A Predictive Performance Model for Superscalar Processors

P. J. Joseph *
Freescale Semiconductor
PJ.Joseph@freescale.com

Kapil Vaswani
Matthew J. Thazhuthaveetil
Indian Institute of Science
{kapil,mjt}@csa.iisc.ernet.in

Abstract

Designing and optimizing high performance microprocessors is an increasingly difficult task due to the size and complexity of the processor design space, high cost of detailed simulation and several constraints that a processor design must satisfy. In this paper, we propose the use of empirical non-linear modeling techniques to assist processor architects in making design decisions and resolving complex trade-offs. We propose a procedure for building accurate non-linear models that consists of the following steps: (i) selection of a small set of representative design points spread across processor design space using latin hypercube sampling, (ii) obtaining performance measures at the selected design points using detailed simulation, (iii) building non-linear models for performance using the function approximation capabilities of radial basis function networks, and (iv) validating the models using an independently and randomly generated set of design points. We evaluate our model building procedure by constructing non-linear performance models for programs from the SPEC CPU2000 benchmark suite with a microarchitectural design space that consists of 9 key parameters. Our results show that the models, built using a relatively small number of simulations, achieve high prediction accuracy (only 2.8% error in CPI estimates on average) across a large processor design space. Our models can potentially replace detailed simulation for common tasks such as the analysis of key microarchitectural trends or searches for optimal processor design points.

1. Introduction

Processor architects are constantly confronted with the challenge of designing high performance microprocessors while simultaneously meeting constraints such as power consumption and design costs. The problem of identifying *optimal* processor configurations is further exacerbated for the following reasons. First, processors are evolving into increasingly complex systems with a large number and type of components such as caches, predictors and queues. As a consequence, architects must deal with a large microarchitectural design space consisting of several interacting parameters. Furthermore, modern day workloads are composed of a large spectrum of programs with widely differing characteristics.

Architects have addressed these problems in the past by exploring the design space using detailed microarchitectural simulations on representative workloads. However, this approach has high simulation costs due to the low speed of cycle-accurate simulators and large sizes of workloads. As a result, architects are forced to restrict the number of design points that are evaluated. The selection of design points considered for detailed simulation is usually governed by design constraints such as power budgets or based on the architect's experience about the relative importance of certain parts of the design space. This approach to experimentation and optimization has the following drawbacks:

- The underlying assumptions based on expert knowledge can be in error, due to the changing significance of micro-architectural parameters across process technologies, and the changing workloads.
- Lack of statistical rigor, and hence conclusions drawn can be incorrect.
- Lack of insights on issues such as the nature of performance bottlenecks, significance of individual parameters and their interactions, and characteristics of the workload.

Models for processor performance. Much like in other disciplines of science and engineering, the use of analytical models can potentially address these limitations. An analytical model that accurately characterizes the relationship between processor performance and various microarchitectural parameters would, in theory, obviate the need for detailed, expensive simulations. However, existing analytical modeling techniques for processor performance [11, 16] are based on several simplifying assumptions and only model a

*This work was done when the first author was at Indian Institute of Science.

small number of microarchitectural parameters. As a result, these models lack the accuracy or the flexibility for use in a real-world processor design cycle.

As an alternative to analytical models, several *empirical* modeling techniques for processor performance have been proposed and evaluated. For instance, Joseph et al. [10] model performance as a *linear* combination of individual microarchitectural parameters and their interactions. The key characteristic of their approach is that the linear models are learnt from data obtained from a small number of simulations performed at carefully selected points in the design space. While linear models were shown to be extensible and capable to providing accurate estimates of the *significance* of parameters and their interactions, they suffer from one significant drawback. The inherent nature of linear models prevents accurate representation of non-linear response behavior observed for processor performance. For instance, it has been empirically shown that the number of instructions issued per cycle varies nonlinearly with instruction window size[11]. We further illustrate this limitation of linear models using a simple experiment. We measured the change in superscalar processor performance as modeled by Simplescalar [2] by varying two parameters, the L1 instruction cache size and the L2 cache latency, while keeping other microarchitectural parameters fixed. Figure 1 illustrates the variation in performance for *vortex*, a benchmark from the SPEC CPU2000 suite. As expected, higher L2 cache latencies have a larger influence on performance when the L1 instruction cache is relatively small, indicated in Figure 1 by the curvature in response and sharp changes for lower cache sizes. As the number of parameters in the design space increases, such changes in the response surface can be hard to capture using linear models alone, and may result in low prediction accuracies.

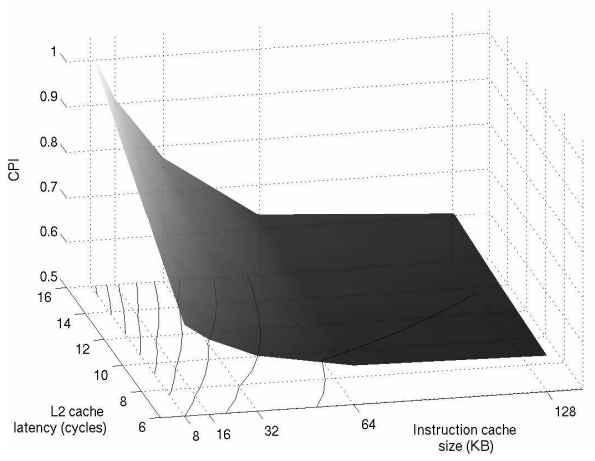

Figure 1. The CPI response surface.

Main contributions. In this paper, we propose the use of *nonlinear regression modeling techniques* to build accurate predictive models for processor performance. Specifically, we choose to build models using Radial Basis Function (RBF) networks [3], primarily because of their ability to approximate many complex functions, and the relative ease with which these models can be generated. We propose a procedure, BuildRBFmodel, that can be used to construct accurate RBF network models for a given program-input pair at low simulation cost. The procedure involves following steps.

1. The design space of interest is specified by determining the microarchitectural parameters that should be included in the model.
2. An initial a set of design points within the design space (a *sample*) is selected for simulation. We use a space-filling criteria called the L^2-star discrepancy [8] for selecting the sample.
3. Processor response for the sample is obtained using detailed, cycle accurate simulation.
4. The set of design points together with the response (*sample data*) is used to build an RBF network model.
5. An estimate of the model's accuracy is obtained by using a predetermined set of randomly generated points in the design space.
6. This process is repeated with increasing sample sizes until a desired level of accuracy is obtained.

We evaluated our models using programs from the SPEC CPU2000 benchmark suite. We built a separate model for each program-input pair and used the model to predict performance at unexplored points in the design space. Our results show that nonlinear models can achieve good prediction accuracy (2.8% error in CPI on average) across the design space with reasonable simulation cost. Furthermore, we find that the nonlinear models can accurately capture important microarchitectural trends and variations in performance. Therefore, the nonlinear models we build could potentially replace detailed simulation in several stages of the processor design cycle.

Paper outline. This paper is organized as follows. Section 2 describes the approach we use for building nonlinear models. We discuss issues such as specification of the design space, criteria for the selection of design points for simulation and describe how RBF network models can be used for approximating arbitrary functions. Section 3 describes the experimental framework we use to evaluate our modeling procedure. We present the results of our evaluation in Section 4. Section 5 discusses related work in the area of processor performance modeling. We conclude in Section 6.

2. Building Nonlinear Regression Models

2.1. Specifying the Design Space

The first step in the construction of empirical models involves the specification of the microarchitectural design space of interest i.e. selection of parameters to be included in the model, and the ranges of these parameters. The modeling process does not constrain the number of parameters that can be included in the model. The only cost associated with including a larger number of parameters is the increased cost of simulation[1]. For building models of processor performance, we derive from our previous study that estimate the significance of microarchitectural parameters [10] and include nine parameters that have the largest impact on processor performance. These parameters are listed in Table 1. We chose ranges for these parameters to be wider than the range of most current superscalar implementations. Hence, all significant and feasible design points are included within this design space.

2.2. Selection of Design Points

The careful selection of processor design points used for performance simulation is critical for building accurate predictive models. Firstly, this *sample* must be representative of the entire design space i.e. it should include points distributed in all regions of the design space since the presence of large non-sampled regions can lead to low model accuracy. Furthermore, the size of the sample should be small enough to keep the simulation cost low. Thus, a good sample selection strategy should space out points throughout the design space just close enough to capture variations in the response.

We achieve good sampling of the design space by using a variant of latin hypercube sampling [15]. In this scheme, the sample is ensured to have points corresponding to all settings of a parameter, and the settings of each of the parameters are randomly combined. For a typical set of processor micro-architectural parameters as in Table 1, a latin hypercube sample will have points for all levels of pipeline depth, all reorder buffer sizes, all L2 cache sizes, and so on. This strategy has been shown to have better coverage as compared to a simple random selection of points in the design space [6].

To further improve the quality of latin hypercube samples, we use space-filling measures to quantify the extent to which a sample covers the design space. The specific space filling metric we use, referred to as the $L2$-star discrepancy, has been analytically derived in Hickernell [8], and it measures the deviation of the sample from a uniform distribution of points in the space. Lower values of L^2-star discrepancy denote better space filling. For building our models, we generate a large number of latin hypercube samples and choose the one with the best $L2$-star discrepancy metric.

Choosing a correct sample size for experimentation is another important decision. The sample must be sufficiently large to cover the design space, and should be as small as possible to keep the simulation cost low. We observe that a suitable size can be determined using the L^2-star discrepancy metric for different sample sizes. This is demonstrated in Figure 2, for the samples we used in our experimentation. There is a knee in this curve where the metric starts to taper, indicating that further increases in size has lower impact on space coverage. We observe that a choice of sample size near this region meets our requirements.

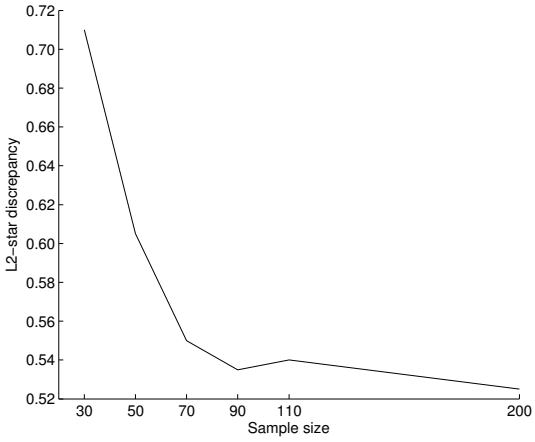

Figure 2. The best obtained L^2-star discrepancy with number of simulations for our experimental setup.

2.3 Interpolation using Radial Basis Function Networks

After we have decided on the sample of design points, we simulate the performance at each of the selected design points. We then use Radial Basis Function (RBF) networks [3] to model and interpolate the simulation output data to unexplored design points. Figure 3 illustrates the RBF network, and how we use it in our context. Structurally, the network consists three layers, an input layer which reads n inputs, a hidden layer that consists of m *radial basis functions* and an output layer consisting of a linear additive function which produces the response. Each RBF receives the entire input vector and generates a response. The output layer

[1]Our previous studies [10] have shown that the minimum number of simulation runs required for building accurate models is roughly 10 times the number of parameters.

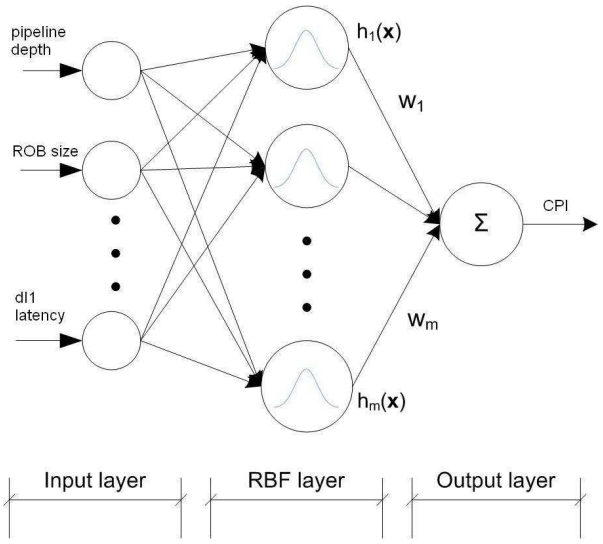

Figure 3. A Radial Basis Function network.

function accumulates responses from all RBF and generates a final response. The relationship between the input and the output of this whole network can be mathematically represented by

$$f(\mathbf{x}) = \sum_{j=1}^{m} \mathbf{w}_j \mathbf{h}_j(\mathbf{x}) \quad (1)$$

where h_j are the non-linear radial basis functions, $\mathbf{x}$ is the vector of input settings, w_j are the weights associated with the RBFs, and m is the number of RBFs chosen.

Each radial basis function $\mathbf{h_j}(\mathbf{x})$ is typically characterized by a center point $\mathbf{c_j}$ within the parameter space and a radius vector $\mathbf{r_j}$. Specifically, we use functions of the type

$$h(\mathbf{x}) = exp(-\sum_{k=1}^{n} \frac{(x_k - c_k)^2}{{r_k}^2}) \quad (2)$$

where the center point $\mathbf{c} = [c_1 c_2 .... c_n]$ and radius vector $\mathbf{r} = [r_1 r_2 .... r_n]$. Note that this function has the highest response when input corresponds to the center $\mathbf{c}$, and the response decreases as the distance of the input vector from the center increases, controlled by the radius vector $[r_1 r_2 .... r_n]$. Hence, for any input the functions which have centers close to the input have the highest response, and the overall response is effectively the weighted mean of these significant RBF responses.

For this study, we map the microarchitectural design space to the input space of the RBF network and use CPI as the response. We use the data generated from simulation to determine (i) specific design points as RBF centers (ii) radii of influence of these centers (iii) weights so that the whole network provides a good model of the CPI response metric for the entire parameter space. For this we use a scheme based on regression trees which was originally devised by Orr et al. [17]. In our context, this scheme identifies contiguous regions of the design space - specified by ranges of the design parameters - that have similar performance as measured by CPI. RBF centers are chosen at the centers of such regions, and the radius of the RBF is chosen in proportion to the size of the region. We next describe regression trees and how they identify regions within the design space.

2.4. Generating Regression Trees

As the first step in model building, we build regression trees for the sample data. Regression trees recursively partition the input parameter space of the sample into two partitions such that the variability of data in the resulting partitions is minimum. Any partitioning is governed by two parameters, an input parameter k and a value b for that input parameter. Design points with parameter values less than or equal to b form one branch of the binary tree, and the data with parameter values greater than b form the other branch. The parameter k and the boundary b are chosen to minimize the residual error between the approximation and the data, as we explain below. Suppose that a partition separates data $S = 1, 2, .., p$ at value b into left and right subsets S_L and S_R, of sizes p_L and p_R, such that

$$S_L = \{i : x_{ik} \leq b\} \quad (3)$$
$$S_R = \{i : x_{ik} > b\} \quad (4)$$

where x_{ik} is the setting of input parameter k at data point i. The mean output value on either side of the bifurcation is given by

$$\bar{y}_L = \frac{1}{p_L} \sum_{i \in S_L} y_i \quad (5)$$
$$\bar{y}_R = \frac{1}{p_R} \sum_{i \in S_R} y_i \quad (6)$$

The residual square error between the mean values and the data is given by

$$E(k,b) = \frac{1}{p}(\sum_{i \in S_L} (y_i - \bar{y}_L)^2 + \sum_{i \in S_R} (y_i - \bar{y}_R)^2) \quad (7)$$

The bifurcation with lowest E(k,b) over all possible choices of k and b is found by a discrete search over the n input dimensions and p sample points. This essentially partitions the data along an input parameter and a parameter value which minimizes variation in the resulting partitions.

The regression tree is constructed by recursively partitioning the data. Each partition creates two terminal nodes in the tree, and these nodes are recursively bifurcated. The

parameters which cause the most output variation tend to be split earliest and most often. The process of recursive splitting is continued until there are no more than p_{min} data points in all terminal nodes, where p_{min} is a method parameter whose best value is determined by experimentation. The result of the construction is a tree where each node represents a region of the design space. The root node represents the whole space, and nodes at higher depths represent increasingly smaller sub-regions.

2.5. Generating RBF network from Regression Trees

Constructing an RBF network from the regression tree involves choosing RBF centers, and the radius of the RBFs from the structure of the regression tree. The regression tree contains a root node, some non-terminal nodes, and terminal nodes. Each node has an associated hyper-rectangle in the input parameter space represented by a center $\mathbf{c}$ and a size $\mathbf{s} = [s_1 s_2 .... s_n]$, where each s_i represents the length of the corresponding parameter range associated with the hyper-rectangle. We associate RBF centers with the centers of these hyper-rectangles, choose RBFs of the type in Eq. 2, and the RBF radii $\mathbf{r} = [r_1 r_2 .... r_n]$ of these RBFs are set to the size of the associated hyper-rectangle scaled by a parameter α

$$\mathbf{r} = \alpha \mathbf{s} \tag{8}$$

We choose a best suited α based on experimentation.

We select a subset of the regression tree node centers as RBF centers such that the resulting model generalizes well. We achieve this using model selection criteria [7], which help to select a model that fits well on the training data and also has a small number of model parameters - in this case determined by the number of RBFs. We use Akaike's Information Criteria (AIC_c) for subset selection. This criteria is defined as

$$AIC_c = p * log(\hat{\sigma}^2) + 2m + \frac{2m(m+1)}{(p-m-1)} + constant \tag{9}$$

where p is the size of the sample, m is the number of RBF centers chosen , and $\hat{\sigma}^2$ is the error variance of the RBF network on the sample. These criteria combine model accuracy and complexity, and have low values when both the error variance and the number of RBF centers are simultaneously low. We choose the subset producing lowest values of the selection criteria.

We order the selection of RBF centers for inclusion in the model using the tree based selection ordering strategy in Orr et al. [17]. This strategy first selects the center of the root node of the regression tree; in this case the center of the considered design space. Then it considers the centers of the two child nodes, and chooses between including and excluding each of the total three centers. Amongst the 8 different possibilities, it chooses the one which most decreases the model selection criterion. Once this is done, it moves deeper in the regression tree for considering additional centers in a similar fashion.

2.6. Implementation

We use a modified version of the function *rbf_rt_1* in Mark Orr's MATLAB software [18] for our model construction. The software was updated to use AIC_c in the model selection. We used it to determine optimal values of the method parameters, p_{min} and α, to build the predictive model, and to achieve the prediction at different settings. We determined optimal p_{min} and α for each benchmark by choosing the values which resulted in the lowest AIC_c.

3. Experimental Framework

The accuracy of our model construction depends on the accuracy of the simulator. Hence, we use a detailed and validated superscalar processor simulator in our experimentation. Our simulation framework models pipelined, multiple-issue, dynamically scheduled, speculative execution processors. It models all the performance critical micro-architectural events and structures in superscalar processors. The pipeline, caches, branch direction and target predictors, micro-architectural queues, functional units, DRAM device timing, queuing at the memory controller, and contention for the memory bus are all modeled. We verified the functionality of each component of the simulator individually. To further verify the simulator's accuracy across the processor design space, we validated trends in the summary statistics against another similarly configured verified simulator, *alphasim* [4] at several points in the design space.

We used the simulator to run benchmarks from the SPEC CPU2000 integer suite using the *lgred* data set in MinneSPEC [12] reduced data sets. This is done using traces generated with IBM PowerPC executables, compiled with xlc compiler applying the -O3 option. We run the benchmarks to completion, and do not use any sampling.

We used our simulator to evaluate the CPI at a few combinations of the nine key micro-architectural parameters listed in Table 1. These nine parameters are the most significant factors affecting the CPI of SPEC benchmark execution using MinneSPEC inputs [10]. We note that the relative significance of microarchitectural parameters is input dependent. For instance, the memory subsystem parameters would have a higher influence on performance if the SPEC reference inputs were used. We selected design points for simulation within the 9-dimensional space bounded by the

Parameter	Low Value	High Value
pipe_depth	22	9
ROB_size	37	115
IQ_size	0.31*ROB_size	0.69*ROB_size
LSQ_size	0.31*ROB_size	0.69*ROB_size
L2_size	256KB	8MB
L2_lat	18	7
il1_size	8KB	64KB
dl1_size	8KB	64KB
dl1_lat	4	1

Table 2. Parameter range used for generating test data.

Benchmark	mean	max	std
181.mcf	2.1	12.7	1.8
186.crafty	2.9	10.8	2.7
197.parser	2.2	8.4	2.0
253.perlbmk	4.0	17.0	3.1
255.vortex	3.4	12.0	2.7
300.twolf	3.2	11.9	2.3
183.equake	1.9	5.9	1.3
188.ammp	2.5	4.8	1.2
Average	2.8		

Table 3. Error diagnostics of predictive model.

parameter ranges shown in Table 1. These ranges were chosen to be wider than the range of most current superscalar implementations. A sample of design points in the space was generated by our variant of latin hypercube sampling, and the benchmarks were simulated to completion at each selected set of design points. We then built predictive models using RBF networks for CPI measures obtained from these simulations using the input transformations listed in the table. These particular transformations were chosen for simplicity since the models tend to be accurate irrespective of the transformations.

We used a randomly and independently generated set of *test* data points to empirically estimate the predictive accuracy of the resulting models. We generated fifty such design points within a more restricted parameter space specified in Table 2. All benchmarks were simulated at these test points and the resulting CPI was compared against the CPI predicted by model. We used the mean absolute percentage error in CPI, standard deviation of this error, and the maximum error as model accuracy metrics.

Sample size	30	50	70	90	110	200
p_{min}	1	2	1	1	1	1
α	5	8	10	12	6	7
Number of RBF centers	15	16	22	27	40	76

Table 4. Diagnostics of RBF model for *mcf*.

4 Results

Figure 4 presents the mean, standard deviation, and maximum error of the RBF network models for *mcf* and *twolf*, at different sample sizes. The model error decreases with increasing sample sizes for the benchmarks, and we observe similar plots for other benchmarks. Table 3 presents the error data for eight benchmarks at a sample size of 200. The mean error in prediction across all benchmarks is 2.8%, and the maximum error at any design point for any benchmark is 17%. The floating point benchmarks *equake* and *ammp* have low maximum error values of 5.9 and 4.8 respectively. These errors are smaller than those achieved by any of the existing modeling techniques, despite the fact that our models are built and tested across a significantly larger design space.

We also observe from Figure 4 that although the model error decreases with increasing sample size, this reduction tapers at higher sample sizes. For the programs we study, a sample size of around 90 combines good accuracy with low simulation cost; increasing the sample size beyond this point improves error at a lower rate. This point corresponds to the knee in the L^2-star discrepancy curve (Figure 2).

The regression tree construction method parameters, p_{min} and α also affect the accuracy of models. Table 4 presents the best p_{min} and α values identified during RBF network construction for *mcf* benchmark execution. The table also presents number of RBF centers that were chosen within the design space. The best p_{min} value is typically 1. The radius is typically 5-12 times the size of the regression tree region, implying the influence of an RBF on the predicted response in its own as well as in its neighboring regions. We also observe that the number of RBF centers is typically restricted to much less than half the number of sample points.

The regression tree splitting forms a significant part of the model construction, and contributes to its accuracy. We present in Table 5 the initial and most significant splits. For *mcf* the most significant splits are for L2 latency, L1 data cache latency, and L2 size, while for *vortex* these are L1 data cache latency, instruction cache size and issue queue size. Figure 5 gives another view of the same tree splitting, providing the distribution of the actual parameter values at which splitting occurs.

Parameter	Low Value	High Value	Number of levels	Transformation
Pipeline depth	24	7	18	linear
ROB size	24	128	S	linear
Issue Queue size	0.25*ROB_size	0.75*ROB_size	S	linear
LSQ size	0.25*ROB_size	0.75*ROB_size	S	linear
L2 size	256KB	8MB	6	log
L2 lat	20	5	16	linear
instrn. l1 size	8KB	64KB	4	log
data l1 size	8KB	64KB	4	log
data l1 lat	4	1	4	linear

Table 1. Parameter ranges and levels. S denotes sample size dependent number of parameter levels.

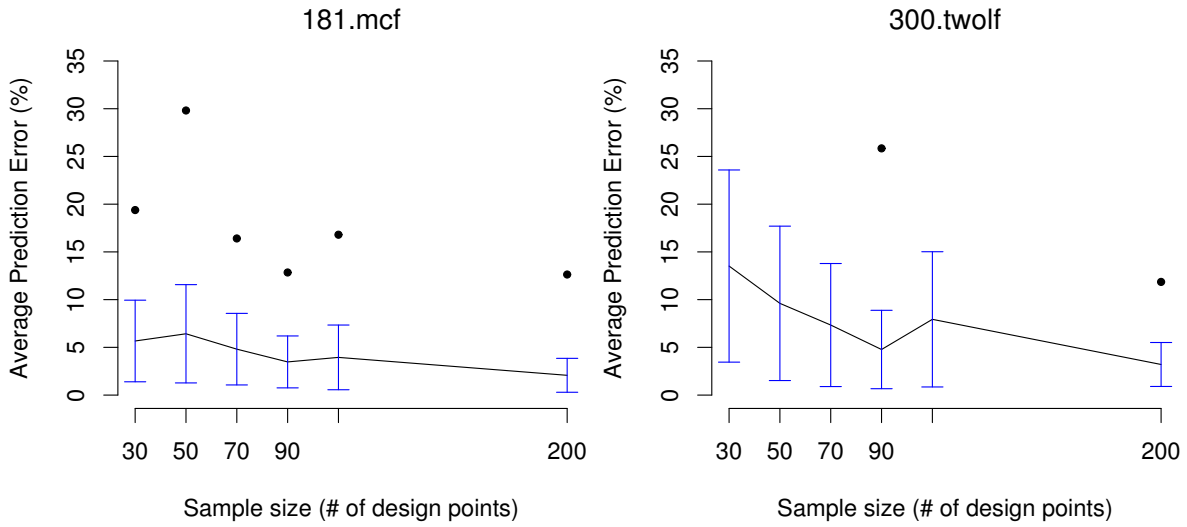

Figure 4. Mean error, standard deviation, and maximum error of our predictive model.

Number		1	2	3	4	5	6	7	8
mcf	parameter	L2_lat	dl1_lat	L2_size	L2_size	L2_size	dl1_lat	ROB_size	pipe_depth
	value	11.5	2.5	370KB	370KB	740KB	2.5	56.5	17.9
	depth	1	2	2	3	3	3	4	4
vortex	parameter	dl1_lat	il1_size	IQ_size	pipe_depth	L2_lat	IQ_size	L2_lat	ROB_size
	value	2.5	12KB	0.34*	18.5	13.5	0.36*	13.5	41.3
	depth	1	2	2	3	3	3	3	4

Table 5. The most significant splitting points during regression tree construction.

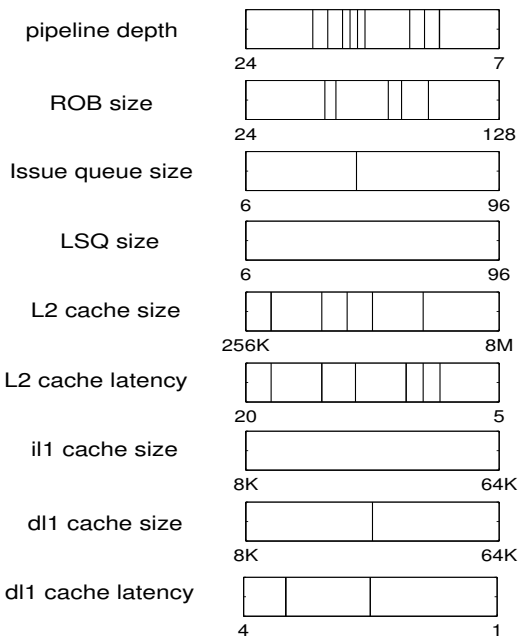

Figure 5. The parameter values in tree splitting for *mcf*.

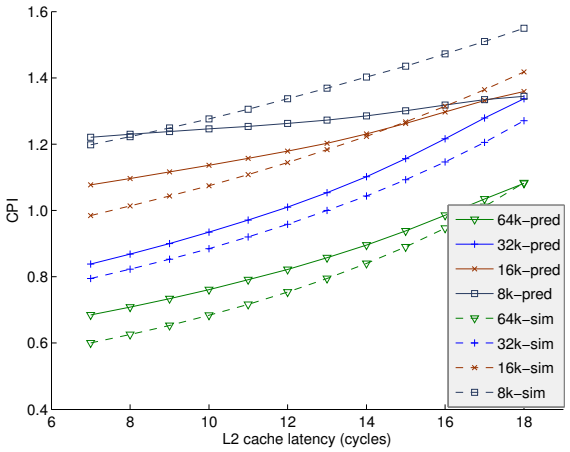

Figure 6. Using RBF network for predicting variation in performance for the *vortex* benchmark due to different icache sizes and L2 latencies

4.1 Predicting Microarchitectural Trends

Apart from high predictive accuracy, another important characteristic of a high quality model is the ability to predict major trends in response. To evaluate the RBF network models on this count, we identified several potentially significant two-factor interactions, predicted the nature of the interaction using our model and compared the prediction against data obtained from detailed simulation. Figure 6 presents the results for one such interaction between the instruction cache size and the L2 cache latency for the *vortex* benchmark. In this figure, solid lines represent the simulated CPI whereas dashed lines represent the CPI predicted by our model. We find that the predicted values closely mirror the trends obtained from simulation, except for configurations with low instruction cache sizes and high L2 latencies where the model fails to predict the large drop in performance.

4.2. Comparison with Linear Regression Models

Having established the quality of nonlinear models, we were interested in answering the following question: how do simple linear regression models [10] compare against their non-linear counterparts in terms of the accuracy of prediction? To ensure fair comparison in our experimental setup, we built linear regression models for the 9-parameter design space specified in Table 1. We used the same space-filling samples that were used for building nonlinear model. The amount of data we have restricts the linear model to those with the main effects and all two-parameter interactions only. After building a linear model, we use variable selection based on the AIC criteria to eliminate insignificant factors from the model. Finally, we estimate the prediction accuracy of linear models using the same set of test data points we used for evaluating nonlinear models.

Figure 7 plots the change in predictive accuracies of both linear and nonlinear models for different sample sizes for three benchmarks. The nonlinear models consistently have model. For *mcf*, the linear model has a higher mean CPI error of 6.5% as compared to 2.1% for the non-linear model even with a sample size of 200. We find similar differences between predictive accuracies for all benchmarks we evaluated.

5. Related Work

As a result of the high cost of using simulators, efforts have been made at developing models as alternatives to simulation for exploring the processor design space. Theoretical models [11, 16] relate performance to a few key micro-architectural parameters and program characteristics. These models typically measure the program execution speed under an ideal micro-architecture, and then account for the slowdowns due to cache misses, branch mispredictions, and

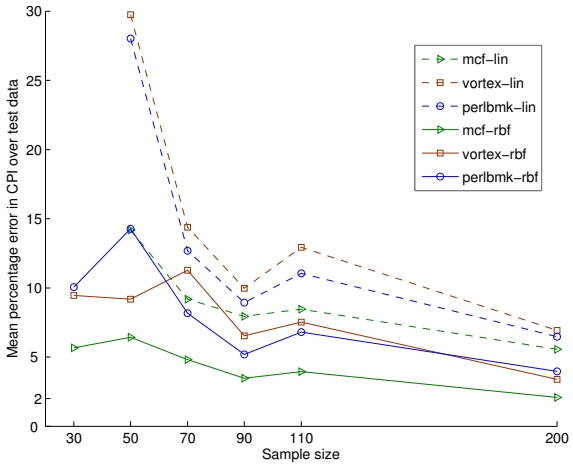

Figure 7. Comparison of the predictive accuracies of linear and RBF network models.

other significant micro-architectural events. Building the model requires the collection of these micro-architectural statistics at the considered design point. They are useful to evaluate and compare the performance of closely related designs, but they have not been demonstrated to be accurate across the entire feasible design space.

Statistical simulation [5, 19] uses profiled program statistics as well as cache and branch prediction statistics to generate a synthetic instruction trace which can guide processor simulation. These simulations converge to a steady value quickly and hence make it feasible to study a larger set of configurations. However, its accuracy has not been demonstrated across the entire design space. Its accuracy is typically tested by varying a few parameters[5].

Yi et al. [20] have quantified the significance of micro-architectural parameters by conducting simulations based on foldover Plackett-Burman experimental designs. Plackett-Burman designs are a way of choosing n parameter settings which allow the accurate estimation of the n parameter effects in a little over n simulations. However, these designs cannot quantify all the interactions between processor parameters, which we observe are significant.

In all these modeling schemes, the designer is assumed to have prior knowledge or must make simplifying assumptions about the relative significance of micro-architectural parameters. For instance, experimentation based on the Plackett-Burman design inherently assumes that all parameter interactions are negligible, and Karkhanis and Smith [11] assume a set of significant miss events. Eeckhout et al. [5] pre-determine a set of program properties and events to profile for statistical simulation.

Joseph et al. [10] propose an iterative procedure for building accurate linear regression models from simulation data, and obtained estimates of the significance of microarchitectural parameters in superscalar processors. However, we observe that the accuracy of linear models in predicting processor performance is much lower than the non-linear models we present.

Agakov et al. [1] develop statistical models relating program execution to compiler optimization sequences for several benchmarks, and use it to predict the best optimization sequence for a new program. The prediction is done using the model of the statistically closest program in terms of program properties. Marin and Mellor-Crummey [14] build models to predict application performance on processors using a combination of architecture independent program parameters and micro-architectural parameters. The focus of the work is to predict the application performance for various input sizes on a target architecture. In contract, we predict the performance of a fixed application and input combination on different target architectures. The two approaches can possibly be combined to learn a model relating application characteristics, input sizes, and micro-architectural parameters to execution speed.

In parallel with our work, Lee and Brooks [13] and Ipek et al. [9] have independently developed predictive models for processors. Lee and Brooks use regression splines to build predictive models from simulation data. Ipek et al. use artificial neural networks for the same purpose. The key differences in our approach are: (1) use of latin hypercube sampling to select design points for simulation, (2) partitioning of the design space, and (3) and use of localized radial basis functions. These techniques help us achieve high accuracy at relatively low simulation cost, .

6. Conclusion

In this paper, we have developed models capable of accurately predicting performance across the entire superscalar processor design space. We achieve this by training and optimizing radial basis function networks using simulation data from a carefully selected set of design points spread throughout the design space. We obtain an average prediction error of 2.8%, and this is significantly better than the accuracy of other modeling techniques reported in literature. The model is accurate enough to be potentially used by processor architects to systematically explore the design space for optimal design points.

In constructing predictive models, we have interpolated processor performance using RBF networks. Other modeling techniques could possibly be used; its usefulness in improving prediction accuracy requires further study. Further, the simulation costs involved in constructing predictive models can potentially be reduced using adaptive sampling, wherein sets of design points to simulate are selected based

on data from initial small samples.

While our focus in this paper has been on developing modeling for performance of speculative superscalar processors similar models can be developed for other metrics such as power consumption. Furthermore, our model building process and strategies can be applied for other types of processors and for building models of systems performance incorporating other parameters such as compiler optimizations. However, achieving accurate models in each of these settings will require further tuning of the model building strategies.

References

[1] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, and G. Fursin. Using Machine Learning to Focus Iterative Optimization. In *Proceedings of Code Generation and Optimization (CGO)*, 2006.

[2] T. Austin, E. Larson, and D. Ernst. SimpleScalar:An Infrastructure for Computer System Modeling. *IEEE Computer*, February 2002.

[3] D. S. Broomhead and D. Lowe. Multivariate Function Interpolation and Adaptive Networks. *Complex Systems*, 2:321–355, 1988.

[4] R. Desikan, D. Burger, and S. W. Keckler. Measuring Experimental Error in Microprocessor Simulation. In *Proceedings of 28th Annual International Symposium on Computer Architecture*, July 2001.

[5] L. Eeckhout, R. H. Bell Jr., B. Stougie, K. D. Bosschere, and L. K. John. Control Flow Modeling in Statistical Simulation for Accurate and Efficient Processor Design Studies. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, 2004.

[6] K. T. Fang, C. X. Ma, and P. Winker. Centered L_2-discrepancy of Random Sampling and Latin Hypercube Design, and Construction of Uniform Designs. *Mathematics of Computation*, 71(237), 2002.

[7] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2001.

[8] F. J. Hickernell. A Generalized Discrepancy and Quadrature Error Bound. *Mathematics of Computation*, 67(221), 1998.

[9] E. Ipek, S. A. McKee, B. R. de Supinski, M. Schulz, and R. Caruana. Efficiently Exploring Architectural Design Spaces via Predictive Modeling. In *Proceedings of ASPLOS - XII*, October 2006.

[10] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. Construction and Use of Linear Regression Models for Processor Performance Analysis. In *Proceedings of the 12th International Symposium on High Performance Computer Architecture (HPCA-12)*, February 2006.

[11] T. Karkhanis and J. E. Smith. A First-order Model of Superscalar Processors. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, June 2004.

[12] A. J. KleinOsowski and D. J. Lilja. MinneSPEC: A new SPEC Benchmark Workload for Simulation-Based Computer Architecture Research. *Computer Architecture Letters*, June 2002.

[13] B. Lee and D. Brooks. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *Proceedings of ASPLOS - XII*, October 2006.

[14] G. Marin and J. Mellor-Crummey. Cross-Architecture Performance Predictions for Scientific Applications Using Parameterized Models. In *Proceedings of Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 2004.

[15] M. D. McKay, R. J. Beckman, and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code. *Technometrics*, 1979.

[16] D. B. Noonburg and J. P. Shen. Theoretical Modeling of Superscalar Processor Performance. In *Proceedings of the 27th International Symposium on Microarchitecture*, December 1994.

[17] M. Orr, J. Hallam, K. Takezawa, A. Murray, S. Ninomiya, M. Oide, and T. Leonard. Combining Regression Trees and Radial Basis Function Networks. *International Journal of Neural Systems*, 10(6):453–465, 2000.

[18] M. J. L. Orr. *MATLAB Functions for Radial Basis Function Networks*. Division of Informatics, Edinburgh University, Edinburgh, Scotland, U.K, June 1999.

[19] M. Oskin, F. T. Chong, and M. Farrens. HLS: Combining Statistical and Symbolic Simulation to Guide Microprocessor Designs. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

[20] J. J. Yi, D. J. Lilja, R. Sendag, S. V. Kodakara, and D. M. Hawkins. Characterizing and Comparing Prevailing Simulation Techniques. In *Proceedings of the 11th International Symposium on High Performance Computer Architecture*, January 2005.