CPR: Composable Performance Regression for Scalable Multiprocessor Models

Benjamin C. Lee †
Microsoft Research
blee@microsoft.com

Jamison Collins, Hong Wang
Intel Corporation
{*hong.wang,jamison.d.collins*}*@intel.com*

David Brooks
Harvard University
dbrooks@eecs.harvard.edu

Abstract

Uniprocessor simulators track resource utilization cycle by cycle to estimate performance. Multiprocessor simulators, however, must account for synchronization events that increase the cost of every cycle simulated and shared resource contention that increases the total number of cycles simulated. These effects cause multiprocessor simulation times to scale superlinearly with the number of cores.

Composable performance regression (CPR) fundamentally addresses these intractable multiprocessor simulation times, estimating multiprocessor performance with a combination of uniprocessor, contention, and penalty models. The uniprocessor model predicts baseline performance of each core while the contention models predict interfering accesses from other cores. Uniprocessor and contention model outputs are composed by a penalty model to produce the final multiprocessor performance estimate. Trained with a production quality simulator, CPR is accurate with median errors of 6.63, 4.83 percent for dual-, quad-core multiprocessors. Furthermore, composable regression is scalable, requiring 0.33x the simulations required by prior regression strategies.

1. Introduction

Modern simulator infrastructure is ill-suited to handle current technology trends and the transition to multiprocessors. Cycle-accurate multiprocessor simulation inherently lacks scalability; simulation times increase superlinearly with the number of simulated cores and simulators are not easily parallelized [1]. Most challenges in microarchitectural simulation arise from the need for a high degree of synchronization. Detailed uniprocessor simulation proceeds cycle by cycle, tracking resource utilization to produce a detailed estimate of performance at the cost of long simulation times. Just as multiprocessors are built from multiple instantiations of uniprocessor cores, multiprocessor simulators often run multiple instantiations of a uniprocessor simulator. However, multiprocessor simulators must also account for inter-processor synchronization that increase the cost of every cycle simulated (*e.g.*, snoops for coherence) and shared resource contention that increases the total number of cycles simulated (*e.g.*, stalls due to eviction from shared cache lines). Collectively, these effects cause multiprocessor simulation times to scale superlinearly with the number of cores.

Recent advances in applying statistical inference to the microarchitectural domain have enabled qualitatively new capabilities in uniprocessor analysis [2], [3], [4], [5], [6]. Uniprocessor inferential models leverage best known practices in statistical inference for highly efficient simulation and analysis. Trained by simulating points sparsely sampled from the design space, these models are computationally efficient surrogates for cycle-accurate simulation, which capture mappings between design parameters and design metrics. However, these same techniques are unlikely to scale as we consider multiprocessor performance models. While a few hundred training simulations are tractable fixed costs for uniprocessor modeling, they are prohibitively expensive for multiprocessor modeling due to superlinear increases in simulation time. Thus, multiprocessor regression models derived from multiprocessor simulations alone are impractical.

Exacerbating these greater multiprocessor simulation times, multiprogrammed workloads produce combinatorial growth in the number of possible workload combinations. The number of such combinations will increase rapidly with the number of cores. Taking the 18 benchmarks in this paper for example, we are confronted with $C_2^{18} = 153$ dual-core and $C_4^{18} = 3060$ quad-core workload combinations. Furthermore, for industrial microprocessor design, 18 benchmarks are only a small representative set from a much more comprehensive study list with over 500 benchmarks. For 500 benchmarks, we could potentially analyze $C_2^{500} = 1.3E{+}05$ dual-core and $C_4^{500} = 2.6E + 09$ quad-core workload combinations. Thus, in addition to ensuring scalability as the number of cores increases, an effective multiprocessor modeling framework must also ensure the marginal cost of model training for each combination of workloads is computationally tractable.

To address these fundamental challenges in multiprocessor simulation times, we propose *composable performance re-*

† *This work was done while B. Lee interned at Intel and studied at Harvard.*

gression (CPR). CPR extends prior successes in uniprocessor statistical inference to the multiprocessor domain while ensuring training costs remain tractable. The framework controls training costs by abstracting the uniprocessor, taking the core as the primary determinant of multiprocessor performance and considering shared resource contention as a secondary penalizing effect. Within this framework, uniprocessor models are the primary building blocks supported by secondary contention and penalty models. Such a framework is highly efficient, building on the information captured by relatively inexpensive uniprocessor models while requiring a far fewer number of multiprocessor simulations to train contention and penalty models to predict multiprocessor performance.

We define *scalability* as the rate of increase in training costs when the number of modeled cores increases. *Composable* models are trained and evaluated separately but composed to estimate a final response. CPR uses composable models to deliver scalability for multiprocessor performance estimates. The following summarizes the key contributions of this work:

1) **Industrial Infrastructure:** We demonstrate regression models' effectiveness for the Intel Core microarchitectural family, using industry-strength simulators actively employed in product development. We construct these models for a broad array of workloads ranging from multimedia to server applications. (Section 2)
2) **Uniprocessor Model:** We first derive uniprocessor regression models, demonstrating the accuracy of these basic building blocks before applying them in the multiprocessor model. Furthermore, we demonstrate the application of these models to evolutionary design optimization, re-tuning design parameters after fundamentally new microarchitectural features are added between consecutive product generations. (Section 2.3)
3) **Multiprocessor Model:** We propose composable performance regression that includes uniprocessor, contention, and penalty models. The uniprocessor model predicts the baseline performance of each core while the contention models predict interfering accesses to shared resources from other cores. Uniprocessor and contention model outputs are composed in a penalty model that produces the final multiprocessor performance prediction. (Section 3)
4) **Case for Scalable Multiprocessor Models:** In addition to being individually accurate, the uniprocessor, contention and penalty models combine to predict dual-core and quad-core performance with median errors of 6.63 and 4.83 percent, respectively. Composable regression is a scalable, efficient approach for constructing these multiprocessor models. Training costs for proposed composable regression are 0.33x those for naively constructed models as the number of multiprocessor cores increases. (Section 4)

Collectively, this work establishes a rigorous foundation for efficiently extending advances in statistical machine learning for uniprocessor design into the multiprocessor domain.

2. Methodology and Background

We develop a scalable and efficient simulation paradigm for multiprocessor design that defines a comprehensive design space, simulates sparse samples from the space, and constructs regression models for performance prediction. We extend prior work for uniprocessor models to those for multiprocessors [6].

2.1. Regression Modeling

Microarchitecture designers often use performance models to predict a performance response y as a function of design parameter values $\vec{x}$. Such models may be expressed as $y = F(\vec{x})$ where $F(\vec{x})$ may be a detailed, cycle-accurate simulator or an empirical model fit using simulated data [7]. While evaluating $F(\vec{x})$ using detailed simulation is the most widely taken approach, the significant computational costs of simulation often hinder the design process. These difficulties have led to computationally efficient surrogates for detailed simulators that predict the response as $y = \hat{F}(\vec{x}) + e$, where $\hat{F}(\vec{x})$ is an approximation of the simulator response and e is the approximation error.

We construct the surrogate $\hat{F}(\vec{x})$ by deriving spline-based regression models from a sparse sampling of the design space [8], [6]. These models express the predicted response as $y = \hat{F}(\vec{x}) = \alpha + \vec{\beta}^T S(\vec{x}) + e$ where α is the model intercept, $\vec{\beta}$ is a vector of regression coefficients and $S(\vec{x})$ is a *spline function* that produces piecewise polynomials. This spline transformation is applied to each x_i in $\vec{x}$ by dividing the domain of x_i into intervals joined at intersections called *knots*. Separate polynomials are fit to data within each interval to produce a piecewise polynomial. For example, a cubic spline on x_i with three knots at a, b, and c is given by Equation (1) where $(u)_+ = u$ if $u > 0$ and $(u)_+ = 0$ otherwise.

$$\begin{aligned} S(x_i) &= \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 (x_i - a)_+^3 \\ &\quad + \beta_5 (x_i - b)_+^3 + \beta_6 (x_i - c)_+^3 \end{aligned} \tag{1}$$

Within this framework, interactions between predictors of performance are captured by product terms using domain-specific knowledge. For example, pipeline depth x_1 interacts with cache sizes x_2 since depth determines pipeline sensitivity to cache misses. This interaction would be captured by a product term $x_3 = x_1 x_2$. These interactions are applied after spline transformations. We prune the number of terms from an interaction $S(x_i)S(x_j)$ by removing doubly non-linear terms (*i.e.*, keep only product terms with at least one linear factor).

Digital Home		
1	audio	audio conversion [9]
2	video	video compression [9]
3	photo	photoshop album
Games		
4	unreal	Unreal Tournament
5	halflife	Half-Life, modified Quake engine
6	homeworld	Homeworld, three-dimensional movement
Multimedia		
7	mentalray	rendering, ray tracing
8	painter	raster graphics package
9	tachyon	ray tracing
Office		
10	outlook	personal information manager
11	access	relational database management system
12	excel	spreadsheet application
Productivity		
13	md2	OpenSSL cryptographic hash function
14	encrypt	file encryption [9]
15	flash	multimedia player
Server		
16	specweb	web server [10]
17	tpcc	on-line transaction processing [11]
18	specjapp	J2EE 1.3 application servers [12]

Table 1. Benchmarks

2.2. Simulation Framework

We use an execution-driven, cycle-accurate, micro-operation (uop) based IA-32 simulator that models a superscalar, out-of-order microprocessor belonging to the Intel Core microarchitectural family. The simulated multiprocessor implements a ring-based interconnect. Each node of the ring is comprised of a superscalar, out-of-order execution IA-32 core and private L1, L2 caches. Each node also contributes capacity to an L3 cache shared among all nodes using a snoopy coherence protocol. The simulator implements a detailed memory subsystem that fully models interconnect topologies and any associated contention. Both uniprocessor and multiprocessor models are validated against product hardware.

The simulated microprocessor executes inputs called *Long Instruction Traces* (LIT's). A LIT is not actually a trace, but a checkpoint of processor and memory state used to initialize an execution-based performance simulator. A LIT also includes a list of *injections*, which are system interrupts needed to simulate events like direct memory accesses (DMA). Since the LIT includes a complete snapshot of system memory, we are able to simulate both user and kernel instructions, as well as wrong-path instructions. We report experimental results based on LIT's of the benchmarks in Table 1. These benchmarks represent a broad range of application areas ranging from the digital home to the server.

2.3. Design Space

Table 2 defines a microprocessor design space with fifteen groups of parameters that characterize key microarchitectural structures. Parameters within each group are varied together to avoid fundamental design imbalances (*e.g.*, the L/S buffer and reorder buffer should vary together in set S_6). The Cartesian product of these sets, $S = \prod_{i=1}^{15} S_i$, defines the overall design space of approximately 4.3 billion points. Using an effective spatial sampling technique [6], which decouples design space size from the number of simulations needed to identify design trends, we find 300 samples obtained uniformly at random from the design space are sufficient for accurate regression.

	Set	Parameters	Range	$\lvert S_i \rvert$
S_1	Branch Pred.	target buffer entries	256 :: 2x :: 1024	3
S_2		target buffer assoc	2 :: 2x :: 16	4
S_3	Stride Pred.	table entries	128 :: 2x :: 512	3
S_4	Loop Detector	table entries	4 :: 2+ :: 12	5
S_5	Mem Disamb.	table entries	32 :: 2x :: 512	5
S_6	Out-of-Order	L/S buffer entries reorder buffer entries reserv. stations	24 :: 8+ :: 96 72 :: 12+ :: 180 28 :: 4+ :: 64	10
S_7	Fill Buffer	buffer entries	4 :: 2+ :: 12	5
S_8	Data TLB	table entries	128 :: 2x :: 512	3
S_9		table assoc	1 :: 2x :: 8	4
S_{10}	Inst Cache	victim cache entries	4 :: 2+ :: 12	5
S_{11}	Data Cache	size (KB)	8 :: 2x :: 128	5
S_{12}		assoc	2 :: 2x :: 16	4
S_{13}	L2 Cache	size (KB)	256 :: 256+ :: 1024	4
S_{14}		assoc	2 :: 2x :: 16	4
S_{15}	L3 Cache	size (MB)	1.0 :: 0.5+ :: 3.0	5

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

Uniprocessor models are the basic building blocks for our multiprocessor modeling framework. Regression models enable rapid design space exploration via fast, accurate predictions. Once constructed, model usage depends very much on the design process. While academic research might explore a large design space and consider each design without bias, industrial product development usually introduces innovative features to the previous product generation to improve particular design metrics (*e.g.*, performance, power, area, cost).

Microprocessor development is driven by regular, alternating advances in silicon and microarchitecture [13]. To ensure silicon advances translate into better user experiences, the microprocessor industry must produce evolutionary improvements across each product generation. These improvements might be delivered via fundamentally new algorithmic or policy features and/or a better allocation of microarchitectural resources. However, new features usually introduce subtle, unknown, or unintended interactions with existing microarchitectural resources, thereby complicating product design. For example, a new performance-enhancing feature might allow designers to meet a particular performance target with fewer resources, thereby saving power and area. Furthermore, designers might add several new features into a single product generation, each with potentially conflicting influences on a structure's optimal size.

We evaluate these scenarios through a case study, demonstrating our regression models can be effectively applied to the industrial, evolutionary design process. In particular, we

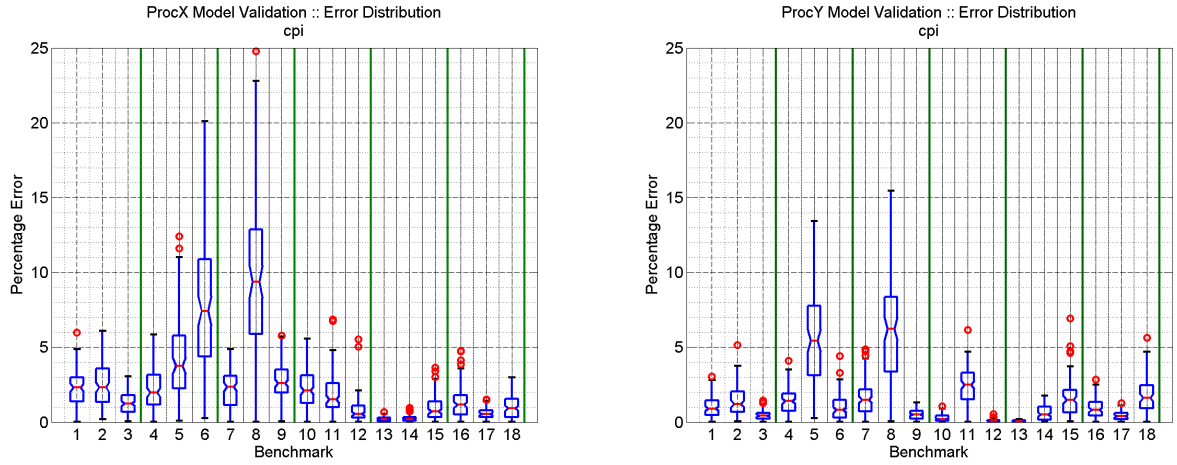

Figure 1. Uniprocessor Model: Error distributions for models of design spaces built around two consecutive microprocessor generations ProcX (L) and ProcY (R).

consider two consecutive generations within the same IA-32 Core Microarchitecture family, which we refer to as ProcX and ProcY. ProcY is an evolutionary progression from ProcX and includes several new microarchitectural features. Aside from these new features, ProcX and ProcY share the same design space of Table 2. We use per benchmark regression models to re-optimize and re-balance the design after these new features are added. Such a case study illustrates the importance of holistic design analysis in evolutionary product development.

2.4. Uniprocessor Regression Model Validation

We construct regression models for the uniprocessor design space using 300 randomly collected training samples. Figure 1 illustrates error distributions when validated against simulation for 50 randomly collected validation points. Sensitivity studies indicate this validation set is sufficiently large and error distributions do not change significantly with additional samples. These boxplots display location (median) and dispersion (interquartile range), identify possible outliers, and indicate the symmetry or skewness of the distribution.[1] These figures demonstrate median errors of 1.43 and 0.77 percent for ProcX and ProcY performance prediction, respectively. Maximum errors and statistical outliers rarely exceed 10 percent.

1. Boxplots are constructed by (1) horizontal lines at median and at upper, lower quartiles, (2) vertical lines drawn up/down from upper/lower quartile to most extreme data point within 1.5 IQR of upper/lower quartile where IQR is the interquartile range between first and third quartile, and (3) circles to denote outliers.

2.5. Case Study: Evolutionary Design Optimization

While prior academic work has demonstrated regression models can be effective for designing microprocessors from a blank slate [14], industry is more likely to implement evolutionary enhancements by adding new microarchitectural enhancing features to existing designs, thereby extending a processor family. For example, Intel's Yonah, Dothan, Merom, and Penryn are four consecutive evolutionary generations of the Intel Core Microarchitecture family. In such a design scenario, regression models enable a comprehensive re-balancing of the microprocessor design. Re-balancing is the process of taking optimal design values from one generation, accounting for new features in the next generation by constructing new models, and determining how previously optimal values change given these new features. For this study, we evaluate three categories of microarchitectural enhancements:

- **Improved Front-End:** An improved front-end will increase fetch throughput and the number of in-flight instructions. Improved branch prediction might be delivered by larger history tables, thereby increasing resource requirements. Alternatively, better prediction schemes might make better use of fewer resources, enabling designers to reduce resource sizes and save power with negligible performance penalties.
- **Improved Memory Subsystem:** Improved cache line replacement or prefetch policies will impact cache hierarchy design. For example, improved loop detection and memory stride prediction may enable more effective use of L1 instruction and data caches, respectively.
- **Improved Out-of-Order Execute:** Improved execution units will impact queue and buffer occupancy throughout

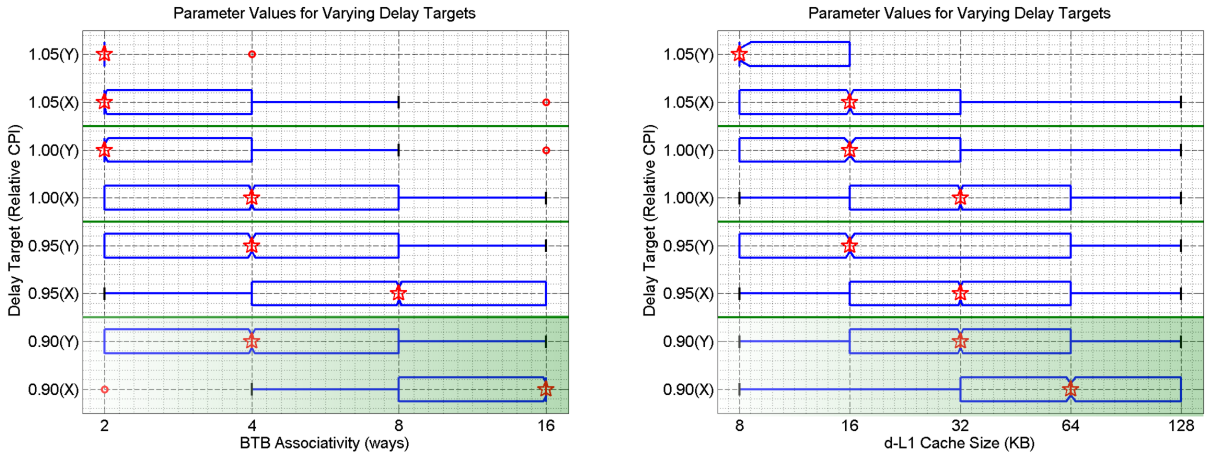

Figure 2. BTB (L) and L1 data cache (R) design values that meet various delay targets from ProcX (without new features) and ProcY (with new features). Note log-scaled x-axis.

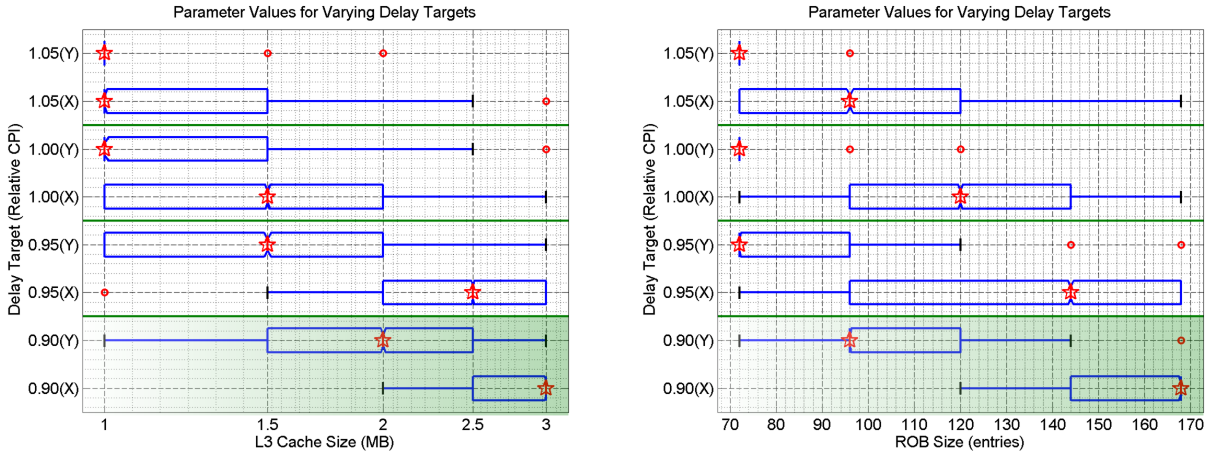

Figure 3. L3 cache (L) and ROB (R) design values that meet various delay targets from ProcX (without new features) and ProcY (with new features). Note log-scaled x-axis for L3 cache.

the pipeline. For example, improved memory disambiguation will reduce the average time a load must wait before issuing. Such a feature enhancement may reduce average occupancy of the memory order buffer while improving instruction throughput.

Suppose designers are given a range of delay targets, defined as CPI numbers relative to ProcX baseline performance.[2] To re-balance the design after introducing new microarchitectural features, we explore the two design spaces of ProcX and ProcY, identifying designs that satisfy particular targets before and after the addition of new features. Limited sensitivity studies are insufficient since they do not identify specific combinations of design values that meet a particular target. For example, one-dimensional sensitivity studies of L1, L2 and L3 cache sizes cannot identify all combinations of cache sizes that meet a delay target and, furthermore, they cannot identify cases where one parameter value increases and another decreases with zero net performance impact.

Instead of one-dimensional sensitivity, we use regression models for more comprehensive optimization, first identifying all parameter value combinations that meet each CPI delay target and then assessing salient trends within these combinations. Since regression models may identify multiple designs that satisfy a given CPI delay target, we use boxplots to show the range of parameter values represented in these designs. Figures 2–3 illustrate the impact of new microarchitectural

2. Designers often identify targets for performance improvements from one generation to the next.

features on design optimization. Parameter values are shown on the x-axis and CPI delay targets are shown on the y-axis.

Figure 2L illustrates reductions in branch target buffer (BTB) associativity after new microarchitectural features are added to the front-end (*e.g.*, better branch predictor). Of the designs meeting a delay target of 0.90 (shaded in green), median associativities for ProcX and ProcY designs are 16 and 4, respectively. For every delay target, ProcY designs meet the same target as ProcX designs but with fewer BTB ways. Fewer BTB ways translate into fewer buffer entries, reduced comparator logic, and potential power savings for a given delay target. These savings are realized only by re-optimizing the design after adding a front-end enhancing feature, such as a better branch predictor.

Similarly, Figure 2R illustrates potential reductions in the L1 data cache size due to an improved back-end (*e.g.*, better prefetch). For various delay targets, the median cache size for ProcY designs is consistently half the median size for ProcX designs (*e.g.*, 32 KB versus 64 KB for 0.90 target; shaded in green). These results suggest memory-enhancing features, such as more intelligent prefetching, might allow designers to attain the same performance despite a smaller cache. Furthermore, more effective L1 cache usage will likely impact the whole cache hierarchy as shown by similar trends for the L3 cache in Figure 3L. Thus, designers might meet a given delay target using fewer resources, thereby saving power, by re-optimizing the cache hierarchy when more efficient cache policies are added.

Figure 3R considers a scenario for the reorder buffer (ROB) where several new microarchitectural features might exert competing influences. A more effective front-end (*e.g.*, branch predictor) might increase the number of in-flight instructions, potentially increasing the optimum ROB size. However, a more effective memory subsystem (*e.g.*, prefetcher) might reduce the number of long latency memory stalls as loads hit in the L1 data cache more often. Fewer memory stalls increase instruction commit throughput, potentially decreasing the optimum ROB size. Figure 3R shows ProcY designs meet delay targets with much lower ROB occupancy than ProcX designs, suggesting improvements to the memory subsystem dominate and recommending a net reduction in ROB size. Relying solely on intuition, the net impact on ROB size is not obvious. Regression models, however, allow us to explore the design space and account for interactions between new microarchitectural features and various resources.

3. Composable Performance Regression

Unique challenges in multiprocessor simulation for multi-programmed workloads prevent us from naively applying statistical modeling methodologies previously found effective in the uniprocessor domain [2], [3], [5], [6]. Simulating

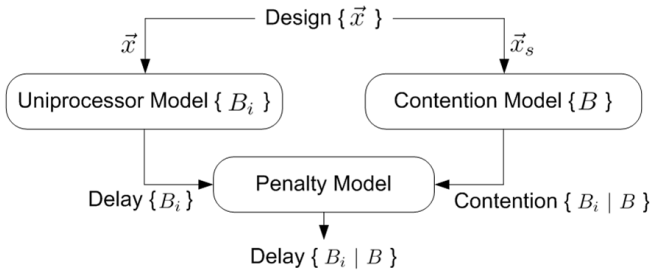

Figure 4. CPR overview of components, interactions.

sampled training designs becomes significantly more expensive when these samples are drawn from the multiprocessor design space. Furthermore, deriving separate models for each combination of workloads each with its own set of training data is intractable due to rapid combinatorial growth in the number of possible multiprogrammed workloads. To address these fundamental challenges, we take the canonical approach of using the uniprocessor core as the primary determinant of multiprocessor performance and consider shared resource contention as a secondary penalizing effect.

3.1. Overview

Composable regression addresses the unique challenges of multiprocessor performance modeling as summarized in Figure 4. Let $\vec{x}$ specify the vector of parameter values characterizing a design and $\vec{x}_s \subset \vec{x}$ specify the sub-vector of these design parameters that characterize resources shared between processors. Consider a set of benchmarks $B = \{B_1, \ldots, B_n\}$ executing on an n-core multiprocessor. The framework iteratively predicts the performance of each benchmark in B. Consider iteration i where we take B_i as the benchmark executing on the core of interest while simultaneously contending with the other benchmarks $B_{\neq i} = \{B_1, \ldots, B_{i-1}, B_{i+1}, \ldots, B_n\}$. As shown in Figure 4, this framework predicts the baseline uniprocessor performance of B_i executing on design $\vec{x}$. In parallel, the framework predicts contention indicators for B_i when it contends with $B_{\neq i}$ for shared resources $\vec{x}_s$. A penalty model computes a linear combination of baseline performance and contention indicators to estimate the performance of B_i when contending with $B_{\neq i}$ benchmarks in a symmetric multiprocessor with n cores of design $\vec{x}$.

3.2. Training

Figure 5 illustrates uniprocessor model construction. This model requires N_{uni} samples from the full design space. A uniprocessor simulator takes these inputs to provide observed CPI delay values $D_i(\vec{x})$ for a particular benchmark B_i. These delay values are fit to the design values $\vec{x}$ with

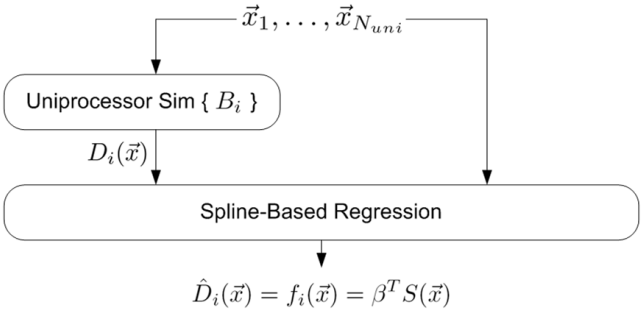

Figure 5. Uniprocessor training with spline-based regression on full parameter space.

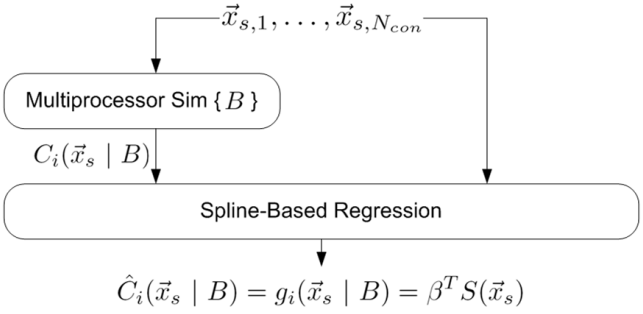

Figure 6. Contention training with spline-based regression on shared resource space.

restricted cubic splines (*i.e.*, piecewise cubic polynomials denoted by transformation S), producing a surrogate model for the uniprocessor simulator. This surrogate is the same model discussed and applied in Section 2.3. The model uses three knots for each design parameter of Table 2. Interactions are specified between caches sizes in adjacent levels of the memory hierarchy. The resulting model estimates the uniprocessor CPI delay of benchmark B_i as a function of transformed design values: $\hat{D}_i(\vec{x}) = f_i(\vec{x})$.

Figure 6 illustrates contention model construction. This model requires N_{con} samples from the space of shared resources $\vec{x}_s$, a subset of the full design space. $N_{con} < N_{uni}$ since the contention model considers only a subset of the full parameter space and thus produces a smaller model requiring fewer training simulations. Although cores share only the L3 cache, we consider L1 data, L2, and L3 caches in $\vec{x}_s$ since the design of higher cache levels determine L3 cache traffic. An n-core multiprocessor simulator takes these shared design values to provide observed contention indicators $C_i(\vec{x}_s \mid B)$ for a particular combination of benchmarks B. Although we use cycle-accurate simulation to produce these contention indicators, trace simulation may be sufficient (*e.g.*, cache simulation), further reducing contention model training costs relative to uniprocessor training costs.

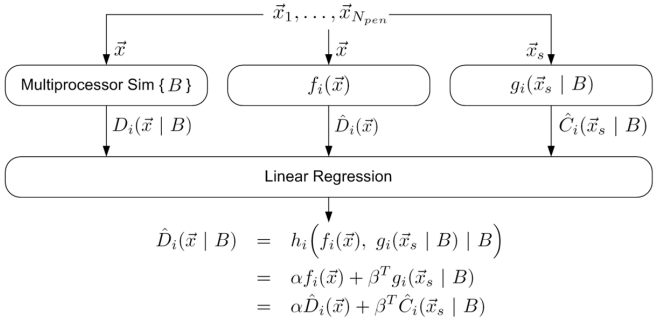

Figure 7. Penalty training with linear regression on full parameter space.

We define $C_i(\vec{x}_s \mid B)$ as a vector of three contention indicators: d-L1 misses, L2 hits, and L3 hits. We find accurate contention models are more easily constructed for the three we choose. Intuitively, these metrics are suitable contention indicators since they correlate with average memory access latencies and the memory intensity of the workloads. These contention indicators are fit to design values of shared resources $\vec{x}_s$ with restricted cubic splines, producing a surrogate for multiprocessor simulations. The regression model uses four knots for the d-L1, L2 caches and five knots for the L3 cache. Additional spline flexibility is assigned to the L3 cache due to its significance as the primary point of contention in our simulated multiprocessor. Interactions are specified between cache sizes in adjacent levels of the memory hierarchy. Such interactions capture locality effects and are necessary for inclusive caches. The resulting model estimates multiprocessor contention seen by benchmark B_i executing with $B_{\neq i}$ as a function of shared design values: $\hat{C}_i(\vec{x}_s \mid B) = g_i(\vec{x}_s \mid B)$.

Figure 7 illustrates penalty model construction. This model requires N_{pen} samples from the full design space $\vec{x}$. A multiprocessor simulator takes these inputs to provide observed multiprocessor delay values $D_i(\vec{x} \mid B)$ for a particular benchmark B_i executing with $B_{\neq i}$. Unlike the uniprocessor and contention models, which are constructed using simulation and design parameter values, the penalty model is constructed using simulation and outputs from the other two models. Specifically, the multiprocessor delay values are fit to predictions from the uniprocessor and contention models (*i.e.*, $\hat{D}_i(\vec{x})$ and $\hat{C}_i(\vec{x}_s \mid B)$, respectively). The resulting regression model predicts the delay of benchmark B_i when executing with other benchmarks in B as a function of design parameters. This prediction is made via a uniprocessor delay model $f_i(\vec{x})$ and multiprocessor contention model $g_i(\vec{x}_s)$. The overall result is a composed regression model: $\hat{D}_i(\vec{x} \mid B) = h_i\Big(f_i(\vec{x}),\ g_i(\vec{x}_s \mid B) \mid B\Big)$.

In contrast to uniprocessor and contention models, the penalty model is less complex and, therefore, less expensive to construct. Quantifying model complexity by the number of terms in the model (*i.e.*, number of regression coefficients that must be fitted), the penalty model has complexity advantages from taking fewer predictors and not performing spline transformations. The penalty model takes only four predictors: baseline uniprocessor performance $D_i(\vec{x})$ and three contention metrics $\hat{C}_i(\vec{x}_s \mid B)$: d-L1 misses, L2 hits and L3 hits. In contrast, the uniprocessor model takes fifteen predictors, one for each design parameter in Table 2. Furthermore, spline transformations are applied to each of these fifteen predictors, which significantly increases the number of terms in the uniprocessor model as shown in Equation (1). Thus, composable regression models package uniprocessor performance and multiprocessor contention information for a fifteen-element design vector $\vec{x}$ into a more compact four-element vector $\{D_i(\vec{x}), \hat{C}_i(\vec{x}_s \mid B)\}$ for more efficient, complexity-effective linear regression. The efficiency and complexity advantages translate into fewer multiprocessor simulations required for model construction. A smaller model that fits fewer coefficients will require less data for training.

3.3. Prediction

Consider an n-core symmetric multiprocessor executing benchmarks $B = \{B_1, \ldots, B_n\}$. Given the three components of the composable regression model, we iteratively evaluate the performance of each core i for design $\vec{x}$.

1) **Uniprocessor Prediction:** Evaluate uniprocessor performance of B_i executing for design $\vec{x}$. Compute $\hat{D}_i(\vec{x}) = f_i(\vec{x})$.
2) **Contention Prediction:** Evaluate multiprocessor contention of B_i executing with $B_{\neq i}$ on a symmetric multiprocessor for a particular configuration of shared resources $\vec{x}_s$. Compute $\hat{C}_i(\vec{x}_s \mid B) = g_i(\vec{x}_s \mid B)$.
3) **Multiprocessor Prediction:** Evaluate multiprocessor performance of B_i executing with $B_{\neq i}$ on a symmetric multiprocessor for core designs $\vec{x}$. Compute $\hat{D}_i(\vec{x} \mid B) = h_i\Big(f_i(\vec{x}),\ g_i(\vec{x}_s \mid B) \mid B\Big)$.

This process is repeated for every benchmark executing on the multiprocessor to get per core CPI delays.

4. Multiprocessor Model Evaluation

We illustrate the effectiveness of composable regression models, showing these models accurately estimate multiprocessor performance. Furthermore, model construction costs are scalable in contrast to intractable exhaustive simulation and inefficient naive regression.

Dual-Core Benchmarks				
Set	(1)	(2)		
1	painter	homeworld		
2	access	mentalray		
3	specjapp	specweb		
4	homeworld	tachyon		
5	dense	flash		
Quad-Core Benchmarks				
Set	(1)	(2)	(3)	(4)
1	dense	excel	flash	md2
2	video	specjapp	specweb	tachyon
3	excel	homeworld	audio	unreal
4	outlook	encrypt	halflife	homeworld
5	painter	mentalray	outlook	encrypt

Table 3. Multicore Benchmarks

4.1. Accuracy Analysis

We validate each component in the composable regression framework against detailed simulation before evaluating its overall accuracy. We construct each model by sparsely simulating points sampled uniformly at random from the design space. We train these three model components with $N_{uni} = 300$ and $N_{con} = N_{pen} = 50$. We separately simulate another 50 randomly sampled points for validation. This is a sufficiently large validation set; further increasing the validation set size does not significantly change the error distribution. We evaluate these validation points for five sets of benchmark combinations sampled uniformly at random from the space of possible combinations as shown in Table 3. We validate against detailed multicore simulators used by product teams. The product development infrastructure used in this study scales up to four cores.

Figure 1R in Section 2.3 illustrates uniprocessor model accuracy. We build the composable framework to model microprocessor cores from the ProcY design space. As described in Section 2.3, these models achieve median errors of 0.77 percent relative to detailed simulation. In the context of our composable framework, these errors refer to the relative difference between simulation and the uniprocessor regression model $f_i(\vec{x})$.

Figures 8–9 illustrate the error distributions for contention indicators predicted by the model $g_i(\vec{x}_s \mid B)$. Figure 8 and Figure 9L present errors for the dual-core case. Contention indicators are predicted across all benchmarks with median errors of 3.17, 2.46, and 1.30 percent for d-L1 misses, L2 hits and L3 hits, respectively. Median errors for each benchmark are usually below 5 percent and always below 10 percent. Figure 9R presents L3 hit prediction errors for the quad-core case and is representative of error distributions for other quad-core contention indicators. The median error across all benchmarks is 3.06 with most benchmarks reporting median errors less than 10 percent. Although the variance (*i.e.*, spread) of the error distributions for quad-core L3 cache predictions, these errors do not propagate into our final multiprocessor performance errors. Overall, these models are

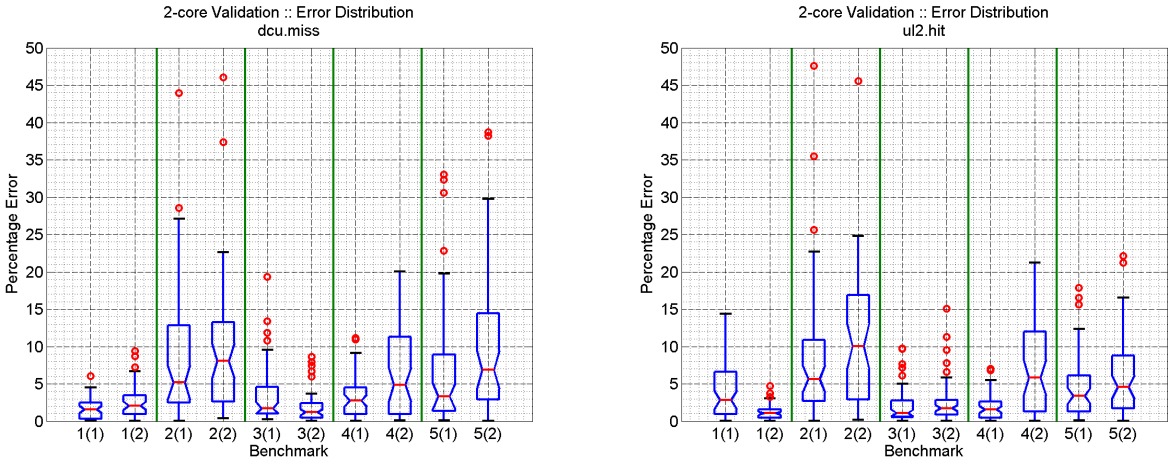

Figure 8. Error distributions for contention models predicting dual-core d-L1 cache misses (L), L2 cache hits (R).

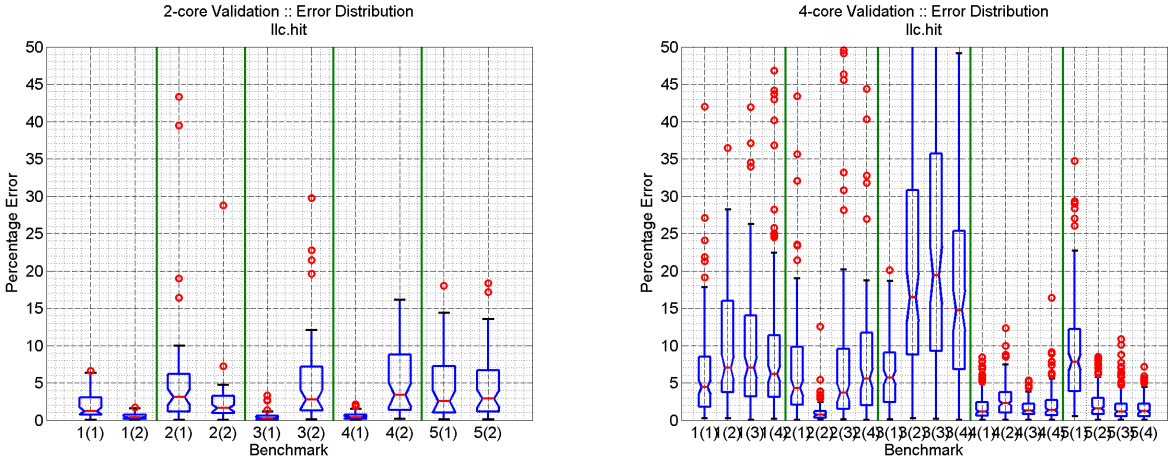

Figure 9. Error distributions for contention models predicting dual-core L3 cache hits (L), quad-core L3 cache hits (R).

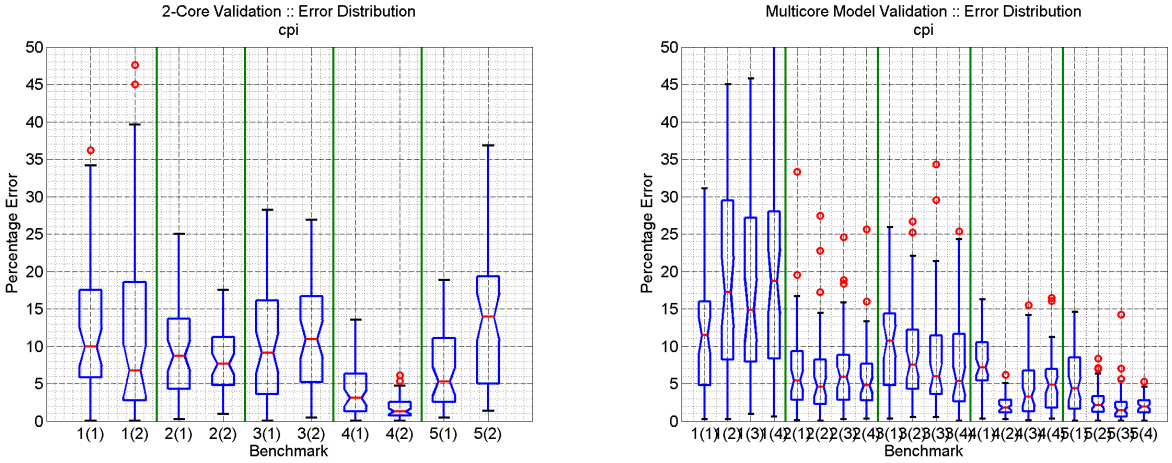

Figure 10. Error distributions for dual-core (L) and quad-core (R). Models predict per core CPI.

sufficiently accurate for early stage design optimization.

The third benchmark set (*excel-homeworld-audio-unreal*) exhibits particularly high error rates for quad-core L3 hits with median errors of 15.29 percent across the four benchmarks. However, we did not find any systematic errors to account for the difference. We might hypothesize the third benchmark set more significantly exercises the shared L3 such that the model becomes less accurate as contention increases. However, we did not find any evidence to support this hypothesis. The second set has more accurate models despite accessing the shared L3 more frequently and exhibiting greater contention. Similarly, the first and fourth sets exhibit L2 miss rates comparable to the third set. However, both the first and fourth sets have more accurate contention models despite greater L2 miss rates that lead to shared L3 accesses. Overall, the third set is more an exception than the general case and we do not find any evidence to suggest its greater model error is due to greater L3 contention.

Figure 10 illustrates the error distributions for multiprocessor delays $h_i\Big(f_i(\vec{x}),\ g_i(\vec{x}_s \mid B) \mid B\Big)$ for dual- and quad-core benchmark sets. The dual-core model accurately predicts the performance of each core with median errors of 6.63 percent across all benchmarks. Median errors for specific benchmark sets range from 1.6 percent (fourth set: *homeworld-tachyon*) to 11.1 percent (third set: *specjapp-specweb*). Similarly, the quad-core model predicts the performance of each core with median errors of 4.83 percent across all benchmarks. Median errors for specific benchmark sets range from 1.31 percent (fifth set: *painter-mentalray-outlook-encrypt*) to 18.62 percent (first set: *dense-excel-flash-md2*).

Note that larger quad-core contention errors for the third benchmark set in Figure 9R do not necessarily translate into larger quad-core CPI errors in Figure 10R. The penalty model that produces the final CPI prediction is likely robust to errors from uniprocessor or contention models since the linear regression fits multiprocessor performance to the provided predictions regardless of errors. If fit well, the penalty model might mask the intermediate uniprocessor or contention errors. However, a fit becomes less likely as such errors increase.

Figure 10R reveals larger errors for the first quad-core benchmark set (*dense-excel-flash-md2*). This set is notable for its extremely low cache activity; other benchmark sets access all levels of hierarchy more frequently. Specifically, on average, the other four benchmark have 2.1x, 1.5x, and 1.7x more d-L1 misses, L2 hits, and L3 hits, respectively. Due to such low activity, the penalty model is fit poorly for the first benchmark set. Significance testing on the three contention indicators suggest d-L1 misses, L2 hits, and L3 hits do not contribute much to model accuracy for the first quad-core benchmark set, probably because these indicator values are small in absolute terms due to low cache activity.

4.2. Scalability Analysis

CPR accurately estimates the performance of each core in a symmetric multiprocessor by leveraging inexpensively constructed uniprocessor models. We make the case for the efficiency of composable multiprocessor efficiency from an analytical study of regression training costs expressed in terms of simulation time, showing CPR incurs low marginal training costs as core count increases.

As in Section 3, let N_{uni}, N_{con}, and N_{pen} be the number of design samples required to train the uniprocessor, contention, and penalty models, respectively. Let T_n be the time required for a single detailed n-core simulation. Similarly, let M_n be the time required to train a n-core model. Equations (2)–(3) express model construction time in terms of simulation costs. The best case scenario is captured by a lower bound $M_n^{(lo)}$, in which we assume all uniprocessor models have already been created and the marginal cost of multiprocessor modeling is the time to construct the contention and penalty model. The worst case scenario is captured by an upper bound $M_n^{(up)}$, in which we must also construct each of the n uniprocessor models. The best case scenario is more likely since architects are unlikely to study multiprocessors without first understanding uniprocessors with uniprocessor regression models for benchmarks of interest.

$$M_n^{(lo)} = (N_{con} + N_{pen}) \cdot T_n \quad (2)$$

$$M_n^{(up)} = N_{uni} \cdot n \cdot T_1 + (N_{con} + N_{pen}) \cdot T_n \quad (3)$$

$$M_n^{(nv)} = N_{naive} \cdot T_n \quad (4)$$

For comparison, we consider naive regression modeling that implements, without modification, prior uniprocessor approaches for multiprocessor analysis [14]. For n-core multiprocessor, this naive approach samples N_{naive} points from the multiprocessor design space, evaluates each sample with n-core simulation, and fits a spline-based regression model to this training data. The cost of this approach is expressed as $M_n^{(nv)}$ in Equation (4).

$T_n = n^\gamma \cdot T_1$ expresses multiprocessor simulation times in terms of uniprocessor simulation times where γ is a growth factor. In the ideal case, $\gamma = 1$ and simulation times scale linearly with the number of cores. However, for our product simulators, $\gamma > 1$ due to the additional complexity of coherence, synchronization modeling that adds overhead to each cycle simulated and shared resource contention that increases the number of cycles simulated for each benchmark. Such simulation time scaling for various growth factors are representative of those observed in practice from simulation wall clock times. Superlinearly increasing multiprocessor simulation times have also been observed for other industrial and academic simulators [1].

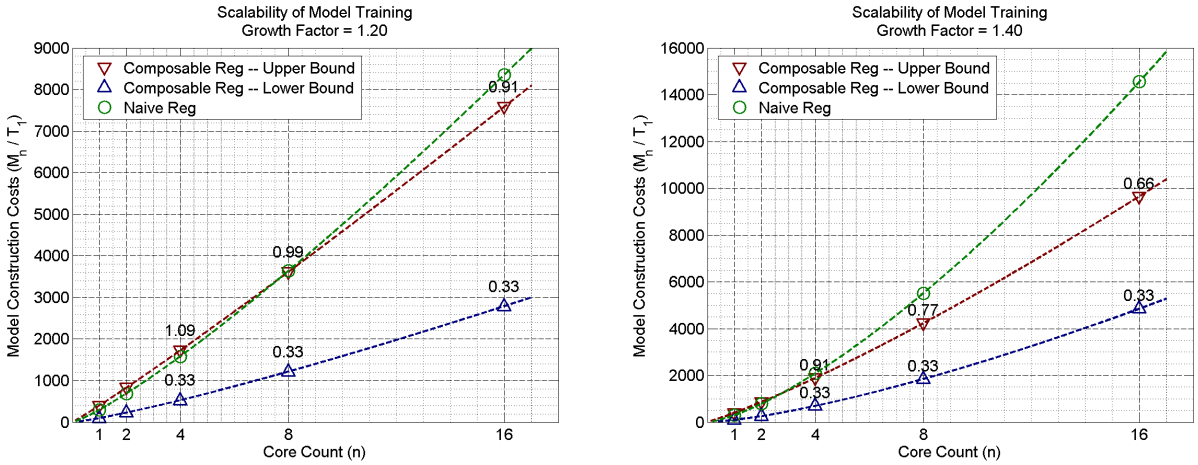

Figure 11. Modeling training costs for $\gamma = 1.2$ (L) and $\gamma = 1.4$ (R).

These growth factors are dependent on particular simulator implementations, but we take growth factors of 1.2 and 1.4 to model varying workload behavior across multiprogrammed combinations or varying degrees of multiprocessor simulation efficiency. In practice, we observe $1.2 < \gamma < 1.4$ for our product simulators. For example, $T_4 \approx 5.3T_1$ and $T_8 \approx 12.1T_1$ in a representative $\gamma = 1.2$ simulator. These costs increase significantly for the more inefficient $\gamma = 1.4$ simulator where $T_4 \approx 7.0T_1$ and $T_8 \approx 18.4T_1$.

Figure 11 illustrates the number of required simulations to train composable regression models for two growth factors: $\gamma = 1.2$, $\gamma = 1.4$. Costs are expressed in terms of the number of uniprocessor simulations M_n/T_1. Trends are illustrated for $N_{uni} = N_{naive} = 300$, $N_{con} = N_{pen} = 50$. $N_{uni} = N_{naive}$ conservatively assumes naive multiprocessor models and uniprocessor models can be fit with the same number of training samples. In practice, we expect $N_{naive} \geq N_{uni}$ since the multiprocessor design space is at least as complex as the uniprocessor design space.

$$\frac{M_n^{(lo)}}{M_n^{(nv)}} = \frac{(N_{con} + N_{pen}) \cdot n^\gamma}{N_{naive} \cdot n^\gamma} = \frac{N_{con} + N_{pen}}{N_{uni}} \quad (5)$$

$$\frac{M_n^{(up)}}{M_n^{(nv)}} = \frac{N_{uni} \cdot n + (N_{con} + N_{pen}) \cdot n^\gamma}{N_{naive} \cdot n^\gamma} \quad (6)$$

Numbers tracking the composable regression lines in Figure 11 indicate the cost advantage relative to naive regression as computed by Equations (5)–(6). Figure 11 illustrates training costs with superlinear increases in simulation time ($\gamma = 1.2$,$\gamma = 1.4$). For both values of γ, composable regression incurs 0.33x the costs of naive regression in the best case (green circles versus blue up-triangles). This best case assumes a library of uniprocessor of regression models has already been constructed for benchmarks of interest. According to Equation (5), composable regression costs will always be 0.33x those for naive regression in the best case for our particular combination of $N_{uni} = 300$, $N_{con} = 50$, and $N_{pen} = 50$.

However, in the worst case, every composable regression model first requires the construction of uniprocessor models. For $\gamma = 1.2$, these worst case costs for compoosable models are comparable to those for naive regression (green circles versus red down-triangles). However, for $\gamma = 1.4$, composable regression is much less expensive than naive regression in all scenarios with at least four cores. Composable regression incurs 0.66x to 0.91x the costs of naive regression for any model with more than two cores. Thus, even worst case costs of composable regression are lower than those for naive regression. This cost advantage will further improve for less scalable multiprocessor simulators (increasing γ) and larger multiprocessors (increasing n).

Although we present both upper and lower bounds, simulation costs for regression modeling will asymptotically approach the lower bound over time. Even if designers start in the worst case scenario where no uniprocessor models exist, they will accumulate these models as they perform design space studies. Over time, these accumulated models will likely span the space of uniprocessor workloads, leaving only the marginal costs of constructing contention and penalty models, the scenario captured by our lower bound.

5. Related Work

Li, *et al.*, decouple core and cache simulations in chip multiprocessors [15]. The core simulator generates single-core traces of L2 cache accesses that are annotated with timestamps. These traces feed a cache simulator to model interactions within shared caches. In contrast, we rely on

regression models to estimate core and cache effects. Furthermore, we pass between models a few key contention metrics instead of detailed traces.

Ipek, *et al.*, and Joseph, *et al.*, separately predict the performance of design spaces with automated artificial neural networks (ANN) trained by gradient descent and predicted by nested weighted sums [3], [4]. Joseph, *et al.*, did not consider the multiprocessor design space. Ipek, *et al.*, train neural networks for the multiprocessor design space by sampling and simulating multiprocessor designs. In contrast, we construct composable models to reduce training simulation costs by relying on a combination of training data from uniprocessor, shared resource, and multiprocessor simulations. Instead of neural networks, we use spline-based regression models from prior work by Lee and Brooks [6], [14]. Relative to neural networks, regression may require greater statistical analysis during construction, but may be more computationally efficient, numerically solving and evaluating linear systems for training and prediction, respectively.

Dubach, *et al.*, reduce the marginal cost of building neural networks by expressing a new benchmark's performance as a linear combination of performance from already modeled benchmarks [2]. The linear model, relying on performance trends captured by previously constructed networks, is less expensive to construct than a full-fledged neural network trained from scratch. Dubach, *et al.*, implement this approach for serial workloads and do not consider multiprocessors. Their proposed approach is orthogonal to ours and combining the two approaches is an avenue for future work.

Khan, *et al.*, implement models for thread-parallel SPLASH workloads and speculatively parallelized SPEC workloads [5] for a four-core symmetric multiprocessor. Although the authors attempt to control training costs with techniques similar to those by Dubach, *et al.*, they rely solely on multiprocessor simulations to train neural networks. In contrast, we implement composable models, reducing the reliance on expensive multiprocessor simulations while leveraging inexpensive uniprocessor and shared resource simulations. Furthermore, we extend the state-of-the-art in predictive modeling to an industrial simulator infrastructure with a broad range of workloads. In contrast to prior work, we consider multiprogrammed workloads and propose a framework to handle combinatorial workload complexity, which is not present in thread-level parallel workloads.

6. Conclusions

Recent advances in applying statistical regression to microprocessor studies have enabled fundamentally new capabilities in uniprocessor design space exploration. To extend these achievements to the multiprocessor domain, we propose composable regression models that combine uniprocessor, contention, and penalty models to accurately predict multiprocessor performance running multiprogrammed workloads with median errors of 4.83 and 6.63 percent for dual- and quad-core predictions. Furthermore, composable regression is scalable, requiring 0.33x the simulations of naive regresion for model construction. Collectively, this work establishes a rigorous foundation for large-scale multiprocessor analysis.

Acknowledgment

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

References

[1] J. Donald and M. Martonosi, "An efficient, practical parallelization methodology for multicore architecture simulation," *IEEE Computer Architecture Letters*, August 2006.

[2] C. Dubach, T. Jones, and M. O'Boyle, "Microarchitectural design space exploration using an architecture-centric approach," in *MICRO: International Symposium on Microarchitecture*, December 2007.

[3] E.Ipek, S.A.McKee, B. 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.

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

[5] S. Khan, P. Xekalakis, J. Cavazos, and M. Cintra, "Using predictive modeling for cross-program design space exploration in multicore systems," in *PACT: International Conference on Parallel Architectures and Compilation Techniques*, Sept 2007.

[6] B. Lee and D. 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.

[7] S. Duvall, "Statistical circuit modeling and optimization," in *5th International Workshop Statistical Metrology*, June 2000.

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

[9] *PCMark04*, Futuremark Corporation.

[10] *SPECweb99*, Standard Performance Evaluation Corporation.

[11] *TPC-C v5*, Transaction Processing Performance Council.

[12] *JAppServer2004*, Standard Performance Evaluation Corporation.

[13] S. Shenoy and A.Daniel, "Intel architecture and silicon cadence: The catalyst for industry innovation," *Technology at Intel Magazine*, October 2006.

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

[15] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron, "CMP design space exploration subject to physical constraints," in *HPCA: International Symposium on High-Performance Computer Architecture*, February 2006.