is large and k is small (typically less than three), the number of possible contour maps $\binom{p}{k} = \frac{p!}{(p-k)!k!}$ increases rapidly. While regression models can be used to quickly construct these contours, such contours are useful only if designers can sift through the plethora of data to find rough topologies that expose interesting design compromises. Thus, roughness metrics must accompany contour maps for tractable analysis and interpretation.

3.5 Related Work

Design space characterization is a necessary first step prior to performance and power optimization. We compare our approach to comprehensive analysis against related

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

3.5.1 Sensitivity

Zyuban, *et al.*, measures hardware and voltage intensity to quantify compromises between energy and delay from circuit-level tuning and voltage scaling, respectively [75]. Intensity is computed as $\frac{D}{\delta D}\frac{\delta E}{E}$ where D is delay and E is energy. Zyuban uses these intensity metrics to derive conditions for optimal microarchitectural power-performance from mathematical relations, but do not compute the needed gradients. Our proposed regression models provide a mechanism for computing these gradients. Instead of implementing symbolically derived optimality conditions, we would optimize with heuristics using empirically derived regression models as objective functions.

Markovic, *et al.*, further the work by Zyuban, *et al.*, by deriving sensitivity $\frac{\delta E/\delta X}{\delta D/\delta X}$ for tunable circuit parameters X such as gate sizing, supply voltage, and threshold voltage [49]. Markovic, *et al.*, show optimal values for the circuit parameters are those that equalize sensitivity. Sensitivity is equalized by jointly optimizing registers and logic within microarchitectural blocks (*e.g.*, arithmetic-logic units). In contrast to this circuit-level emphasis, we consider high-level interactions across a wide range of microarchitectural blocks and cache structures. Furthermore, they calculate the needed gradients from analytical circuit equations and simulations while we illustrate the feasibility of analogous studies at the microarchitectural and macro block level using statistical inference.

3.5.2 Optimizing Pipeline Depth

Most prior work in optimizing pipeline depth focuses exclusively on improving performance. Kunkel, *et al.*, demonstrate that vector code performance is optimized on deeper pipelines while scalar codes perform better on shallower pipelines [41]. Dubey, *et al.*, develop a more general analytical pipeline model to show that the optimal pipeline depth decreases with increasing overhead from partitioning logic between pipeline stages [15].

Prior work also finds optimal pipeline depths from simulation. In particular, Hartstein, *et al.*, perform detailed simulations of a four-way superscalar, out-of-order microprocessor with a memory execute pipeline to identify a 10.7 FO4 performance optimal pipeline design for the SPEC2000 benchmarks [26]. Similarly, Hrishkesh, *et al.*, perform simulations for an Alpha 21264-like machine to identify 8 FO4 as a performance optimal design running the SPEC2000 benchmarks [28]. Zyuban, *et al.*, find 18 FO4 delays the power-performance optimal pipeline design point for a single-threaded microprocessor [74].

3.5.3 Roughness Metrics

Green, *et al.*, propose roughness metrics to penalize the least squares fit for spline-based regression [24]. For example, a roughness term may be added to the sum of square errors minimized in least squares. Accounting for roughness when fitting regression coefficients, this roughness penalty approach favors smooth regression equations. We use roughness metrics only to characterize the performance and power regression models and do not implement roughness penalties.

3.6 Summary

This chapter applies the computational efficiency of statistical inference to comprehensive design space characterization, addressing fundamental limitations in low-dimensional sensitivity analysis. Instead of constraining a study's scope based on simulation costs, we fully characterize performance and power trends for spaces containing hundreds of thousands of designs. This scalability ensures a broad understanding of the space, thereby effectively managing design and metric diversity for emerging design priorities.

We supplement sensitivity analysis with more comprehensive and sophisticated techniques. Pareto frontiers quickly identify the set of performance and power efficient designs, revealing feasible performance targets and power budgets for a comprehensive space. Contour maps tractably visualize performance and power gradients, illustrating bottlenecks within a workload and contrasting bottlenecks across workloads. Roughness metrics efficiently quantify the degree of non-linearity and non-monotonicity with a design topology, focusing designer attention on rough design regions likely to contain interesting trends and compromises. Although these techniques are commonly applied in other engineering domains, their robust application to computer engineering is enabled by the speed of statistical inference. By using regression models to capture the relationship between design metrics and design parameters, we remove detailed microarchitectural simulation from the critical path.

Designer intuition and experience are invaluable assets in hardware design. As designers move into domains where intuition is less mature, however, they will require robust characterization techniques to supplement their domain knowledge. This chap-

ter demonstrates a few of these techniques, providing a quantitative basis for design characteristics, which were only understood subjectively until now. In general, statistical inference bridges the divide between detailed simulation and robust analysis, allowing designers to leverage the wealth of history and literature in classical analysis.

Chapter 4

Optimizing Performance and Power Topologies

Contents

Given metrics, every space of designs will contain a global optimum that maximizes those metrics. Robust optimization will traverse the design space to identify either this optimum or some other design that performs comparably for the metrics of interest. Given this definition of robustness, low-dimensional sensitivity analysis and optimization lacks robustness due to its limited scope and significantly sub-optimal results. To improve robustness, we increase the scope of optimization and consider a design space that varies all parameters simultaneously.

Designers have two options when optimizing a large, comprehensive space of designs: exhaustive or heuristic search. Exhaustive search evaluates the design metric for every point in the space to report the global optimum with certainty. Thus, exhaustive search is a trivially robust optimization strategy. Exhaustive search with regression models may be tractable for design spaces with up to hundreds of thousands of points (*e.g.*, seven-dimensional design space of Table B.3). However, such an approach becomes intractable when considering application-specific models for a large number of applications or when considering even larger design spaces with hundreds of billions of points (*e.g.*, fifteen-dimensional design space of Table B.4).

In contrast, heuristic search iteratively traverses the design space, evaluating metrics for a subset of points in the space, to report a possible optimum. The incomplete evaluation of the space makes heuristic search attractive in scenarios where exhaustive search is prohibitively expensive or intractable. However, heuristic-reported optima are potentially deficient relative to the true global optimum. The robustness of

heuristic search highly depends on the heuristic and its implementation. Strategies to improve robustness of iterative optimization heuristics (*e.g.*, increasing the number of iterations) also increase their computational costs.

This chapter considers robust optimization before applying both exhaustive and heuristic optimization to analyze fundamental hardware design paradigms:

- **Robust Optimization:** We consider factors influencing the robustness of gradient ascent, a popular and well-known optimization heuristic. Robust implementations require a greater number of random trials, iterations per trial, and predictions of metrics per iteration. Simulation alone cannot provide the large number of required performance and power estimates. Regression models provide the needed efficiency for robust implementations of these iterative optimization heuristics. (Section 4.1)

- **Multiprocessor Heterogeneity:** Multiprocessor heterogeneity mitigates the performance and power penalties of homogeneous design compromises. We implement exhaustive optimization with regression models to identify efficiency maximizing designs for each workload. These per workload optima are clustered to identify heterogeneous compromises where the degree of heterogeneity determines the number of clusters used. The analysis quantifies efficiency trends and limits from multiprocessor heterogeneity. (Section 4.2)

- **Microarchitectural Adaptivity:** Microarchitectural adaptivity reconfigures the hardware to match application dynamics, thereby enhancing performance while localizing associated power costs. We implement heuristic optimization

with regression models to identify, for each of many adaptive intervals, the best configuration from a space with hundreds of billions of hardware configurations. The analysis quantifies efficiency trends and limits from microarchitectural adaptivity. (Section 4.3)

Collectively, our analyses illustrate robust optimization techniques, demonstrating both exhaustive and heuristic optimization for fundamental hardware design paradigms. These more sophisticated optimization techniques, combined with regression models, enable a more detailed analysis of these design paradigms than was previously possible using traditional approaches to simulation.

4.1 Robust Optimization

To consider the trade-offs between heuristic robustness and computational cost, we consider a representative heuristic: gradient ascent [51]. Also known as steepest ascent or hill climbing, gradient ascent is an iterative optimization heuristic that begins at an initial point and steps toward a local maximum by moving in the direction of steepest change specified by the gradient. In practice, the gradient is approximated by identifying the direction toward a point's best neighbor in the p-dimensional space. Since the heuristic may identify a local maximum, robust implementations iterate and start at different initial points to ensure a reported maximum is reached consistently, thus increasing the likelihood of an accurate approximation to the global maximum. The computational efficiency of regression models provide an opportunity to assess gradient ascent in a manner previously not possible by

Trials	Number of random starting points. Gradient ascent returns best result across all trials.
Iterations	Number of steps required before a trial converges to a point with no significantly better neighbors.
Deficiency	Difference between optima from gradient ascent and exhaustive search.

Table 4.1: **Gradient Ascent Terms and Definitions**. Gradient ascent is an iterative optimization heuristic with several measures of cost and effectiveness.

- comparing search results from gradient ascent against the true global optimum, which is identified by exhaustively evaluating regression models.
- assessing the frequency and likelihood of identifying the true global optimum across trials of gradient ascent beginning at random starting points.
- assessing the distribution of convergence times (*i.e.*, number of steps before a trial of gradient ascent stops) across trials of gradient ascent beginning at random starting points.

Collectively, this thorough analysis of gradient ascent cost and effectiveness shows robust implementations are not possible under conventional approaches to simulation. Furthermore, the large number of iterations needed to thoroughly evaluate gradient ascent is prohibitively expensive using detailed simulation. We analyze gradient ascent since it is widely known and is likely the first method tried by a typical user, but our conclusions may be more generally interpreted for iterative optimization heuristics where robustness depends on algorithmic or implementation knobs exposed to the user.

```
for (t = 1 to max_trials)

   x_old = <random initial design>
   e_old = EVAL(x_old)
   term_flag = 0;

   while(term_flag == 0) {

      // evaluate optimal neighbor
      N = neighborset(x_old);
      x_new = argmax{N | EVAL(N)};
      e_new = EVAL(x_new);

      // compare new against previous
      // and terminate if difference is less
      // than threshold (e.g., 1.001)
      if (e_new/e_old < 1.001) {
         term_flag = 1;
      }
      x_old = x_new;
      e_old = e_new;

    } // end while
    metric(t) = e_old;
    design(t) = x_old;

} // end for

// identify optimal design across
// per trial maxima
m = argmax(t | metric(t))
return(metric(m), design(m))
```

Figure 4.1: **Gradient Ascent Implementation**. Gradient ascent iteratively steps in direction of gradient until convergence criteria are satisfied. Multiple trials start at different random points.

	Set	Parameters	Measure	Range	$\|S_i\|$
S_1	Depth	depth	FO4	9::3::36	10
S_2	Width	width	issue b/w	2,4,8	3
		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	KB	16::2x::256	5
S_6	D-L1 Cache	d-L1 sache size	KB	8::2x::128	5
S_7	L2 Cache	L2 cache size	MB	0.25::2x::4	5

Table 4.2: **Design Space II**. Used for design characterization and optimization where regression models are evaluated exhaustively for every point in the space. $p = 7$, $|S| = 3.8\text{E+}5$.

4.1.1 Implementation

Figure 4.1 outlines our implementation of gradient ascent. Each trial begins at a randomly chosen design. Each iteration in the `while` loop of this trial will exhaustively compare the current design against all neighboring designs, identifying and selecting the best neighbor for the next iteration of the loop. A trial terminates if the new design's metric differs from old design's metric by less than some threshold (*e.g.*, 0.1 percent). At termination, the maximum identified by the trial is logged into a list of search results. Gradient ascent returns the best of these results after `max_trials`.

We implement gradient ascent to search the performance (*bips*) and efficiency ($bips/w$) topologies of Table 4.2. The appropriate regression models are used to compute `EVAL(n)` for each neighbor. The set of neighbors is defined by all designs

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

Table 4.3: **Benchmarks**.

that can be reached by changing any combination of parameter values by at most one step where parameter step sizes are specified in the range column of Table 4.2. We define each neighbor in a p-dimensional design space with a p-element vector where each element can take one of three values: step-up, step-down, unchanged. Points at the interior of the design space have 3^p such vectors, resulting in $2,187$ neighbors for our seven-dimensional design space. The computational efficiency of our performance and power regression models enable us to evaluate such a large number of predictions per iteration until the trial converges. This process is repeated for `max_trials`=1,000.

4.1.2 Evaluation

Since each trial begins at a randomly chosen initial design, the search result may vary from trial to trial. Figure 4.2 captures the distribution of identified *bips* and *bips/w* values from the 1,000 trials. The histograms shown for ammp are representative of those for the broader benchmark suite. There are obvious modes for both metrics, but Figure 4.2L suggests the *bips* topology is more effectively explored by gradient ascent. Although sub-optimal local maxima are occasionally identified, 57.6 percent of trials converged to the same optimal value of 1.34 *bips*. In contrast, the *bips/w* mode in Figure 4.2R is more pronounced with 79.1 percent of trials converging to 0.045 *bips/w*, but 14.0 percent of trials identify more efficient designs ranging from 0.048 to 0.062 *bips/w*.

The performance and power trade-offs increase the non-linearity of the *bips/w* topology resulting in modes below the max. Figure 4.3 summarizes the *bips* and *bips/w* differences across the suite of benchmarks by identifying the fraction of trials that report the best results (*i.e.*, the right-most bar of Figure 4.2 for each benchmark). 37.5 percent of trials for *bips* and, for most benchmarks, less than 10.0 percent for *bips/w* report the best result. Although gradient ascent requires only one trial to identify a maximum, a sub-optimal mode reduces the likelihood of such a trial and implies a larger number of trials is needed to find a global maximum with confidence.

Gradient ascent is a search heuristic that may identify local maxima. Multiple trials increase the likelihood of finding the global maximum, but this result is not guaranteed for non-monotonic topologies. Figure 4.4 assesses the deficiency of gradient ascent in the *bips/w* topology for several representative benchmarks. Deficiency

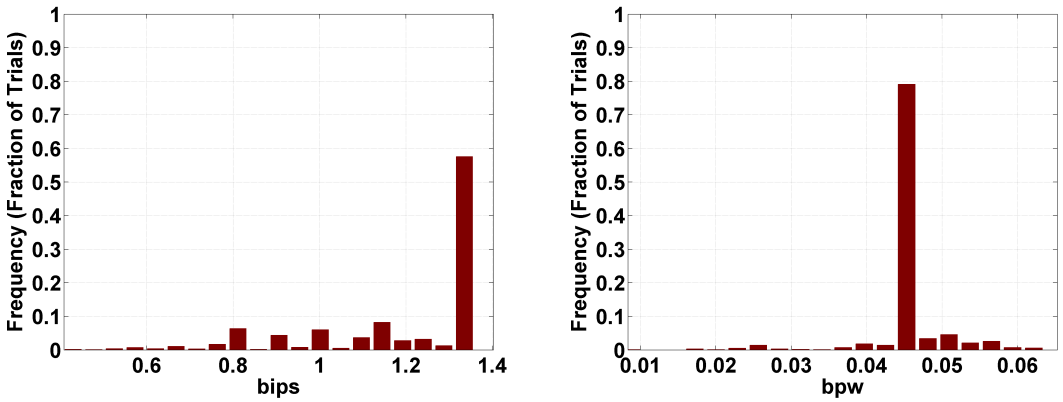

Figure 4.2: **Gradient Ascent Results**. Histogram of values reported by 1,000 trials of gradient ascent with ten *bips* (L) and *bips/w* (R) bins for a representative benchmark SPEC ammp.

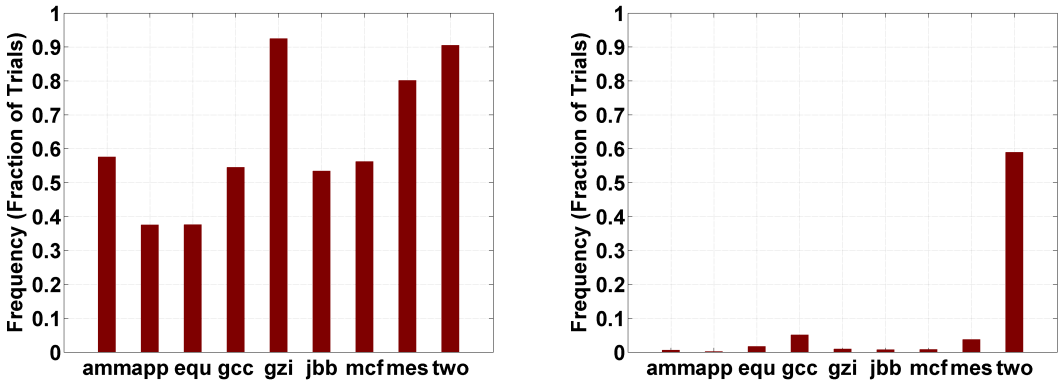

Figure 4.3: **Gradient Ascent Effectiveness**. Number of gradient ascent trials achieving *bips* (L) and *bips/w* (R) in the optimal bin. Equivalent to the right-most bar of Figure 4.2 across nine SPEC benchmarks of Table 4.3.

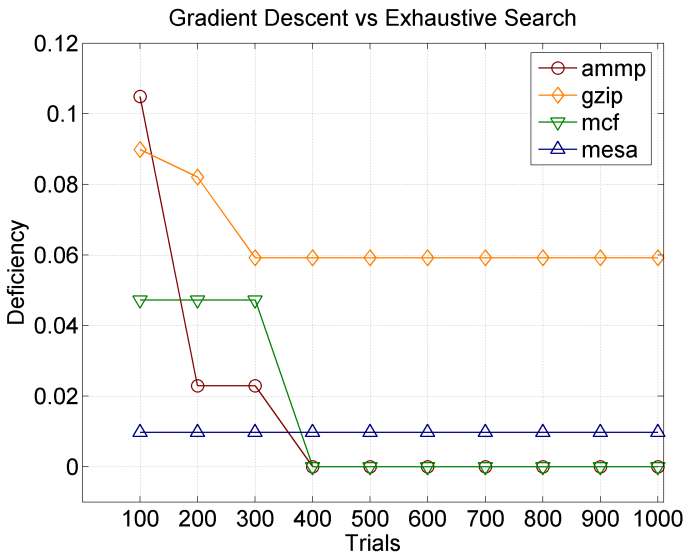

Figure 4.4: **Gradient Ascent Deficiency**. Gradient ascent *bips*/*w* deficiency computed relative to global optimum identified by exhaustively evaluating regression models for every point in the design space of Table 4.2.

quantifies the difference between the results of gradient ascent and those of exhaustive search using regression models. Deficiency may decrease with trial count, but is often observed to stabilize at 400 to 500 trials when optimizing *bips*/*w*. In contrast, gradient ascent achieves zero deficiency when traversing the *bips* topologies. Table 4.4 summarizes gradient ascent effectiveness, assessing stable deficiencies and trial counts needed for stabilization.

In addition to trial count, the number of iterations per trial illustrates gradient ascent's ability to converge to a maximum quickly when starting at various random points in the topology. Figure 4.5 uses boxplots to examine the distribution of convergence times for 1,000 trials when optimizing *bips* and *bips*/*w*. Figure 4.5L indicates the median number of iterations per trial is 10 and most trials converge and exit the `while` loop in fewer than 20 iterations. In contrast, the *bips*/*w* topology may be more

Benchmark	*bips*			*bips/w*		
	GA	Def (%)	Trials	GA (10^{-2})	Def (%)	Trials
ammp	1.36	0.00	100	6.36	0.00	400
applu	1.44	0.00	100	7.37	0.81	500
equake	1.02	0.00	100	5.22	12.94	500
gcc	1.47	0.00	100	5.22	2.72	200
gzip	2.23	0.00	100	8.63	5.92	300
jbb	2.40	0.00	100	7.95	1.89	900
mcf	0.54	0.00	100	2.72	0.00	400
mesa	3.73	0.00	100	8.95	0.97	100
twolf	1.64	0.00	100	6.01	0.91	300

Table 4.4: **Gradient Ascent Results**. Gradient ascent deficiency and the number of trials required to minimize deficiency.

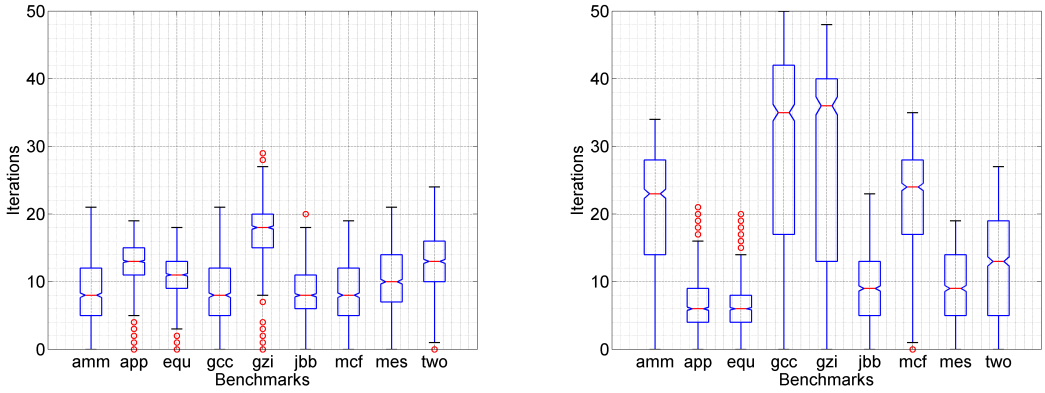

Figure 4.5: **Gradient Ascent Convergence**. Distribution of trial convergence times for 1,000 gradient ascent trials optimizing performance (L) and *bips/w* efficiency (R). Time measured in number of iterations before convergence criteria satisfied.

difficult to traverse with a median convergence time of 9 iterations over five benchmarks (applu, equake, jbb, mesa, twolf), but a median convergence time of 30 over the remaining benchmarks (ammp, gcc, gzip, mcf). Convergence times exceeding 20 iterations is common for these latter benchmarks.

Thus, a robust implementation of gradient ascent may require hundreds of trials and tens of iterations per trial to ensure the optimal design is identified. Each of these iterations must identify the gradient by evaluating the performance and power of all neighboring points. The number of points increases exponentially with design space dimensionality; a point in a p dimensional design space has 3^p neighbors.

Cycle-accurate simulation cannot tractably handle a large number of trials, iterations per trial, or predictions per iteration. Furthermore, in simulation-based iterative heuristics, each step towards the optimum requires simulation. The collected simulator data is specific to the particular optimization path and cannot be re-used for other studies. In contrast, simulations to construct regression models are quickly amortized across multiple optimization studies, multiple possible optimization heuristics, and multiple paths within each heuristic.

4.2 Multiprocessor Heterogeneity

In a uniprocessor or homogeneous multiprocessor design, the core is designed as a compromise between per workload optima to accommodate a range of workloads. Heterogeneous multiprocessor core design mitigates the efficiency penalties of this compromise [39]. Multiple heterogeneous design compromises increase the likelihood that each workload will execute on a compromise that more closely resembles the

workload's optimal design. However, prior work could not identify heterogeneous core designs from a broad space due to simulation costs. These costs constrained prior analyses to consider three to four existing core designs or cores drawn from a modestly sized space. In contrast, this study identifies design compromises for the $bips^3/w$ design metric from a comprehensive design space and quantifies a theoretical upper bound on the potential efficiency gains from heterogeneity.

This section combines regression models with exhaustive optimization and heuristic clustering to assess the trends and limits of multiprocessor heterogeneity:

- **Exhaustive Optimization:** We leverage the computational efficiency of regression models to exhaustively evaluate the performance and power efficiency of every design point. The most efficient design for each workload is identified. These optima are the basis for design compromises.

- **Heuristic Clustering:** We implement K-means clustering to examine a range of design compromises. The continuum between completely homogeneous and completely heterogeneous multiprocessors is expressed in the clustering heuristic as a continuum between one cluster across the workload suite and one cluster per workload, respectively. By varying the number of clusters, we assess varying degrees of heterogeneity.

- **Heterogeneity Efficiency Trends:** We assess $bips^3/w$ efficiency trends and limits from multiprocessor heterogeneity. Complete heterogeneity, where every workload executes on its efficiency maximizing design, achieves significant efficiency gains of up to 2.4x relative to homogeneous multiprocessors. We also

	Depth	Width	Reg	Resv	I-$ (KB)	D-$ (KB)	L2-$ (MB)	Delay Model	Power Model
ammp	27	8	130	12	32	128	2	1.0	35.9
applu	27	8	130	15	16	8	0.25	0.8	39.6
equake	27	8	130	15	64	8	0.25	1.2	41.5
gcc	15	2	70	9	16	8	1	1.2	44.1
gzip	15	2	70	6	16	8	0.25	0.8	24.2
jbb	15	8	80	12	16	128	1	0.6	80.9
mcf	30	2	70	6	256	8	4	3.5	12.9
mesa	15	8	80	13	256	32	0.25	0.4	86.9
twolf	27	8	130	15	128	128	2	1.1	34.5

Table 4.5: **Per Benchmark Optima for Heterogeneity Clustering**. $bips^3/w$ maximizing designs for nine SPEC benchmarks of Table 4.3. Per benchmark optima are identified by exhaustively evaluating performance and power regression models for design space of Table 4.2.

expose diminishing marginal returns; four heterogeneous core designs are sufficient to achieve efficiency gains of 2.2x for the nine SPEC workloads considered.

4.2.1 Exhaustive Optimization

We consider the space of Table 4.2 with 375,000 points spanned by seven design parameters. Exhaustively evaluating every point in this design space using regression models for performance and power, we identify each benchmark's global optimum with respect to $bips^3/w$. As shown in Table 4.5, the optimal design parameters exhibit significant diversity across benchmarks with depth ranging from 15 to 30 FO4, width ranging from 2 to 8 instructions decoded per cycle, and L2 caches ranging from 0.25 to 4 MB. Each benchmark's execution characteristics are reflected in its optimal architecture. For example, compute-intensive gzip has the smallest L2 cache while memory-intensive mcf has the largest.

These per benchmark optima are the basis for design compromises with varying degrees of heterogeneity. At one end of this continuum, we consider no heterogeneity and identify a single, homogeneous design compromise for these nine optima. At the other end of this continuum, we consider complete heterogeneity and implement all nine optima with no compromise required. K-means clustering is used to explore all intermediate options where the per benchmark optima of Table 4.5 constitute the set to be partitioned into K subsets.

4.2.2 Heuristic Clustering

We combine our regression models with *K-means clustering*. A K-clustering partitions a set S into K subsets to optimize some clustering criterion, usually a similarity metric. Well defined clusters are such that all objects in a cluster are similar and any two objects from distinct clusters are dissimilar. General K-clustering is NP-hard and K-means clustering is a heuristic approximation [21, 47].

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

1. Define K centroids, one for each cluster, and place randomly at initial locations in space containing objects to be clustered.

2. Assign each object to cluster with closest centroid.

3. When all objects have been assigned, recompute placement of K centroids to minimize its distance to objects in its cluster.

4. Since centroids may have moved in step 3, object assignment to clusters may change. Steps 2 and 3 are repeated until centroid placement is stable.

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

$$d_{Euclidean} = \left(\sum_{i=1}^{p} |a_i - b_i|^2 \right)^{1/2} \tag{4.1}$$

For example, pipeline depth values range from 12 to 30 FO4 in increments of 3 with a mean of 21 and standard deviation of 6.48. The normalized depth values range from -1.39 to 1.39 with mean 0 and standard deviation of 1.0. We then use samples from regression model training to compute the correlation between depth and $bips^3/w$ for a weighting factor. The process is repeated for each of the p parameters to produce design vectors $\vec{a}$ and $\vec{b}$.

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

Table 4.6: **Heterogeneous Cluster Centroids**. Design specifications of centroids from K-means clustering for per benchmark optima of Table 4.5. Figure 4.6 shows cluster assignments for benchmarks. Per benchmark optima identified from design space of Table 4.2 for nine SPEC benchmarks of Table 4.3.

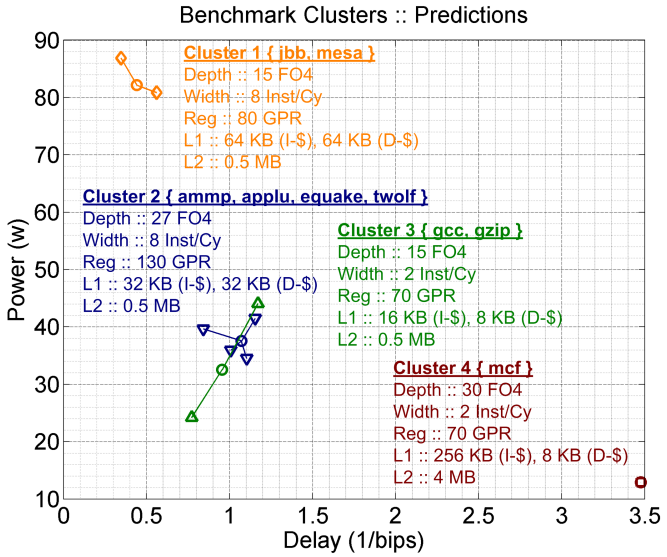

Figure 4.6: **Optimization and Clustering**. Delay and power for per benchmark optima of Table 4.5 (radial points) and resulting compromises/centroids of Table 4.6 (circles).

4.2.3 Heterogeneity Efficiency Trends

Each K-means cluster corresponds to a grouping of similar architectures and each centroid represents its cluster's compromise architecture. We take the number of clusters as the number of distinct compromise designs and, thus, a measure of heterogeneity. Table 4.6 uses a $K = 4$ clustering to identify compromise architectures and their average power-delay characteristics when executing their associated benchmarks. This analysis illustrates our models' ability to identify optima and compromises occupying very different parts of the design space. For example, the four compromise architectures capture all combinations of pipeline depths and widths. Cluster 1 contains the aggressive deep, wide pipeline for jbb and mesa. Cluster 4, containing the memory-intensive mcf, is characterized by a large L2 cache and shallow, narrow pipeline. Clusters 2 and 3 trade-off pipeline depth and width depending on application-specific opportunities for instruction level parallelism. Identifying diverse optima is increasingly important as we observe microarchitectural differentiation for various market segments and applications.

Figure 4.6 plots the delay and power characteristics of the nine per benchmark architectures executing their corresponding benchmarks (radial points). Aggressive architectures with deep, wide pipelines are located in the upper left quadrant and the less aggressive cores with shallow, narrow pipelines are located in the lower right quadrant. Deep, narrow and shallow, wide architectures both occupy the moderate center. The four compromise architectures executing their benchmark clusters are also plotted (circles) to demonstrate the extent of delay and power compromises. Although we cluster in a p-dimensional microarchitectural space, the strong relationship

between a design and its design metrics means we also observe clustering in the 2-dimensional delay-power space. Spatial locality between a centroid and its cluster's objects suggest modest delay and power penalties from architectural compromises. Thus, delay and power for the benchmark suite executing on a $K = 4$ heterogeneous multiprocessor are similar to those when executing on the nine per benchmark optima. As a corollary, the benchmarks could achieve close to ideal $bips^3/w$ efficiency on this heterogeneous design.

Figure 4.6 also reveals new opportunities for workload similarity analysis based on microarchitectural resource requirements. For example, ammp, applu, equake, and twolf may be similar workloads since they are most efficient at similar pipeline dimensions and cache sizes. While much prior work in similarity analysis has been used to reduce the number of benchmarks for microarchitectural simulation, similarity exposed by microarchitectural clustering may be most useful for hardware accelerator design. In the ideal case, accelerators would be designed for every kernel of interest. However, resource constraints necessitate compromises and penalties from such compromises may be minimized by designing an accelerator to meet the needs of multiple similar kernels. Further exploring these opportunities is future work.

Figure 4.7 plots predicted $bips^3/w$ efficiency gains for the nine benchmarks and their average as the number of clusters increases in the K-means heuristic. Recall cluster count quantifies the degree of heterogeneity. Efficiency is presented relative to the POWER4-like baseline (cluster count 0, Table 4.7). The homogeneous architecture identified by K-means clustering (cluster count 1) is predicted to improve average efficiency by 1.46x with the largest gains for mesa (4.6x) at the expense of

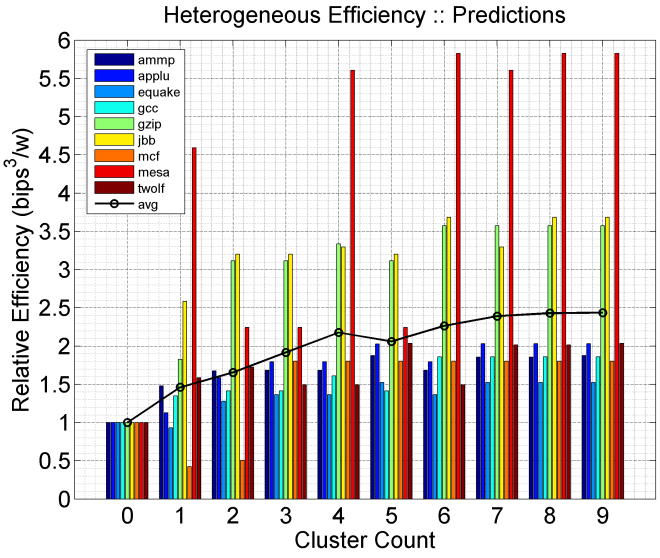

Figure 4.7: **Heterogeneity Efficiency Trends and Limits**. Predicted efficiency gains as a function of heterogeneity. Cluster 0 is baseline of Table 4.7, cluster 1 is homogeneous multicore from K-means, cluster 4 is heterogeneous multicore of Table 4.6, cluster 9 is heterogeneous multicore of Table 4.5.

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

Table 4.7: **POWER4 Baseline**. Superscalar, out-of-order microarchitectural design resembling the IBM POWER4.

mcf (0.46x). For three cores, all benchmarks see benefits from heterogeneity resulting in an average gain of 1.9x. We observe diminishing marginal returns in heterogeneity beyond four cores. The four cores in Table 4.6 are predicted to benefit efficiency by 2.2x, 8 percent less than the theoretical upper bound of 2.4x that is achievable only from the much greater heterogeneity of seven to nine cores. The benefits for nine different cores is the theoretical upper bound on heterogeneity benefits as each benchmark executes on its $bips^3/w$ maximizing core.

4.2.4 Heterogeneity Validation

Figure 4.8 compares the simulator reported heterogeneity gains against those of our regression models. The models are pessimistic for lower degrees of heterogeneity (*i.e.*, cluster counts less than four). The gap between predicted and simulated efficiency narrows from 37.9 percent at cluster count zero to 14.4 percent at cluster count three. The simulated four core average benefit is 2.0x compared to the regression-predicted benefit of 2.2x. This point of diminishing marginal returns from additional heterogeneity is predicted with a 7.8 percent error; the regression models are relatively optimistic. At higher degrees of heterogeneity (*i.e.*, cluster counts greater than six), we observe much greater accuracy with error rates less than 3.0 percent. The predicted upper bound on heterogeneity benefits of 2.4x is accurate with only 1.7 percent difference in simulation.

Figure 4.9 assesses benchmark level effects, illustrating efficiency trends at varying degrees of heterogeneity. The regression models effectively capture application-specific effects such as significant benefits for mesa at the cost of mcf for low cluster

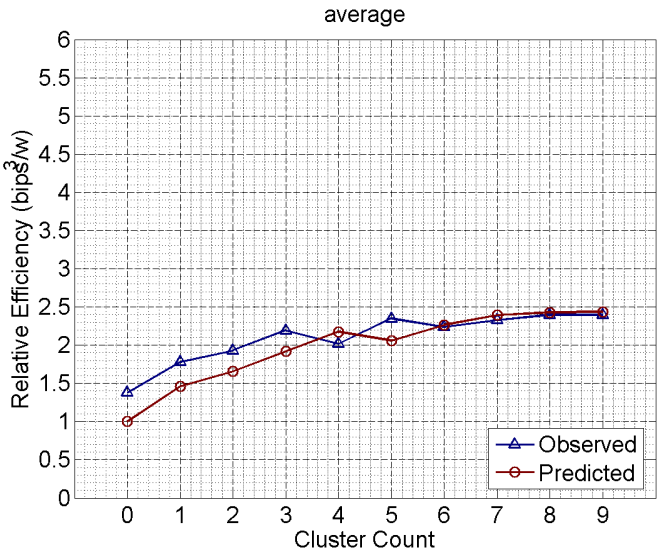

Figure 4.8: **Heterogeneity Validation for Benchmark Average**. $bips^3/w$ efficiency validation for average of nine SPEC benchmarks of Table 4.3. X-axis interpreted as in Figure 4.7.

counts. We also observe similar heterogeneity trends for benchmarks within the same cluster. For example, Figure 4.6 identified a cluster with ammp, applu, equake and twolf. Since these benchmarks have similar resource requirements at the microarchitectural level, their achieved efficiency gains in the range of 1.5x to 2.0x are also similar. Collectively, these figures illustrate our models' abilities to capture the relative benefits of heterogeneity across benchmarks.

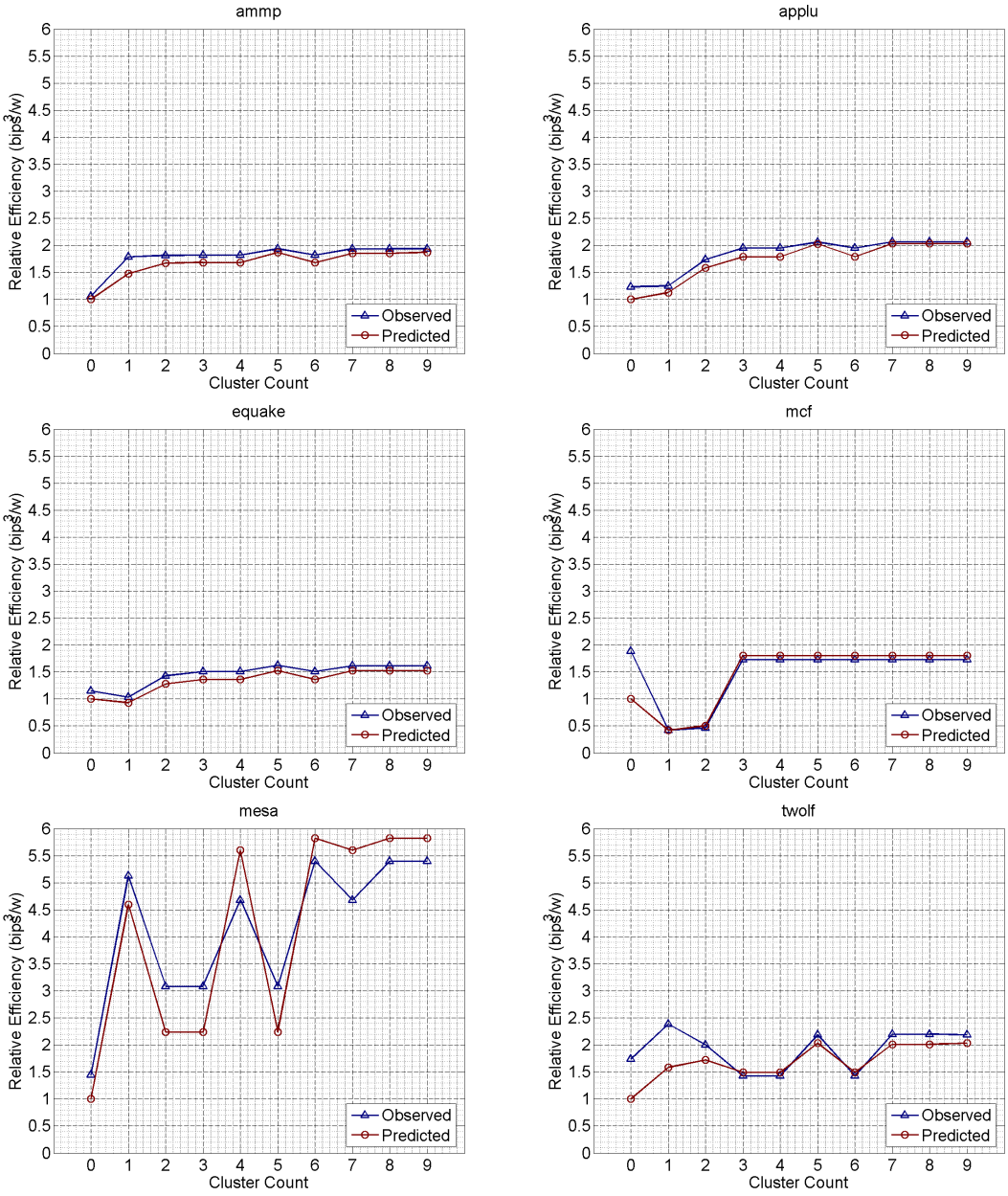

Figure 4.9: **Heterogeneity Validation for Representative Benchmarks**. $bips^3/w$ efficiency validation for representative SPEC CPU benchmarks of Table 4.3. X-axis interpreted as in Figure 4.7.

4.3 Microarchitectural Adaptivity

Adaptive microarchitectures arise from a design paradigm that promises greater performance and power efficiency by dynamically allocating computational resources to meet the requirements of a workload more effectively. Adaptivity enables power efficient, high-performance microarchitectures by provisioning (or over-provisioning) resources to maximize overall performance while increasing the locality of associated power costs; resources are active and consume power only in periods of computation that utilize those resources. The feasibility of this paradigm will be decided by assessing benefits and costs, but a comprehensive assessment of potential benefits has long eluded researchers due to the challenging dimensionality of the analysis.

Combining regression with heuristic optimization, we assess the trends and limits of microarchitectural adaptivity in both temporal and spatial dimensions:

- **Adaptivity Dimensions:** Microarchitectural adaptivity is defined in two dimensions. The temporal dimension describes the frequency of hardware adaptivity while the spatial dimension describes the scope of these reconfigurations. Using traditional approaches to simulation, an analysis of high temporal and spatial adaptivity is intractable.

- **Heuristic Optimization:** We leverage the computational efficiency of regression models to implement genetic algorithms, an iterative optimization heuristic, capable of traversing a design space with hundreds of billions of points. Genetic algorithms identify $bips^3/w$ optima for each adaptive interval to assess potential benefits from comprehensive spatial adaptivity.

- **Temporal Adaptivity:** We assess varying degrees of temporal adaptivity under comprehensive spatial adaptivity, demonstrating significant efficiency gains of up to 5.3x (median 2.4x) from high temporal adaptivity (*e.g.*, adapt for every interval of 0.08M instructions) relative to low-temporal adaptivity (*e.g.*, adapt for every application).

- **Spatial Adaptivity:** We assess varying degrees of spatial adaptivity under high temporal adaptivity, showing three parameters are often sufficient to achieve, on average, 77.3 percent of fifteen-parameter adaptive efficiency. However, the three most significant parameters differ across applications and may be unknown during hardware design.

Collectively, this work establishes a rigorous foundation for assessing the benefits of comprehensive microarchitectural adaptivity and motivates a complementary, equally rigorous analysis of the associated costs and complexities.

4.3.1 Adaptivity Dimensions

An alternative to static general-purpose design, adaptive computing increases the flexibility of the microarchitecture along two dimensions. In one dimension, the degree of *temporal adaptivity* determines the frequency at which resources reconfigure to optimize efficiency. Broadly, this dimension might range from application-level adaptivity to interval-level (*i.e.*, sub-application) adaptivity. Greater temporal adaptivity improves microarchitectural responsiveness to underlying workload heterogeneity but places greater burdens on the adaptive control algorithm.

In the other dimension, the degree of *spatial adaptivity* determines the microarchitectural scope of reconfigurations. The number and adaptive range of reconfigurable design parameters is a unit of measurement for this dimension. More comprehensive spatial adaptivity leverages synergies and interactions between parameters to improve efficiency at the cost of greater complexity.

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

The computational complexity of this problem has hindered advances in microarchitectural adaptivity as prior work often constrained adaptivity in the temporal (*e.g.*, applications[48], working sets[14], subroutines[29], and multimedia frames[30]) and/or spatial (*e.g.*, two or three parameters among depth [17, 67], width[30], queues [2, 61], and caches [5, 48]) dimension. Constraints to temporal adaptivity produce analyses that do not fully illustrate the potential efficiency gains of dynamic structural reconfiguration. Without an analysis of comprehensive spatial adaptivity, prior studies do not account for interactions between parameters as structures adapt, resulting in migrating bottlenecks and limited efficiency gains. Furthermore, only a comprehensive

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

Table 4.8: **Design Space III**. Used for design optimization where regression models are optimized with with iterative heuristics. $p = 15$, $|S| = 2.8\text{E+}11$.

study will reveal the most significant parameters for adaptivity in the presence of such interactions.

4.3.2 Heuristic Optimization

We consider the space of Table 4.8 with 240 billion points spanned by twelve design parameters. Figure 4.10 summarizes the analysis framework. Temporal sampling identifies representative instructions, which are divided into intervals depending on the desired degree of temporal adaptivity. In parallel, we define the space of adaptive

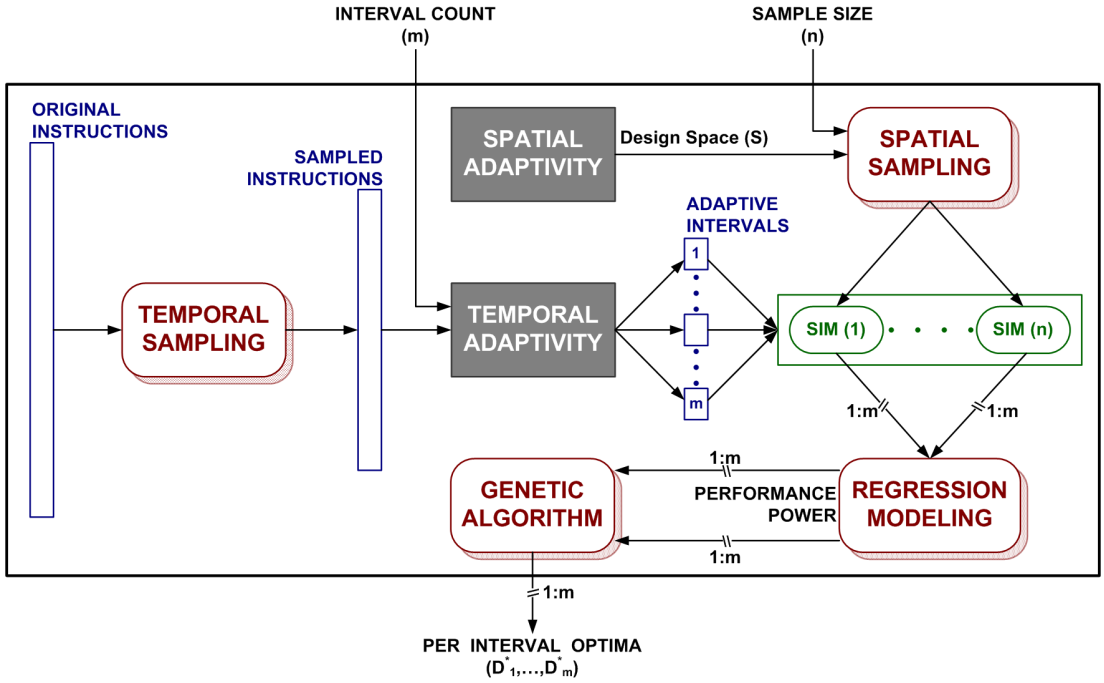

Figure 4.10: **Framework for Adaptivity Analysis**. Framework combines elements of temporal and spatial sampling to construct regression models. Regression models are used to implement genetic algorithms that iteratively search the adaptive space for efficiency maximizing designs. Efficient designs are identified for each adaptive interval.

parameters S. We then sparsely sample and simulate n designs from this space for each of m adaptive instruction intervals. These simulations train per interval regression models for performance and power. Finally, we identify optimal configurations for each interval using genetic algorithms on the regression models.

We estimate upper bounds on the efficiency gains from microarchitectural adaptivity by quantifying gains under best-case, oracle-driven scenarios. In practice, this approach requires identifying efficiency maximizing designs for each interval. Although regression models are orders of magnitude faster than detailed microprocessor simulation, exhaustive search to optimize these models across a design space of 240

billion points remains intractable. We must combine regression models with scalable heuristics, such as genetic algorithms, for global combinatorial optimization.

Genetic algorithms mimic the process of natural selection in which a candidate solution to the optimization problem is treated as an organism and a candidate's optimality is treated as the organism's fitness [22]. Breeding among highly fit organisms increases the likelihood of passing desirable attributes to future generations. Breeding among less fit organisms and the possibility of mutation ensures population diversity. As the population evolves from one generation to the next, the population of candidate solutions improves and the likelihood of observing the global optimum within the population increases. We describe the genetic algorithm in the context of microprocessor optimization and discuss tunable elements of this algorithm including (1) population size and number of generations, (2) parent selection, (3) genetic crossover, and (4) mutation rates.

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

Parent Selection. Both parents are selected by fitness rank where fitness is

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

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

Mutation. Mutations randomly alter an offspring's genetic code to increase population diversity and provide a mechanism for escaping local optima. We implement an aggressive mutation scheme in which each value of the offspring's design vector can independently mutate up or down by one step (as defined by ranges of Table 4.8) with 5 percent probability. This particular mutation rate is empirically found effective when sweeping a range of possible values. If this rate is too low, many potentially

good innovations will be missed. If this rate is too high, the algorithm's ability to preserve desirable attributes will be degraded.

Parents are repeatedly selected to produce mutated offspring until the previous generation is replaced. The algorithm proceeds until the pre-determined generation limit is reached. At termination, the best design in the population is returned. It is not computationally possible to validate results from genetic algorithms against those from exhaustive search. However, we find the same optima are produced with high frequency when repeatedly invoking the genetic algorithm with different starting populations, giving us confidence in the algorithm's ability to converge consistently toward superior designs.

4.3.3 Temporal Adaptivity

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

We compare each benchmark's efficiency gains from sub-application adaptivity against those from application-level adaptivity in which an oracle provides the best configuration for the overall application (Table 4.9). This baseline architecture is identified from optimizing the 81.92M-instruction interval. Thus, each benchmark's

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

Table 4.9: **Per Benchmark Optima for Adaptivity Baseline**. $bips^3/w$ maximizing designs for benchmarks of Table C.1. Per benchmark optima are identified by genetic algorithms implemented with performance and power regression models for Table 4.8.

baseline microarchitecture has already been optimized for overall application efficiency and we quantify only the additional impact from increasing sub-application temporal adaptivity.

Efficiency Trends. Figure 4.11 presents performance, power, and efficiency trends as the adaptive period decreases from 81.92M to 0.08M instructions. The period of 0.08M instructions represents the greatest temporal adaptivity as the microarchitecture adapts to maximize $bips^3/w$ efficiency every 0.08M instructions. These figures illustrate monotonically improving efficiency as temporal adaptivity increases with up to 5.3x efficiency gains (gcc). The source of efficiency gains vary across benchmarks and arise from performance improvements and/or power reductions.

For example, Figure 4.11UL illustrates efficiency gains for blast dominated by *performance improvements*. These trends are also representative of those for equake and mcf. Adapting the microarchitecture every 0.08M instructions improves efficiency

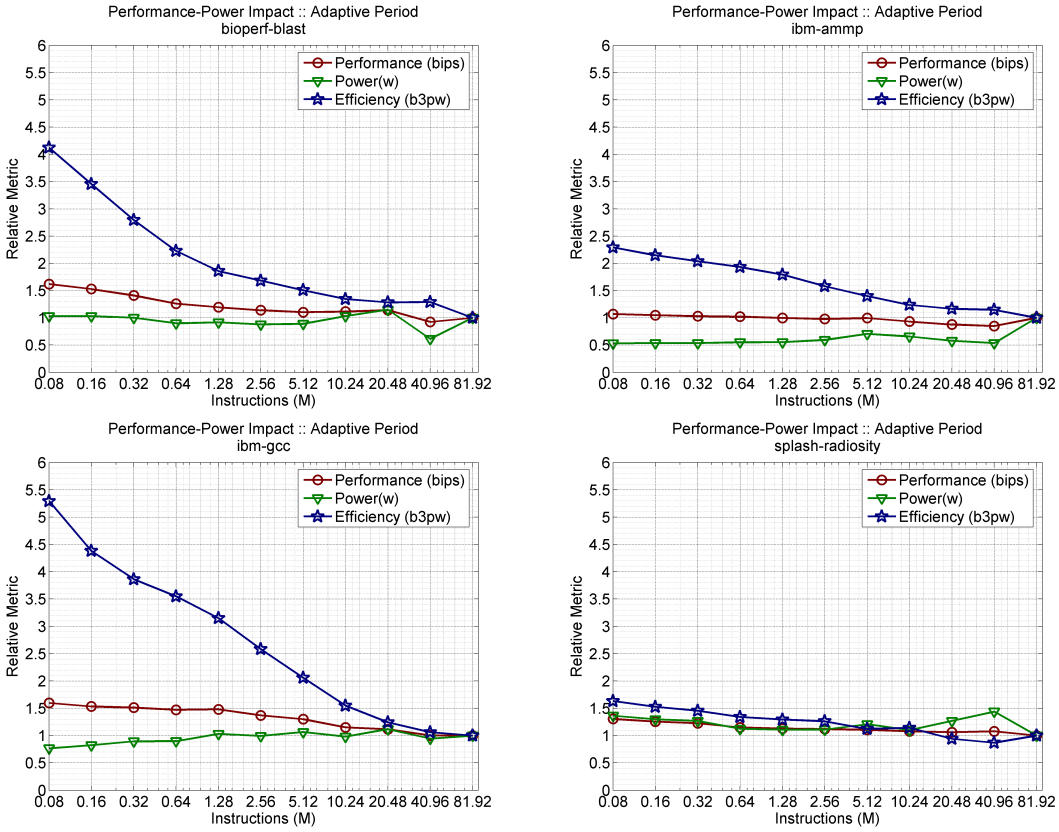

Figure 4.11: **Temporal Adaptivity Trends**. Representative $bips^3/w$ efficiency trends for blast (UL), ammp (UR), gcc (LL) and radiosity (LR). Microarchitecture reconfigures every 81.92M (low temporal adaptivity) to 0.08M instructions (high temporal adaptivity).

by 4.1x, derived from a 62.0 percent increase in performance and negligible 3.1 percent increase in power relative to application-level adaptivity. Power trends are flat and incremental performance improvements contribute to much of the efficiency gains as the adaptive interval sizes decrease from 81.92M to 0.08M instructions.

In contrast, Figure 4.11UR illustrates efficiency gains for ammp characterized by significant *power reductions*. These trends are also representative of those for applu and cholesky. Ammp achieves maximum efficiency gains of 2.3x from a modest 6.9 percent increase in performance and 46.7 percent decrease in power relative to the static baseline. Although modest, incremental performance improvements between 2.56M- and 0.08M-instruction intervals provides monotonically increasing efficiency, adaptivity notably reduces power by 29.4 to 46.7 percent contributing significantly to greater efficiency across all adaptive periods.

Figure 4.11LL illustrates the more common case in which increasing temporal adaptivity both *increases performance and decreases power*. Trends are illustrated for gcc, but are representative of those for gzip, jbb, raytrace, and ocean. Microarchitectural reconfigurations every 0.08M instructions improves gcc performance by 59.6 percent and reduces power by 23.25 percent for a 5.3x increase in efficiency. Adaptive optimizations for many short intervals exploit their differing computational requirements. Greater power may be consumed for high performance intervals that require additional resources, but the associated high power costs are incurred only for the duration of these particular intervals and do not translate to significantly higher power dissipation for the overall workload. Similarly, low power designs with fewer resources are often favored for non-computational intensive intervals, thereby

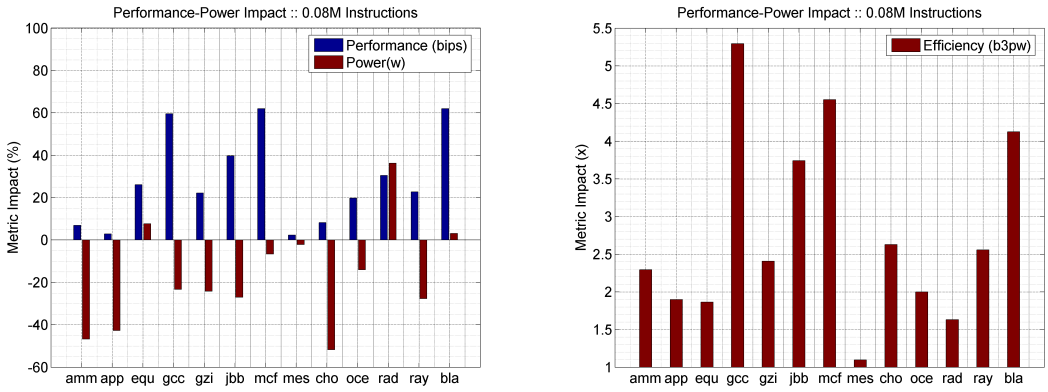

Figure 4.12: **Temporal Adaptivity and Efficiency**. Performance, power (L) and efficiency impact (R) from high temporal adaptivity. Microarchitecture reconfigures every 0.08M instructions (high temporal adaptivity).

reducing power without significantly impacting performance. Thus, higher temporal adaptivity matches hardware to application dynamics and localizes power costs, simultaneously enabling net performance increases and net power reductions.

Lastly, Figure 4.11LR illustrates a trend observed only for radiosity in which adaptivity *increases performance and power* together for a net efficiency gain. Microarchitectural reconfigurations every 0.08M instructions improves radiosity efficiency by 1.6x from a 30.5 and 36.3 percent increase in performance and power, respectively. The $bips^3/w$ metric emphasizes performance over power such that a one percent increase in performance is efficient if power increases by less than approximately three percent. In the case of radiosity, performance and power increases track linearly as temporal adaptivity increases, improving efficiency despite increasing power costs.

Figure 4.12 summarizes the potential performance, power and efficiency impact of high temporal adaptivity (0.08M-instruction intervals) under the comprehensive spatial adaptivity of Table 4.8. Figure 4.12L illustrates diverse performance and power

effects across the benchmark suite. Performance increases by up to 62.0 percent (mcf) and power decreases by as much as 51.7 percent (cholesky). As observed for representative benchmarks in Figure 4.11, various combinations of performance and power compromises are used to achieve greater efficiency and no single trade-off dominates. Figure 4.12R illustrates efficiency gains for the benchmark suite with median and maximum efficiency gains of 2.4x and 5.3x, respectively.

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

Figure 4.13 plots the cumulative distribution function (CDF) for the number of parameters that change between consecutive intervals. Taking Figure 4.13L for a representative workload, raytrace, 50 and 75 percent of transitions between 0.08M-instruction intervals require design value changes for at most 3 and 6 parameters, respectively. 95 percent of these transitions require changes for at most 10 parameters. However, the degree of temporal adaptivity impacts the number of parameter changes between consecutive intervals.

High temporal adaptivity (*e.g.*, 0.08M-instruction intervals) smoothes microarchitectural reconfigurations by enabling smaller, intermediate changes for more frequent, shorter intervals. Reduced temporal adaptivity (*e.g.*, 10.24M-instruction intervals) degrades this smoothing effect, requiring changes to increase in scope to include more

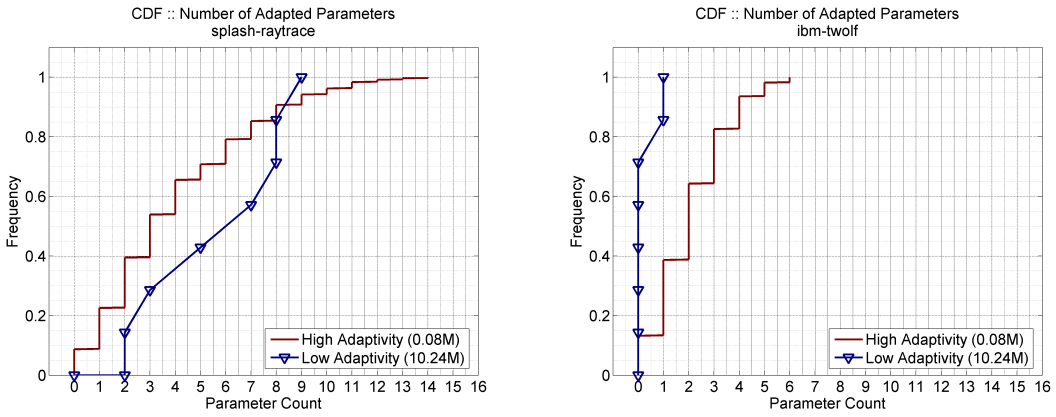

Figure 4.13: **Number of Parameters Utilizing Adaptivity**. Number of parameters that adapt between consecutive intervals for raytrace(L) and twolf(R).

parameters. For example, Figure 4.13L illustrates a CDF shift where less frequent adaptivity requires changes to more parameters. For 10.24M-instruction intervals, every transition requires at least changes for 2 parameters. 50 and 75 percent of transitions now require changes for at most 6 and 8 parameters, respectively.

Reduced temporal adaptivity also acts as a low pass filter on microarchitectural reconfiguration, removing short term variations and leaving only the long term trend. This filtering effect reduces the number of parameter changes optimized for specific short intervals with uncommon resource requirements. Figure 4.13L illustrates this filtering effect where reducing temporal adaptivity from 0.08M-instruction to 10.24M-instruction intervals eliminates the uncommon 5 percent of transitions that change more than 10 parameters. Thus, reduced temporal adaptivity reduces smoothing and increases filtering effects to shift the distribution toward middle range parameter counts. These trends for raytrace are comparable to those for 9 of 14 benchmarks.

In contrast, Figure 4.13R illustrates trends from twolf that are representative of

gzip, mesa, twolf, ocean, and radiosity. Twolf is characterized by low reconfiguration diversity with 95 percent of its transitions between 0.08M-instruction intervals requiring changes to at most 5 parameters. The filtering effects from reduced adaptivity dominate and 70 percent of transitions between 10.24M-instruction intervals do not require reconfiguration.

While Figure 4.13 illustrates the number of parameters that adapt between intervals, Figure 4.14 quantifies the magnitude of changes in design parameter values. We quantify relative step size by reporting the change in a parameter's value relative to the number of steps in the parameter's range. For example, we consider register file sizes from 40 to 130 entries in increments of 10 entries (9 possible steps). If the microarchitecture changes from 70 to 90 entries, the register file effectively takes 2 steps over 9 possible values for a 0.22 relative step size. As relative step sizes approach one, the interval transition approaches reconfigurations that change a parameter from its minimum to its maximum value.

Figure 4.14 indicates parameter values change more significantly when a greater number of parameters change simultaneously. Taking raytrace as an example,[1] the median step size increases from 0.23 to 0.67 as the number of changing parameters increases from 1 to 10. If 7 or more parameters are adapted in an interval transition, 50 percent of these transitions will require changes that span more than half the possible values in parameters' ranges. Note the relative location of the median within each box shifts upward as the number of changed parameters increases, further indicating shifts from small step sizes to large step sizes. Collectively, these trends

[1]The trends for raytrace are representative of all benchmarks except twolf whose trends are flat and not shown.

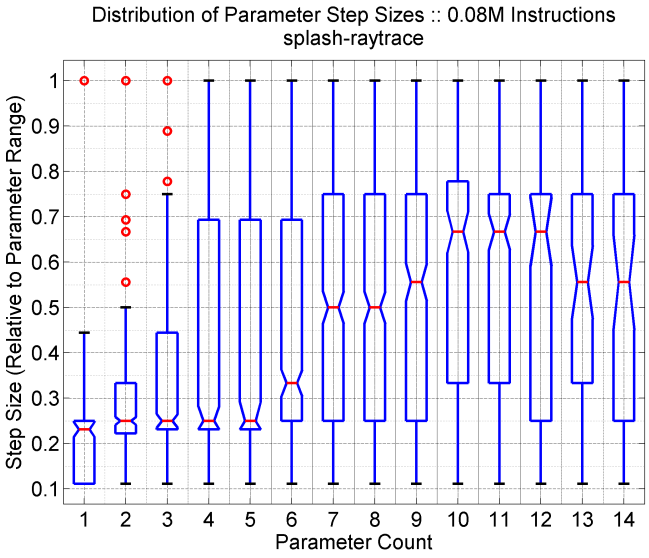

Figure 4.14: **Changes for Parameters Utilizing Adaptivity**. Magnitude of change for parameters that adapt between consecutive intervals for raytrace.

suggest synergies between parameters as they simultaneously change to ensure no bottlenecks are created. As more parameters change, each parameter will change more significantly. Taken together, Figures 4.13–4.14 characterize the utilized adaptivity for representative benchmarks by quantifying the number of parameters that change between consecutive intervals and the magnitude of these changes.

4.3.4 Spatial Adaptivity

The previous analysis of temporal adaptivity assumed comprehensive spatial adaptivity where every parameter of Table 4.8 could be changed to fit each interval's requirements. However, we can also use our framework to assess the impact of reduced spatial adaptivity. In particular, we reduce the number of parameters available for reconfiguration while assuming high temporal adaptivity where reconfigurations

occur every 0.08M instructions. We also consider the effects of increased spatial adaptivity by adding dynamic voltage and frequency scaling to the adaptive space.

Reduced Spatial Adaptivity. We identify the three most significant parameters for achieving efficiency by exhaustively evaluating the $\binom{15}{k}$ possible combinations for $k = 1, \ldots, 3$. In particular, there are $\binom{15}{1} = 15$, $\binom{15}{2} = 105$, $\binom{15}{3} = 455$ ways to select one, two, and three parameter(s) for adaptivity, respectively. For each k and each of the $\binom{15}{k}$ possible combinations given k, we repeat the optimization of Section 4.3.3 to identify the efficiency maximizing combination. Although prior work in adaptive microarchitectures frequently consider two or three parameters, the choice of parameters is often made prior to any analysis. In contrast, we consider two or three parameters empirically found to be most significant for each benchmark. Thus, this analysis compares comprehensive spatial adaptivity against best-case scenarios in limited adaptivity where the most significant parameters are considered.

Table 4.10 identifies the one, two, and three parameter(s) that provide the greatest efficiency gains from microarchitectural adaptivity. Adaptive pipeline depths are most promising with all benchmarks ranking the depth parameter among the top three. Also significant, but more sparsely ranked, are cache hierarchy parameters. Benchmarks benefit from adaptive caches, suggesting optimal cache sizes and associativities vary significantly across intervals. We consider adaptive `memLat` as a proxy for an adaptive L3 cache that reduces effective memory latency. Logic (*e.g.*, `width`) and associated queues (*e.g.*, `lsq, bp`) are less prominent relative to depth, suggesting interval-to-interval variability is greater for memory access patterns than instruction level parallelism.

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

Table 4.10: **Reduced Spatial Adaptivity and Significant Parameters**. Choice of $k = 1, \ldots, 3$ parameters that maximize adaptive efficiency gains. * denotes parameters that became less significant with additional adaptivity (*e.g.*, 2* for gcc l2Assoc indicates it was among the 2, but not the 3, most significant parameters.)

From the perspective of hardware implementation, Table 4.10 has significant implications for design complexity. The highly ranked parameters differ across the benchmark suite and most parameters are highly ranked for at least one benchmark (except the register file and instruction cache parameters). This motivates a hardware substrate for comprehensive adaptivity, especially for the memory hierarchy. Furthermore, these parameter rankings do not necessarily contain hierarchical subsets; a parameter that might be significant when two parameters are considered may be much less significant when three are considered. We observe this scenario for gcc, cholesky, radiosity, and raytrace.

Figure 4.15 quantifies the best achievable efficiency when at most three parameters are chosen for adaptivity. The efficiency for each benchmark is reported under its optimal subset as shown in Table 4.10. Most benchmarks require only a few adaptive parameters to achieve a high fraction of potential efficiency gains. On average,

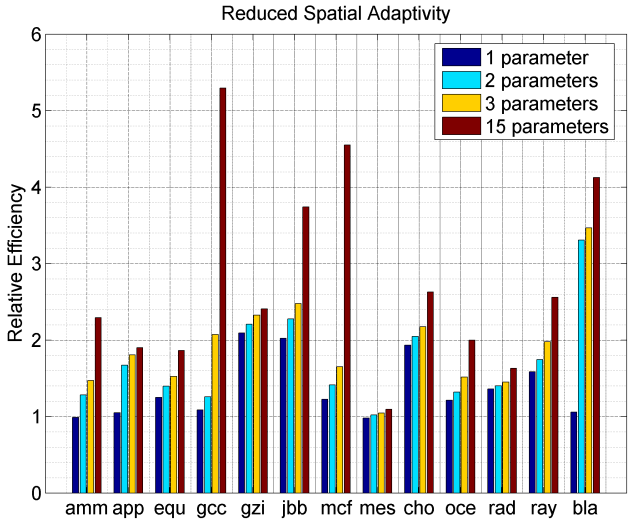

Figure 4.15: **Reduced Spatial Adaptivity and Efficiency**. Efficiency comparison between reduced and comprehensive spatial adaptivity. Efficiency for 1-3 parameters is reported for the 1-3 parameters that maximize $bips^3/w$. Each benchmark is evaluated for different sets of 1-3 parameters as described in Table 4.10.

benchmarks are able to achieve 60.3, 71.1, and 77.3 percent of 15-parameter efficiency as the number of adaptive parameters increases from one to three (medians of 61.4, 76.4, 82.3 percent). However, as illustrated in Table 4.10, the optimal choice of two or three parameters needed to deliver such efficiency may differ substantially across benchmarks. This variation makes identifying any minimal adaptive microarchitectural substrate difficult.

Four benchmarks, ammp, gcc, jbb and mcf, are notable for achieving relatively small fractions of potential. Adapting their three optimally chosen parameters only produces efficiency between 36.2 and 66.3 percent of potential. We observe steady, but modest, efficiency benefits from increasing the number of adaptive parameters from one to three. Given that results are reported for three parameters exhaustively found

to be most significant, additional parameters are also likely to produce only modest and incremental efficiency increases toward the 15-parameter potential. Indeed, a subsequent search for a fourth significant parameter for these benchmarks further increases efficiency by a modest 8 to 10 percent, putting efficiency between 39.4 and 71.7 percent of potential. Thus, we expect much more comprehensive spatial adaptivity is required to close the gap for these benchmarks, drawing on incremental efficiency improvements and synergies from additional parameters.

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

Voltage and Frequency Scaling. Dynamic voltage and frequency scaling (DVFS) adapts pipeline sensitivity to memory latency by, for example, slowing computation and reducing power dissipation during long memory stalls. Since DVFS changes the relative clock speeds between the processor core and memory, we model its effects by scaling off-chip memory latency. Since the performance models were derived for a baseline frequency, we must further scale regression predicted *bips* to account for the frequency change. Similarly, we scale regression predicted power to account for voltage and frequency changes.

Figure 4.16 quantifies efficiency gains from DVFS when applied to various degrees of temporal adaptivity. Most benchmarks do not realize significant $bips^3/w$ efficiency

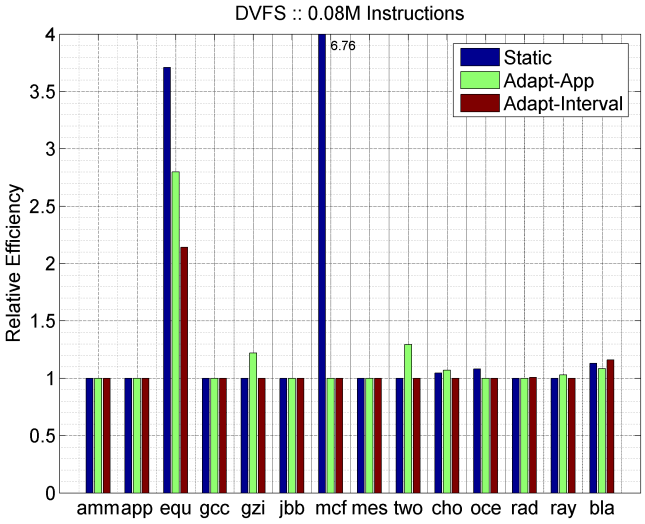

Figure 4.16: **Spatial Adaptivity and DVFS**. Additional efficiency from DVFS applied to various degrees of spatial adaptivity: none (Static), high-spatial/low-temporal (Adapt-App), and high-spatial/high-temporal (Adapt-Interval). Each bar is normalized to the corresponding level of spatial adaptivity without DVFS.

gains. This is due, in part, to the choice of efficiency metric. Recall that $bips^3/w$ is a voltage, frequency invariant metric. As shown in Equation (4.2), DVFS will produce gains for $bips^3/w$ only if it significantly improves pipeline throughput. Specifically, assume baseline throughput ipc_0, voltage V_0, and frequency f_0. Frequency and voltage scaling may impact throughput ipc_1 while scaling voltage and frequency by Δ. Since the Δ scaling cancels, only throughput effects remain.

$$\frac{bips_1^3}{w_1}\left(\frac{bips_0^3}{w_0}\right)^{-1} \propto \frac{ipc_1^3 \times f_0^3 \times \Delta^3}{V_0^2 \times f_0 \times \Delta^3} \times \frac{V_0^2 \times f_0}{ipc_0^3 \times f_0^3} = \left(\frac{ipc_1}{ipc_0}\right)^3 \tag{4.2}$$

Figure 4.16 shows, for benchmarks that benefit from DVFS, the greatest gains arise when DVFS is applied to a static, general-purpose, POWER4-like architecture with no resource adaptivity. For comparison, Figure 4.16 also includes the effects

of DVFS when applied to the comprehensive resource adaptivity of Table 4.8 at the application-level (81.92M instructions) and interval level (0.08M instructions). Each of the three bars is normalized to the static, application-adaptive, and interval-adaptive architecture with no DVFS, respectively. Benchmarks equake and mcf realize gains of 3.7x and 6.8x, respectively, relative to a static architecture with no DVFS.

The benefits of DVFS decrease as resource adaptivity increases. Resource adaptivity at the application-level significantly degrades the additional benefits of DVFS to 2.8x and 1.0x for equake and mcf. Equake efficiency gains are further degraded to 2.1x when sub-application, interval-level adaptivity is introduced. More generally, this data suggests DVFS becomes progressively less effective as an additional tuning parameter when combined with increasing spatial adaptivity. The bulk of *ipc* throughput gains are likely extracted from microarchitectural resource tuning and any additional throughput gains from voltage and frequency scaling on the tuned architecture are likely modest and incremental. Since Equation (4.2) indicate such gains are required to make DVFS effective, DVFS is likely to have a diminished role in a microarchitecture capable of comprehensive spatial adaptivity.

Overall, this comprehensive assessment of efficiency benefits from microarchitectural adaptivity reveals significant benefits from high temporal adaptivity. Furthermore, we find these gains are most accessible from a hardware substrate with comprehensive spatial adaptivity due to differing adaptive requirements across applications.

4.4 Related Work

Design optimization typically takes the form of exhaustive simulation, but a few researchers have considered more sophisticated techniques. Our contributions to multiprocessor heterogeneity and microarchitectural adaptivity build upon prior work, which relied primarily on simulation to consider constrained studies of their performance and power benefits.

4.4.1 Optimization

Eyerman, *et al.*, combine synthetic trace simulation with heuristics to search for global optima within a design space, producing a comprehensive survey of these heuristics for the microarchitectural design space [19]. Their analysis included gradient ascent, genetic algorithms, and tabu search. However, these simulations are specific to a given optimization problem since they simulate design points along a particular path taken to the estimate of a particular metric's optimum. In contrast, regression models require simulations per design space that may be formulated once and used in multiple studies. Furthermore, Eyerman, *et al.*, compute gradient ascent deficiency relative to results from other heuristics while we assess deficiency relative to exhaustive search with regression models. We study gradient ascent costs in greater detail, decomposing costs into trial, iteration, and neighbor counts.

4.4.2 Multiprocessor Heterogeneity

For homogeneous multiprocessors, Davis, *et al.*, suggest less aggressive in-order cores are performance optimal [12], and Huh, *et al.*, suggest larger out-of-order cores max-

imize throughput [31]. Both design spaces are relatively modest as experience and intuition were used to prune the space. In contrast, we consider the entire design space, enabling the discovery of potentially unexpected optima.

Kumar, *et al.*, identify heterogeneous cores constructed from existing core designs [38, 40] or cores designed from scratch using a modestly sized design space [39]. Design alternatives were evaluated with exhaustive simulation to illustrate the potential energy efficiency of heterogeneity. In contrast, we implement a more thorough analysis, considering heterogeneity trends as the number of design compromises increases and heterogeneity limits as we explore the full continuum between complete homogeneity and complete heterogeneity. Both analyses are intractable in simulation for a diverse, broadly defined design space.

4.4.3 Microarchitectural Adaptivity

There has been much prior work in microarchitectural adaptivity [3]. Broadly, prior work differs in their study of particular adaptive structures and/or control algorithms. These prior studies consider either high temporal, low spatial adaptivity or low spatial, high temporal adaptivity as shown in Figure 4.17. Each study limited the adaptive scope to include limited combinations of design values for at most two or three microarchitectural parameters and implemented a heuristic control algorithm to predict the best configuration from these limited choices. Most papers cite or imply computational costs as the cause for these restrictions. In contrast, we consider both high temporal and spatial adaptivity and leverage advances in statistical inference and optimization to control computational costs. This enables a significant expansion in the

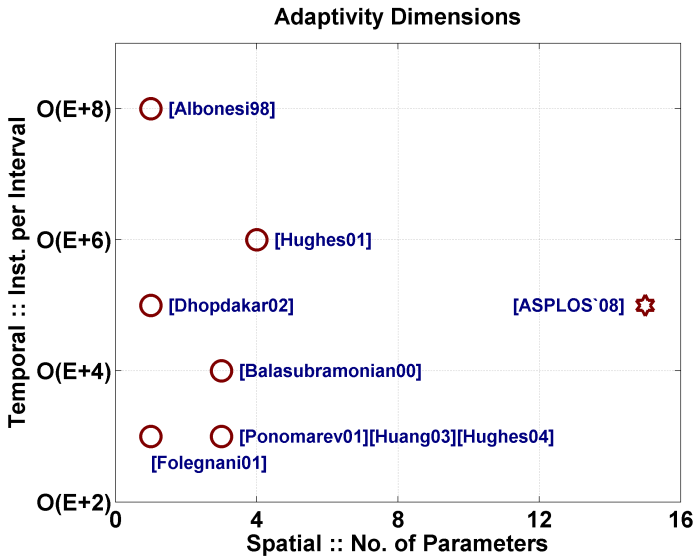

Figure 4.17: **Related Work in Adaptivity**. Prior studies considered low temporal or spatial adaptivity. In contrast, we consider much higher spatial adaptivity without compromising temporal adaptivity.

space of adaptive structures to include all major datapath and memory parameters. Furthermore, we search the design space to identify efficiency maximizing designs for each instruction interval, thereby providing an oracle-based analysis of potential gains from comprehensive spatial adaptivity.

Hardware Mechanisms. Many microarchitectural structures have been selected for adaptivity studies and, collectively, they represent all major design parameters. However, each study considers only a very small subset of these parameters. Albonesi, *et al.*, pioneers this work by describing buffering mechanisms that enable adaptivity in data caches and instruction queues [2]. Following work by Mai, *et al.*, and Balasubramonian, *et al.*, separately expand on these ideas for the memory hierarchy [5, 48]. Folegnani, *et al.*, consider optimizing issue logic and queues while Ponomarev, *et al.*, study adaptivity for the reorder buffer and other various queues [20, 61]. Adaptive

pipeline depth and width are also considered for energy and reliability [17, 30, 67]. Collectively, these prior works provide the basis for implementing many of the configurations we consider in our comprehensive evaluation of adaptivity. However, each of these prior works consider only a few parameters and, notably, often separate pipeline and cache adaptivity. In contrast, we consider adapting all major microarchitectural parameters simultaneously.

Control Algorithms. Control algorithms and heuristics have been proposed to trigger adaptive reconfiguration at various granularities. Ponomarev, *et al.*, use prior occupancy to predict future occupancy of various queues every 2,000 cycles [61]. Dhodapkar, *et al.*, assess the similarity of working sets to trigger reconfiguration [14]. Hughes, *et al.*, consider adapting resources for multimedia workloads based on frame type while Huang, *et al.*, consider adapting resources for code sections defined by subroutine boundaries [29, 30]. Collectively, prior work provides heuristics for dynamically determining optimal configurations and have been demonstrated for a limited number of adaptive parameters. These heuristics apply concepts from phase analysis to identify an effective degree of temporal adaptivity. Our framework may be extended to include phase information, but we currently use phase-oblivious adaptive intervals. Furthermore, in contrast to prior heuristics, we provide an oracle-based assessment of efficiency gains from comprehensive adaptivity. This optimistic analysis allow designers of control algorithms to quantify their heuristic effectiveness against a best case scenario.

4.5 Summary

This chapter applies the computational efficiency of statistical inference to comprehensive design space optimization, addressing fundamental limitations in current simulation methodology. Current approaches exhaustively evaluate designs in simulation to identify an optimum, an approach lacking scalability as design diversity increases design space sizes. Robust heuristic optimization is intractable as heuristics iteratively search for the optimum and simulations would be required for every iteration.

In contrast, statistical inference addresses limitations in both exhaustive and heuristic optimization. Exhaustively evaluating regression equations to identify an optimum is tractable for spaces with hundreds of thousands of points. Heuristic optimization becomes possible as regression equations replace simulation in the iterative loop, allowing robust implementations to use a large number of iterations and improve search effectiveness. These contributions in microarchitectural optimization enable qualitatively new analyses of fundamental design paradigms, including multiprocessor heterogeneity and microarchitectural adaptivity.

Multiprocessor heterogeneity is, more generally, a framework for identifying multiple design compromises from per workload optima. The concept of multiple compromises is the basis for special-purpose hardware accelerators, an emerging design priority. Given a broad range of workloads critical to performance, a heterogeneity analysis identifies common resource requirements within clusters or classes of workloads. These commonalities form the basis for any hardware accelerator targeting a given class of workloads. Heterogeneity analysis requires best known practices in modeling to tractably consider a comprehensive design space, optimization to quickly

identify per workload optima from the space, and clustering to effectively identify multiple design compromises between the optima.

Microarchitectural adaptivity is, more generally, a framework for reasoning about resource management to match workload dynamics. Identifying, at high frequency, optimal resource configurations from a large space of possible configurations requires efficient heuristic optimization. Not only must the heuristic identify optima from multi-billion point spaces, it must do so for every adaptive interval. This chapter quantifies trends and limits using oracle-driven resource management, but future work will require time series analysis to deliver efficiencies at run-time.

The optimization results from heterogeneity and adaptivity analyses are easily reconciled with designer intuition. Designers might easily sanity-check heuristic results that identify the k most efficient design compromises or identify the k most important design parameters for adaptivity. Confirming the results with intuition, designers more readily accept heuristic results and examine design implications. However, designers relying solely on intuition are unlikely to reveal trends and limits for these design paradigms as intuition is less effective as the scope of study increases. Thus, synergies between intuition and robust optimization are necessary.

Chapter 5

Conclusions and Future Directions

Contents

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

Statistical inference enables further research in pressing microarchitectural design questions. For example, inferential models could enable comprehensive analysis and optimization for integrated graphics processors or system-on-chip accelerator-based architectures. Designers will also need a fundamental understanding of the trade-offs between core complexity and interconnect complexity.

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

5.1 Summary of Themes and Results

Statistical Inference. The complexity of modern microarchitectural designs is captured using extensible software simulators and evaluated using efficient inferential models trained with those simulators. Thus, designers sacrifice little detail for significant gains in speed and tractability.

- **Comprehensive:** Scalable regression models capture performance and power trends for spaces with up to 15 parameters spanning 240 billion points.
- **Sparsely Trained:** As few as 500 training simulations, sampled sparsely and uniformly at random, are sufficient to construct effective regression models.
- **Accurate:** Regression models predict performance and power with median errors of 5 to 8 percent and maximum errors rarely exceeding 20 percent. Such

accuracy is sufficient for early stage design optimization.

- **Efficient:** Performance and power regression estimates are expressed as matrix-vector multiplication. Depending on model size and complexity, thousands of predictions per second are possible.

Comprehensive Design Characterization. Holistic design analysis is necessary given technology trends that increase design diversity. Leveraging the efficiency of inferential models, this dissertation considers design spaces several orders of magnitude larger than those tractable in detailed simulation. This more complete understanding is critical as Moore's Law provides increasingly abundant microarchitectural resources and designers must use these resources to deliver performance in a power efficient manner.

- **Pareto Frontiers:** Regression models identify pareto optima from a space with hundreds of thousands of designs, minimizing delay for a given power budget or minimizing power for a given delay target. Performance and power predictions for pareto optima, with median errors between 5 and 9 percent, are as accurate as those for the overall design space.

- **Contour Analysis:** Performance and power topologies are visualized to reveal bottlenecks within a workload and across workloads. Regression models can quickly generate all $\binom{p}{2}$ contour maps for p-dimensional design spaces.

- **Roughness Analysis:** High-order derivatives and multi-dimensional integrals are numerically computed for regression equations to quantify contour rough-

ness, ranking contours to focus designer attention to interesting design regions. Contours ranked rough graphically appear rough.

Robust Design Optimization. Increasing metric diversity requires effective design optimization. Exhaustive optimization is robust and is feasible for spaces with hundreds of thousands of designs. For larger multi-billion point spaces, heuristic optimization iteratively traverses a subset of the space to search for optima and are more effective with a larger number of iterations. Robust optimization enables previously intractable analyses of energy efficiency for emerging design paradigms, such as heterogeneous multiprocessors or adaptive microarchitectures.

- **Robust Optimization:** Iterative optimization heuristics require a large number of performance and power predictions across many iterative loops. Robust heuristics are implemented by replacing simulation with regression models within the loop. With a larger number of iterations, heuristics are more likely to identify the true optimum.

- **Multiprocessor Heterogeneity:** Multiple design compromises mitigate the performance and power penalties of those compromises. For nine workloads and four design compromises, heterogeneity delivers a 2.2x increase in $bips^3/w$ efficiency, only 8 percent less than the 2.4x increase from much higher degrees of heterogeneity with seven to nine compromises.

- **Microarchitectural Adaptivity:** Reconfigurable hardware provides resources to match application dynamics. Frequent adaptivity across a large space of configurations improves $bips^3/w$ efficiency by up to 5.3x (median 2.4x). From

a space of fifteen adaptive parameters, three parameters are often sufficient to achieve, on average, 77.3 percent of fifteen-parameter adaptive efficiency. However, the three most significant parameters differ across applications and are unknown during hardware design.

5.2 Future Directions

This dissertation emphasizes the performance and power trends within the microprocessor core. Once the core has been effectively modeled, interactions between cores in multiprocessor systems might be scalably analyzed. Beyond microarchitecture, statistical inference is highly extensible and might be broadly applied across the hardware-software interface. In both trajectories for future work, modular inference and composable models are necessary for scalable training and evaluation.

5.2.1 Modeling Methodology

Statistical Inference and Machine Learning. Other techniques in statistical inference may be applicable. A few of these techniques were described in Section 2.3.4. Quantifying and comparing the accuracy and computational efficiency of these techniques is an avenue for future work. Machine learning techniques seek to automate model construction, removing the user from the derivation process. Heuristics and algorithms drive the derivation, eliminating the need for user feedback. These automated approaches are easier to adopt and use, but tend to be less efficient. Comparing the effectiveness and efficiency of statistical inference and machine learning is an avenue for future work. Assessing the benefits of automated model construction

is more difficult but also necessary to balance computational efficiency and ease of construction.

Time Series Analysis. We focus primarily on predicting spatial characteristics, performing multivariate regression to model performance or power topology as a function of design parameters. In addition to this spatial dimension, computer system design often includes a temporal dimension where past system behavior may be indicative of future system behavior. Predicting events or behavior in time may require time series regression which identifies correlations in time (*e.g.*, if an event occurs now, another event is likely to occur t time steps from now). We may also require more sophisticated techniques, such as hidden Markov models (HMM's). HMM's consist of unobserved states and observed characteristics. HMM training must determine the nature of the unobserved states as well as the probability of observing a characteristic given a particular state. This formulation is attractive since many computer systems are fundamentally implemented as state machines and HMM's provide mechanisms to model the underlying state machine from observed system behavior. Thus, exploring techniques for time series analysis is a rich avenue for future work.

5.2.2 Multiprocessor Core Interaction

Homogeneous Multiprocessor Models. This dissertation primarily considers microprocessor cores without considering their interactions. In the future, such interactions should be considered for multiprocessor analysis. Interactions might arise from communication through shared memory, contention for shared resources, and synchronization for parallel workloads. Models for microprocessor cores and mechanisms to

account for interactions would provide a more thorough assessment of multiprocessor performance and power.

Building on this dissertation's contribution for uniprocessor core models, a potential multiprocessor framework might use a combination of uniprocessor, contention, and penalty models. Figure 5.1 illustrates such a framework, predicting n-core homogeneous multiprocessor performance for core design X. The uniprocessor model, as demonstrated in this dissertation, would predict delay as a function of the homogeneous core design X for each benchmark B_i in $B = \{B_1, \ldots, B_n\}$.

In parallel, the contention model predicts measures of interaction between the cores (*e.g.*, cache misses due to contention and interference) as a function of design parameters for shared resources $X_s \subset X$ when executing a particular combination of benchmarks B. The contention model would be trained by only a small number of multiprocessor simulations since the size of X_s is much smaller than the size of X. Given per-core delays and per-core contention metrics, the penalty model would estimate contention-adjusted delay for benchmark B_i executing in a multiprocessor with other benchmarks in the set B. This penalty model would also require only a small number of multiprocessor simulations since it is a small model which takes only core delay and contention as inputs. Thus, this approach requires a larger number of inexpensive uniprocessor simulations to capture most of the core design complexity while minimizing the number of more expensive multiprocessor simulations to capture only the interactions between cores. This modular approach abstracts each core using an inferential model and combines individual core performance using hierarchically composed penalizing models.

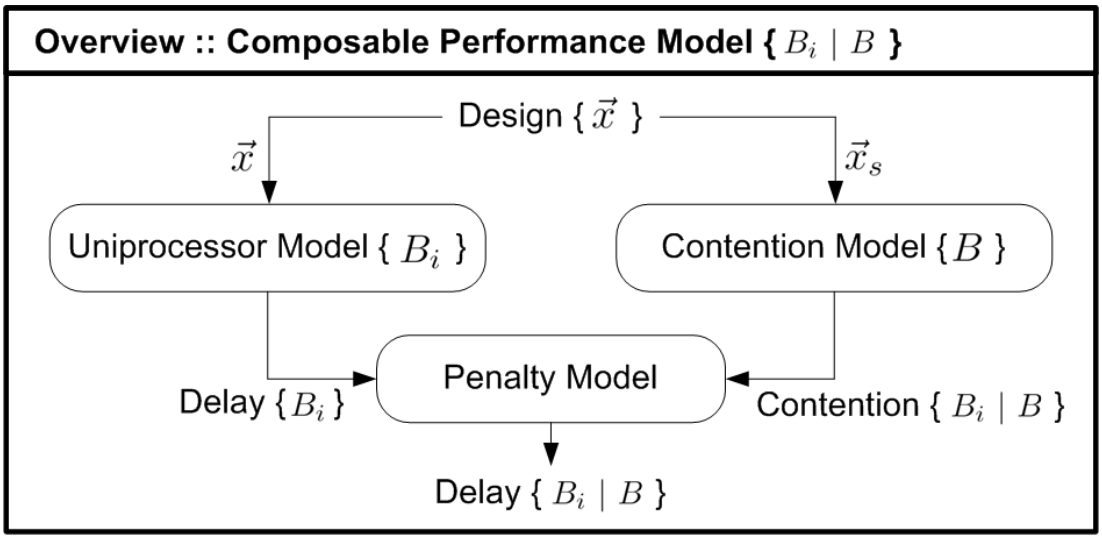

Figure 5.1: **Composable Multiprocessor Models**. Contention models adjust uniprocessor performance and power estimates with a penalty model. Uniprocessor models would be trained by core simulations while contention and penalty models would be trained by multi-core simulations.

Heterogeneous System-on-Chip Architectures. The modular framework for homogeneous multiprocessors extends naturally to heterogeneous multiprocessors and system-on-chip architectures by generalizing each of the three component models. Generalizing the uniprocessor model, we might replace the homogeneous core model with libraries of inferential models containing one model for each core type; each model would encapsulate the performance and power trends for each core's design space. The library would include models for both general-purpose and special-purpose cores. For general-purpose cores, we could consider trade-offs between complex and simple processing elements. Although models in this dissertation capture design trends for cores with out-of-order execution, future heterogeneous architectures may implement a combination of out-of-order and simpler, in-order execution to best serve combinations of latency-sensitive and thread-level parallel applications.

For special-purpose cores, the framework could consider both coarse-grained (e.g., GPU – graphics processing units, DSP – digital signal processing) and fine-grained (e.g., audio, video compression) accelerators. For an example of coarse-grained accelerators, the optimal GPU design (e.g., number of vector computing tiles and tile geometry) is impacted by the integration of a general-purpose core onto the same chip. Contention for off-chip bandwidth will fundamentally change GPU performance and power characteristics. In contrast, fine-grained accelerators for audio and graphics compression might expose tunable design parameters that include the size of data blocks for compression, the hash function, and the size of compressed output blocks. These parameters are highly dependent on bandwidth and memory utilization requiring, therefore, joint optimization with other cores on the chip. In contrast to coarse-grained variants, fine-grained accelerators most efficiently implement particular algorithms, but their contribution to overall system performance is limited since they may not effectively accelerate general-purpose code. The library of inferential models for the diverse core types found in system-on-chip architectures would be basic building blocks for a generalized framework.

The framework would generalize the contention model, replacing it with a broadly defined interaction model. This interaction model would estimate the performance and power overheads from inter-core communication, coherence, and contention. While the core models are likely constructed from detailed simulation, these interaction models may be constructed by less expensive means. Communication models, for example, might be derived analytically for very regular workloads (e.g., vector operations, compression) depending on whether code and data are communicated via

explicit message passing or shared memory. Coherence and contention models may require only functional simulation to, for example, extract cache access characteristics and may not require full microarchitectural simulation with detailed timing models. Given sufficient training data, these interaction models will estimate relevant statistics (e.g., cache access patterns, interconnect utilization) to characterize the interaction between various diverse cores. Lastly, the framework would generalize the penalty model, replacing it with a broadly defined adjustment model to calibrate individual core performance given various points of interaction.

5.2.3 Hardware-Software Interface

Statistical inference and regression modeling establishes a strong foundation for interdisciplinary research across the hardware-software interface. Inferential models may be constructed to encapsulate performance and power trends at each abstraction layer. Given such models, the challenge becomes creating clean interfaces between models to enable optimization across abstraction layers. Such interdisciplinary understanding enhances performance and power, for example, by understanding interactions between compiler optimizations and microarchitectural design or by exposing circuit implementation details to influence the design of microarchitectural blocks.

Applications. Application performance optimization is increasingly important as applications are ported to novel architectures. Effective performance tuning eases the transition by parameterizing the application with knobs that impact performance. The optimal knob configurations vary from platform to platform, requiring models to explore this space. For example, parameterized numerical methods and scientific

computing applications will expose knobs for the data decomposition (*i.e.*, blocks of work), the processor topology (*i.e.*, processor assignments to those blocks), and the algorithms (*i.e.*, numerical algorithms used for each block).

We have applied successfully statistical inference and machine learning to numerical methods, including semi-coarsening multigrid (SMG) and High-Performance LINPACK (HPL) [46]. SMG solves discretized three-dimensional equations while HPL solves a dense linear system. Regression models effectively capture performance trends as the working set size and processor topology vary in SMG. Similar models for HPL estimate performance from the matrix block size, processor assignment to those blocks, and algorithmic choices for the LU decomposition.

More generally, implementing parallel algorithms or applications is difficult. With parameterized or sketched implementations, designers focus on delivering functionality while expecting models and optimization heuristics to extract performance by identifying the optimal parameter configurations. Such an approach separates parallel functionality from parallel performance and eases programmer burden. Furthermore, these parameterized implementations provide software sufficiently flexible to adapt to the multiple possible trajectories into multiprocessor hardware design.

Compilers. Effective back-end compiler optimizations are critical to delivering performance, but the effects and interactions between individual optimizations are highly complex and non-intuitive. Identifying the best combination of optimization flags to activate is difficult. Iterative compilation techniques search the space of optimizations to optimize metrics, such as performance, energy, and code size.

Prior works prune the search space and apply heuristics, such as genetic algo-

rithms, to reduce search time [11, 37, 68]. Instead of measuring the impact for each set of optimizations, prior studies also propose analytical models to estimate performance [68, 72]. Statistical inference, machine learning, and heuristic optimization may be applied to improve search efficiency [1, 8, 9]. These predictive models encapsulate the performance trends in back-end compiler optimization like our regression models encapsulate performance and power trends in microarchitectural design.

Microarchitectural simulation currently relies on static instruction traces collected for particular hardware platforms. The same trace is used for every point in the design space. However, application performance for any design depends on the effectiveness of compiler optimizations for that design. In the future, hardware design should re-optimize the application as different hardware design points are considered. For example, designers might re-schedule instructions or apply different combinations of optimization flags. We must determine the impact of separately optimizing software and hardware, considering the possibilities of joint optimization to extract greater software performance from a hardware design space.

Circuits and Devices. Transistor tuning becomes increasingly important in nanoscale technologies. Not only must transistors be sized correctly, circuit delay analyses must account for process variations and statistical deviations from nominal sizes. Statistical inference and machine learning may be applied to capture relationships between circuit delays and device parameters (*e.g.*, transistor length, width, threshold voltage). Such predictive models might be trained with data from detailed circuit simulations and used for circuit tuning, statistical timing analysis, and Monte Carlo experiments to evaluate process variations.

Microarchitectural simulation relies on abstractions of the underlying circuits. However, the capabilities of any tunable microarchitectural block from Tables B.2–B.4 depend on their circuit implementations. Modular inference may provide opportunities to integrate circuit and microarchitectural analysis by first encapsulating their respective performance and power trends using separate inferential models and then composing these models for joint analysis and optimization.

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

Bibliography

[1] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. O'Boyle, J. Thomson, M. Toussaint, and C. Williams. Using machine learning to focus iterative optimization. In *CGO: International Symposium on Code Generation and Optimization*, Mar 2006.

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

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

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

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

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

[7] David Brooks, Pradip Bose, Viji Srinivasan, Michael Gschwind, Philip G. Emma, and Michael 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.

[8] J. Cavazos, G. Fursin, F. Agakov, E. Bonilla, M. O'Boyle, and O. Temam. Rapidly selecting good compiler optimizations using performance counters. In

CGO: International Symposium on Code Generation and Optimization, Mar 2007.

[9] J. Cavazos and M. O'Boyle. Method-specific dynamic compilation using logistic regression. In *OOPSLA: Conference on Object-Oriented Programming Systems, Languages, and Applications*, Oct 2006.

[10] C.Dubach, T.Jones, and M.O'Boyle. Microarchitectural design space exploration using an architecture-centric approach. In *MICRO: International Symposium on Microarchitecture*, December 2008.

[11] K.D. Cooper, P.J. Schielke, and D. Subramanian. Optimizing for reduced code space using genetic algorithms. In *LCTES: Conference on Languages, Compilers, and Tools for Embedded Systems*, May 1999.

[12] J.D. Davis, J. Laudon, and K. Olukotun. Maximizing CMP throughput with mediocre cores. In *PACT: International Conference on Parallel Architectures and Compilation Techniques*, September 2005.

[13] T.F. Devlin and B.J. Weeks. Spline functions for logistic regrssion modeling. In *Eleventh Annual SAS Users Group International Conference*, Cary, NC, 1986.

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

[15] P. Dubey and M. Flynn. Optimal pipelining. *J. Parallel and Distributed Computing*, 1990.

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

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

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

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

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

[21] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to NP-Completeness*. W.H. Freeman, 1979.

[22] G.H. Givens and J.A. Hoeting. *Computational Statistics*. Wiley, 2005.

[23] S. Gochman, R. Ronen, and et al. The Intel Pentium M processor: Microarchitecture and performance. *Intel Technology Journal*, 7(2), May 2003.

[24] P.J. Green and B.W. Silverman. *Nonparametric regression and generalized linear models: A roughness penalty approach*. Monographs on Statistics and Applied Probability, 1994.

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

[26] A. Hartstein and T.R. Puzak. The optimum pipeline depth for a microprocessor. In *ISCA: International Symposium on Computer Architecture*, May 2002.

[27] J.L. Henning. SPEC CPU2000: Measuring CPU performance in the new millenium. *IEEE Computer*, July 2000.

[28] M.S. Hrishikesh, K. Farkas, N.P. Jouppi, D.C. Burger, S.W. Keckler, and P. Sivakumar. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. In *ISCA: International Symposium on Computer Architecture*, May 2002.

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

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

[31] J. Huh, D.C. Burger, and S.W. Keckler. Exploring the design space of future CMPs. In *PACT: International Conference on Parallel Architectures and Compilation Techniques*, September 2001.

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

[33] C.R. Johns and D.A. Brokenshire. Introduction to the Cell Broadband Engine architecture. *IBM Journal of Research and Development*, 51(5), 2007.

[34] P.J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In *HPCA: International Symposium on High Performance Computer Architecture*, February 2006.

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

[36] T.S. Karkhanis and J.E. Smith. Automated design of application specific superscalar processors: An analytical approach. In *ISCA: International Symposium on Computer Architecture*, June 2007.

[37] P. Kulkarni, S. Hines, J. Hiser, D. Whalley, J. Davidson, and D. Jones. Fast searches for effective optimization phase sequences. In *PLDI: Conference on Programming Language Design and Implementation*, June 2004.

[38] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *MICRO: International Symposium on Microarchitecture*, December 2003.

[39] R. Kumar, D. Tullsen, and N. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *PACT: International Conference on Parallel Architectures and Compilation Techniques*, April 2006.

[40] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, and K. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *ISCA: International Symposium on Computer Architecture*, June 2004.

[41] S.R. Kunkel and J.E. Smith. Optimal pipelining in supercomputers. In *ISCA: International Symposium on Computer Architecture*, June 1986.

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

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

[44] B.C. Lee and D.M. Brooks. Efficiency trends and limits from comprehensive microarchitectural adaptivity. In *ASPLOS: International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2008.

[45] B.C. Lee and D.M. Brooks. Roughness of microarchitectural design topologies and its implications for optimization. In *HPCA: International Symposium on High-Performance Computer Architecture*, February 2008.

[46] B.C. Lee, D.M. Brooks, B.R. de Supinski, M. Schulz, K. Singh, and S.A. McKee. Methods of inference and learning for performance modeling of parallel applications. In *PPoPP: Symposium on Principles and Practice of Parallel Programming*, March 2007.

[47] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In *5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

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

[49] D. Markovic, V. Stojanovic, B. Nikolic, M. Horowitz, and R. Broderson. Methods for true energy-performance optimization. *IEEE Journal of Solid-State Circuits*, 39(8), August 2004.

[50] Sun Microsystems. Throughput computing: Changing the economics and ecology of the data center with innovative sparc technology. *Sun Microsystems (http://www.sun.com/processors/throughput/)*, November 2005.

[51] T. Mitchell. *Machine Learning*. WCB/McGraw Hill, 1997.

[52] A. Moore. Decision trees. *Tutorial Slides (http://www.autonlab.org/tutorials/dtree18.pdf)*.

[53] A. Moore. Eight more classic machine learning algorithms. *Tutorial Slides (http://www.autonlab.org/tutorials/bestregress11.pdf)*.

[54] A. Moore. Instance-based learning. *Tutorial Slides (http://www.autonlab.org/tutorials/mbl08.pdf)*.

[55] G. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8), April 1965.

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

[57] D.B. Noonburg and J.P. Shen. Theoretical modeling of superscalar processor performance. In *MICRO: International Symposium on Microarchitecture*, December 1994.

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

[59] M. Oskin, F.T. Chong, and M. Farren. HLS: Combining statistical and symbolic simulation to guide microprocessor designs. In *ISCA-27: International Symposium on Computer Architecture*, June 2000.

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

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

[62] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *ASPLOS: International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

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

[64] B. Sinharoy, R.N. Kalla, J.M. Tendler, R.J. Eickemeyer, and J.B. Joyner. Power5 system microarchitecture. *IBM Journal of Research and Development*, 49(4/5), July/September 2005.

[65] C.J. Stone. Comment: Generalized additive models. *Statistical Science*, 1:312–314, 1986.

[66] C.J. Stone and C.Y. Koo. Additive splines in statistics. In *Statistical Computing Section ASA*, Washington, DC, 1985.

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

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

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

[70] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *ISCA: International Symposium on Computer Architecture*, June 2003.

[71] J. Yi, D. Lilja, and D. Hawkins. Improving computer architecture simulation methodology by adding statistical rigor. *IEEE Computer*, Nov 2005.

[72] M. Zhao, B.R. Childers, and M.L. Soffa. Predicting the impact of optimizations for embedded systems. In *LCTES: Conference on Languages, Compilers, and Tools for Embedded Systems*, June 2003.

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

[74] 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*, August 2004.

[75] V. Zyuban and P. Strenski. Balancing hardware intensity in microprocessor pipelines. *IBM Journal of Research and Development*, 47(5/6), Oct/Nov 2003.

Appendix A

Simulator Framework

We use Turandot, a generic and parameterized, out-of-order, superscalar processor simulator [56]. Turandot is enhanced with PowerTimer to obtain power estimates based on circuit-level power analyses and resource utilization statistics [7]. 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 [74]. Power scales superlinearly as pipeline width increases, using scaling factors derived for an architecture with clustered functional units [73]. Cache power and latencies scale with array size according to CACTI [63]. We do not leverage any particular feature of the simulator in our models and our framework may be generally applied to other simulation frameworks.

Appendix B

Design Spaces

Turandot and PowerTimer originally modeled the POWER4-like design of Table B.1. The performance and power models have since been modified to consider a much larger range of designs.

Within this simulation environment, we consider several design spaces of varying size and complexity as described in Tables B.2–B.4. Each table identifies p sets or groups of parameters varied simultaneously. Parameters within a group are varied together to avoid fundamental design imbalances. The range of values considered for each parameter group is specified by a set of values, $S_1, \ldots, S_p$. The Cartesian product of these sets, $S = \prod_{i=1}^{p} S_i$, defines the entire design space. The cardinality of this product, $|S| = \prod_{i=1}^{p} |S_i|$, defines the design space size.

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

Table B.1: **POWER4 Baseline**. Superscalar, out-of-order microarchitectural design resembling the IBM POWER4.

	Set	Parameters	Measure	Range	$\|S_i\|$
S_1	Depth	Depth	FO4	9::3::36	10
S_2	Width	Width	issue b/w	2,4,8	3
		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 Purpoes (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 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	Main Memory	Main Memory Latency	cycles	70::5::115	10
S_9	Control Latency	Branch Latency	cycles	1,2	2
S_{10}	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_{11}	Floating-Point Latency	FPU Latency	cycles	5::1::9	5
		FP-Divide Latency	cycles	25::5::45	
S_{12}	Memory Latency	Load/Store Latency	cycles	3::1::7	5

Table B.2: **Design Space I**. Used for initial model derivation and proof of concept. $p = 12$, $|S| = 9.4\text{E+}8$.

	Set	Parameters	Measure	Range	$\|S_i\|$
S_1	Depth	depth	FO4	9::3::36	10
S_2	Width	width	issue b/w	2,4,8	3
		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	KB	16::2x::256	5
S_6	D-L1 Cache	d-L1 sache size	KB	8::2x::128	5
S_7	L2 Cache	L2 cache size	MB	0.25::2x::4	5

Table B.3: **Design Space II**. Used for design characterization and optimization where regression models are evaluated exhaustively for every point in the space. $p = 7$, $|S| = 3.8\text{E+}5$.

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

Table B.4: **Design Space III**. Used for design optimization where regression models are optimized with with iterative heuristics. $p = 15$, $|S| = 2.8\text{E+}11$.

Appendix C

Benchmarks

We report experimental results based on PowerPC traces of the benchmarks in Table C.1 [4, 27, 69]. The traces used in this study were sampled from the full reference 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 [32]. Our benchmark suite is representative of larger suites frequently used in the microarchitectural research community [60].

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

Table C.1: **Benchmarks**.