Machine Learning Models to Predict Performance of Computer System Design Alternatives

Berkin Ozisikyilmaz, Gokhan Memik, Alok Choudhary
Department of Electrical Engineering and Computer Science
Northwestern University, Evanston, IL 60208
{boz283, memik, choudhar}@eecs.northwestern.edu

Abstract

Computer manufacturers spend a huge amount of time, resources, and money in designing new systems and newer configurations, and their ability to reduce costs, charge competitive prices, and gain market share depends on how good these systems perform. In this work, we concentrate on both the system design and the architectural design processes for parallel computers and develop methods to expedite them. Our methodology relies on extracting the performance levels of a small fraction of the machines in the design space and using this information to develop linear regression and neural network models to predict the performance of any machine in the whole design space. In terms of architectural design, we show that by using only 1% of the design space (i.e., cycle-accurate simulations), we can predict the performance of the whole design space within 3.4% error rate. In the system design area, we utilize the previously published Standard Performance Evaluation Corporation (SPEC) benchmark numbers to predict the performance of future systems. We concentrate on multiprocessor systems and show that our models can predict the performance of future systems within 2.2% error rate on average. We believe that these tools can accelerate the design space exploration significantly and aid in reducing the corresponding research/development cost and time-to-market.

1. Introduction

Computer manufacturers spend considerable amount of time, resources, and money to design new desktop/server/laptop systems each year to gain advantage in a market that is worth hundreds of billions of dollars. When a new computer system is designed, there are many different types of components (such as CPU type, CPU frequency, motherboard, memory type, memory size, memory speed, busses, hard disk, etc.) that need to be configured. It is also hard to understand the different tradeoffs and interactions among these components. Designers cannot also use simulation or other modeling techniques, because at this high level, the existing models tend to have high inaccuracies resulting in possibly reducing the efficiency of end systems. As a result, systems designers need to rely on the existing systems' performance and their intuitions during the design of new systems. In this work, we aim to fill this important gap and provide tools to guide the systems design process.

The design of any computer component is complicated. For example, during the design of microprocessors, several parts of the processor need to be configured (e.g., cache size and configuration, number of ALUs, etc. need to be selected). Currently, the most common methodology architects use is to simulate possible configurations using cycle-accurate simulators and make decisions based on these outcomes of these simulations. During the design of a CPU, there are various parameters that need to be set. For example, in Section 4.1 we have selected 24 different parameters that can be varied for a CPU. If a designer wants to simulate 4 different values for each parameter, then there are 4^{24} combinations, i.e., the *design space* consists of 4^{24} elements. Finding the best configuration that meets the designers' constraints among these is called the *design space exploration*. Each element in the design space can take hours to days to simulate, therefore it is not possible to simulate all the configurations on a cycle-accurate simulator. This limits the number of configurations architects can consider. Currently, most designers rely on heuristics to guide them during this design process. For example, simulated annealing [1] has been used to find the set of configurations they will evaluate. However, our work differs from such approaches in two important ways. First, we use the same modeling techniques on the systems data published on the SPEC webpage to create accurate models. Second, even for the simulation data, we show that our models provide higher accuracy levels than the existing methods.

In summary, in this work we develop predictive models that will aid the developers. Specifically, we

a) develop predictive models using neural networks and linear regression to estimate the performance of a system by just using the information about its components,

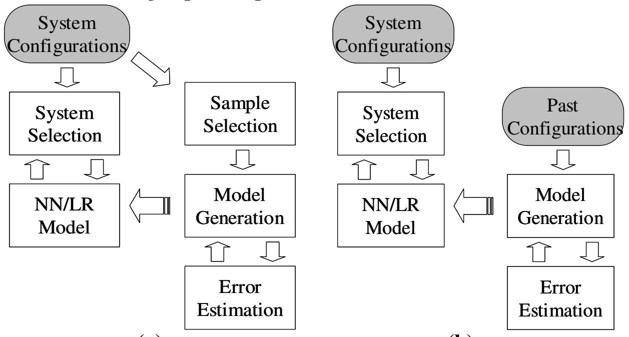
b) show that the performance of a system can be accurately predicted by using information from past systems, and

c) show that the performance of a processor can be accurately predicted by using a small fraction of the overall set of possible simulations.

The rest of this paper is organized as follows. In Section 2, we give an overview of design space exploration and how our models can be used by system manufacturers. In Section 3, we present our predictive models. Section 4 presents the results. Sections 5 and 6 present the related work and conclusions, respectively.

2. Overview of Predictive Modeling

In this work, we develop two types of models. These models correspond to how the designers can utilize the predictive models. Both approaches are depicted in Figure 1. The first one (Figure 1(a)) is called *sampled design space exploration.* It chooses a random subset of the configurations and using the performance of these configurations predicts the performance of the rest of the design space. This can be achieved by developing the selected configurations and evaluating their performance or by simulating the system on a simulator and using the performance numbers obtained from the simulator for the selected configurations. Then this data is used to generate a predictive model. As we will describe in Section 3, we have developed several models based on linear regression and neural networks. Using the error estimation provided by modeling and validation process, we select the model (neural network or linear regression) that provides the highest accuracy. This model is then used during the design space exploration to estimate the performance of the target systems. In addition to this mode of operation, we can also generate models by using the results of the previous systems in the market. This mode of operation is called *chronological predictive models*, which uses historical performance announcements to predict the performance of future systems. In this modeling task, the selection of the input data set is determined by the already published results. Let's assume without losing any generality that we are trying to estimate the performance of the systems that will be built in 2007. We can then utilize the results announced in 2006 to develop a model, i.e. use as our training data. We estimate the error of the developed models using 2006 data set and then use the best model to predict the performances of future systems as shown in Figure 1(b). We must note that there may be other means of utilizing the predictive models during the design space exploration. However, we restrict ourselves to sampled design space exploration and chronological predictions, because they exhibit the most beneficial use for the design space exploration.



(a) (b)

Figure 1. Overview of design space exploration using predictive modeling: (a) sampled design space exploration and (b) chronological predictive models.

An important aspect of our work is the use of real data. In this work, we do not only show that our models can be effectively used to model simulation data, but more importantly we show that the real system performance can be accurately predicted. Specifically, we show that our models have high accuracy on performance numbers created via simulation and then show that these methods also achieve high accuracy when data from real systems are used. We train and evaluate our models using previously announced SPEC results [4] (Section 4.3). For example, to develop our models for the chronological estimations, we utilize the SPEC announcements made in 2005 to train our models and then predict the performance of the systems announced in 2006. Hence, we can precisely report how accurately we can estimate the real system performance. We must highlight that SPEC rating is the most common means of comparing the performance of different systems and hence they are of tremendous importance to system manufacturers. SPEC ratings are commonly used for marketing and also used for setting the prices of the systems. Consequently, system manufacturers put great effort in optimizing their systems for these ratings.

As we elaborate further in Section 4, the complexity and dimensionality of the data is high: the records based on the simulations include 24 dimensions (i.e., parameters); for SPEC announcements, each record provides information on 32 parameters (representing the dimensionality of the data) in the system. Each SPEC announcement also provides the execution times of SPEC applications as well as the SPEC ratings (output measures). Similarly, the simulations also provide the execution time in terms of cycles (output measure). Despite the diversity of the data sets, our predictive models are very accurate in estimating the system performance for both sampled design space exploration and chronological estimations. Specifically, for sampled design space exploration, we obtain 3.5% error rate on average when only 1% of the design space is used for training. For chronological predictions, on the other hand, the estimation error is 2.2% on average. Considering the tremendous cost advantages of using predictive models and such highly accurate predictions, the use of machine learning tools during system design space exploration, therefore, could be significant competitive advantage.

3. Predictive Models

In this paper, we use predictive modeling techniques from machine learning [5, 6] to obtain estimates of performance of systems by using information about their components as the input. We use a total of nine models. The four linear regression models are described in the next section. Section 3.2 discusses the five neural network based models developed in this work.

3.1. Linear Regression (LR) Models

Regression analysis is a statistical technique for investigating and modeling the relationship between variables. In this model, we have n observations $y=y_1,\ldots,y_n$ called the response variables and $x_i=x_{i,1},\ldots,x_{i,p}$ for i=1..n that are predictor or regressor variables. The simplest linear regression is of the form $y=\beta_0+\beta_1 x+\varepsilon$. In this formula β represents the coefficients used in describing the response as a linear function of predictors plus a random error ε. In our input data set we have multiple predictor variables, causing the response y to be related to p regressor/predictor variables. The model then becomes $y=\beta_0+\beta_1 x+\beta_2 x+\ldots+\beta_p x+\varepsilon$, where y and x are vectors of n numbers (observations). The fitting of a regression model to the observations is done by solving the p+1 β coefficients. The method of least squares error (LSE) is used. In this model, it is assumed that the error term ε has $E(\varepsilon)=0$, $Var(\varepsilon)=\sigma^2$, and that they are uncorrelated. The least-square equation is of the form [7]

$$S(\beta_0, \beta_1, \ldots, \beta_p) = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n}\left(y_i - \beta_o - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

$S(\beta)$ may be minimized by solving a system of p+1 partial derivatives of S with respect to $\beta_{ij} \in [0,p]$. The solutions to these equations are the estimates for the coefficients β.

We used the linear regression model inside the SPSS Clementine [8] tool. In Clementine there are 4 available methods for creating the linear regression models: Enter (**LR-E**), Stepwise (**LR-S**), Forwards (**LR-F**), and Backwards (**LR-B**). In our experiments, for sampled design space we have seen that the

Backwards (**LR-B**) method produced the best results. Therefore, we only present results for LR-B. The Backwards method builds the equation in steps. The initial model contains all of the input fields as predictors, and fields can only removed from the model. Input fields that contribute little to the model are removed until no more fields can be removed without significantly degrading the model. Generally, we found that the linear regression models can be built quickly for our system. It took on the order of milliseconds to generate the models from our input data set.

3.2. Neural Network Models

Neural networks, or more accurately, Artificial Neural Networks (ANN), have been motivated by the recognition that the human brain processes information in a way that is fundamentally different from the typical digital computer [6]. A neural network is basically a simplified model of the way the human brain processes information. It works by simulating a large number of interconnected simple processing units that resemble abstract versions of neurons. The feedforward ANN are multivariate statistical models used to relate p predictor variables $x_1,\ldots,x_p$ to q response variables $y_1,\ldots,y_q$. The model has several layers, each consisting of either the original or some constructed variables. The most common structure contains three layers: the inputs which are the original predictors, the hidden layer comprised of a set of constructed variables, and the output layer made up of the responses. For a hidden unit the activation function can be linear, hard limit, sigmoid, or tan-sigmoid function. The model is very flexible containing many parameters and it is this feature that gives a neural network a nearly universal approximation property. The usual approach to estimate the parameters is by minimizing the overall residual sum of squares taken over all responses and all observations. This is a nonlinear least-squares problem. Often backpropagation procedure, variation of steepest descent, is used.

We used the SPSS Clementine tool to build the ANN models. The neural network node provides five different training methods: Quick (**NN-Q**), Dynamic (**NN-D**), Multiple (**NN-M**), Prune (**NN-P**), and Exhaustive Prune (**NN-E**). In our models, we use two methods. The first one is the Single layer (**NN-S**) method (a modified version of NN-Q) and has a constant learning rate. This method uses only one hidden layer, which is smaller than others. As a result, the models are faster to train. This model is similar to the one developed by Ipek et al. [2]. Note that Ipek et al. use this model to for estimating the results of processor simulations. The other method that we use is Exhaustive prune (NN-E) method. In this method, network training parameters are chosen to ensure a very thorough search of the space of possible models to find the best one. It is the slowest of all, but often yields the best results. During our analysis of the models, we have observed that the time it takes to build the neural network models vary significantly. While the NN-S model takes on the order of seconds to build, the NN-E models can take up to tens of minutes for the largest input data sets. However, relative to the time and cost of building a real system, these development times are still negligible.

3.3. Error Estimation using cross-validation

Clementine software does not provide the estimated predictive error for the model it creates. In model creation, Clementine randomly divides the training data into two equal sets, using half of the data to train the model and the other half to simulate. To get an accurate estimate for the estimated predictive error, we have generated five random sets of 50% of the training data, and calculated the error the model achieves on these data subsets using cross-validation. We have taken the average predictive error on these data sets, as well as the maximum of the error. Both of the error estimates are very close, and in general maximum gives a closer estimate. Therefore, in the following sections we will only present the estimates using the maximum error. Note that the *true error rates* of the models are calculated by using the created models on the whole (100% of the) data.

3.4. Data Preparation and Input Parameters

Data preparation is an important part of the predictive modeling. In our experiments, Clementine software automatically scales the input data to the range 0-1 to prevent the effect of scales of different parameters. The linear regression methods expect the input parameters to be numerical. Therefore some of the inputs to Clementine (as they will be presented in the following section) need to be mapped to numeric values. For some other input parameters this kind of transformation is not possible, hence these are omitted by Clementine. However, neural network models can have any type of input (numeric, flag, categorical), and are automatically transformed and scaled to be used in model generation. In this work, we feed all the input available parameters to Clementine. Then the program automatically measures the importance of the parameters, and depending on the methodology adds or removes predictor variables to the model. In some of the chronological design space experiments Clementine omits some predictor variables because these input parameters does not have any variation (e.g. single L2 cache size configuration). Other than this kind of predictor elimination, we don't discard any input.

4. Prediction Results

In our analysis we have used the SPEC benchmark and the corresponding published results. There are several possible methods of presenting SPEC performance numbers. For the first setup, sampled design space exploration, we have used the number of cycles the simulated processor consumes to run the SPEC applications. For the chronological design space exploration, we have used the published SPEC numbers. The most common one, SPECint2000 rate (and SPECfp2000 rate), is the geometric mean of twelve (fourteen) normalized ratios. Specifically, SPEC CPU 2000 contains 12 integer applications, 14 floating-point applications, and base runtimes for each of these applications. A manufacturer runs a timed test on the system, and the time of the test system is compared to the reference time, by which a ratio is computed. The geometric mean of these ratios provides the SPEC ratings. These ratings are important for companies because it is the means how companies compare their products with others in the market, and form their marketing strategies. Hence, in Section 4.3, we present the accuracy of our techniques for this rate. In addition, we have also tested individual SPEC applications and show that they can also be accurately estimated, however due to space constraints their presentations are omitted in this paper.

In Section 4.1, we present the simulation framework that we have used and provide the details of the data that we used in the real system framework. In Section 4.2, we present predictions for sampled design space when data from the simulations are used. Then, in Section 4.3, we present the predictive models for chronological estimations, i.e., the results when the training data set contains records from year 2005 and the predicted data set contains records from year 2006, for single processor and multiprocessor systems, respectively.

4.1. Framework

For sampled design space models, we use SimpleScalar [12] tool set, which is a system software infrastructure used to build modeling applications for program performance analysis, detailed microarchitectural modeling, and hardware-software co-verification. We do not use any particular feature of the simulator in our models; hence our approach may be applied to other simulator frameworks. For our analysis, based on the work by Phansalkar et al. [14] we have selected 12 applications from the SPEC2000 benchmark. The results for the following applications are presented due to space constraints: applu (fp), equake (fp), gcc (int), mesa (fp), and mcf (int). The remaining results are similar. In our simulation framework, we use a partial simulation technique to reduce time for simulation per each application, while incurring a slight loss of accuracy, because the SPEC applications runs billions of instructions, and simulators are usually slow. Since we want to get simulations for all the configurations in our design space, it is impossible to run the applications to completion. We have used SimPoint [13], which uses Basic Block Distribution Analysis as an automated approach for finding the small portions of the program to simulate that are representative of the entire program's execution. This approach is based upon using profiles of a program's code structure (basic blocks) to uniquely identify different phases of execution in the program. We use the simulation points given by SimPoint and execute 100 Million instructions for each interval.

Table 1 shows the parameters used for the microprocessor study, which corresponds to 4608 different configurations per benchmark. Note that this corresponds to performing 4608 simulations for each target benchmark. We have calculated the range of the simulated execution cycles (i.e., the ratio of the fastest to slowest configuration for each benchmark) and the variance: Applu/1.62/0.16, Equake/1.73/0.19, Gcc/5.27/0.33, Mesa/2.22/0.19, and Mcf/6.38/0.71. We can see that the range of the results can be very wide for some applications (e.g., mcf has a range of 6.38). Despite this range, our models can predict the simulation outcome very accurately by using a small fraction of the simulation data (c.f., Section 4.2).

In addition to predicting the simulation outcome (Sampled Design Space described in Section 4.2), we also perform modeling of real system performance (Chronological Estimations described in Section 4.3). For these models, we use the SPEC announcements. SPEC contains results announced since 1999. Hence, we have to first prune the data before developing our models. Specifically, SPEC results contain announcements from Intel, Alpha, SGI, AMD, IBM Power PC, Sun Ultra SPARC, etc. based systems. Among these, we have chosen to analyze the systems based on AMD Opteron (Opteron), Intel Pentium D (Pentium D), Intel Pentium 4 (Pentium 4), and Intel Xeon (Xeon). We also model multiprocessor systems based on AMD Opteron: 2, 4, and 8 way Shared Memory Multiprocessors (SMPs) are modeled, corresponding to AMD Opteron 2, 4, and 8, respectively. The main reason for this selection is that these systems are the most commonly used systems, which is also evidenced by the low number of SPEC entries with the remaining processors. We analyze the systems based on the processor type because we have observed that when different processor types are used, the system configurations were significantly different from each other, preventing us from making a relative comparison.

Table 1. Configurations used in microprocessor study

Parameters	Values
L1 Data Cache Size	16, 32, 64 KB
L1 Data Cache Line Size	32, 64 B
L1 Data Cache Associativity	4
L1 Instruction Cache Size	16, 32, 64 KB
L1 Instruction Cache Line Size	32, 64 KB
L1 Instruction Cache Assoc.	4
L2 Cache Size	256, 1024 KB
L2 Cache Line Size	128 B
L2 Cache Associativity	4, 8
L3 Cache Size	0, 8 MB
L3 Cache Line Size	0, 256 B
L3 Cache Associativity	0, 8
Branch Predictor	Perfect, Bimodal, 2-level, Combination
Decode/Issue/Commit Width	4, 8
Issue wrong	Yes, No
Register Update unit	128, 256
Load/Store queue	64, 128
Instruction TLB size	256, 1024 KB
Data TLB size	512, 2048 KB
Functional Units (ialu, imult, memport, fpalu, fpmult)	4/2/2/4/2, 8/4/4/8/4

An important property of the announcements is that even within a single processor family, the performance numbers showed significant variation: Opteron based systems has 138 records with a range of 1.40 times (i.e., the best system has 1.40 times better performance than the worst system) and variation of 0.08; Opteron 2 based systems have 152/1.58/0.11, Opteron 4 based systems have 158/1.70/0.12, Opteron 8 based systems have 58/1.68/0.13, Pentium D based systems have 71/1.45/0.10, Pentium 4 based systems have 66/3.72/0.34 and Xeon based systems have 216/1.34/0.09 records/range/variation values. The SPEC announcements contain information about the systems as well as execution times of each application. Each announcement provides the configuration of 32 system parameters: company, system name, processor model, bus frequency, processor speed, floating point unit, total cores (total chips, cores per chip), SMT (yes/no), Parallel (yes/no), L1 instruction and data cache size (per core/chip), L2 data cache size (on/off chip, shared/nonshared, unified/nonunified), L3 cache size (on/off chip, per core/chip, shared/nonshared, unified/nonunified), L4 cache size (# shared, on/off chip), memory size and frequency, hard drive size, speed and type, and extra components. Currently, there are 7032 announced results (3550 integer and 3482 floating-point).

4.2. Sampled Design Space Modeling

In this section, we present the prediction accuracy results for the sampled design space exploration. For these experiments, prediction models are created by randomly sampling 1% to 5% of the data and then using this data subset to build (train) the models. We then extract estimated error rates for each model using the approach presented in Section 3 and the true error rates are calculated using the entire data set. As described in Section 2, this approach can be used to reduce the design space size and hence accelerate the design space exploration. In Figures 2 through 6, we present the mean of the percentage prediction error, which is calculated by $100*|\hat{y}_i-y_i|/y_i$, where $\hat{y}_i$ is the predicted and y_i is the true (reported) number for the i^{th} record in the data used. In general, we observe that neural network models have better prediction rates than linear regression models. This relative success is expected because neural networks are better at modeling complex data sets.

Due to limited space, in this section we present the results for the best Linear Regression model (LR-B) and the best Neural Network model (NN-E), and a fast Neural Network model (NN-S). The general trend shows that as the training sample size increases from 1% to 5%, we obtain better prediction accuracy. This is due to the fact that the smaller training sets include less/insufficient information to capture the underlying complex relationship between design parameters. In some instances, we may observe that the error rates may have increased a little or stayed about the same when going to a higher training sample size. The main reason for this is the random selection of the training set. Even though the data selection is random, it is possible that the selected points may not be uniform through out the design space; hence the created model fits a portion of the space very accurately and not fitting the rest. Another point to observe is that Neural Network models generally have better prediction accuracy than Linear Regression models. This is due to the fact that linear models are inadequate to model the nonlinear changes and predictor interactions, while neural networks' complex data modeling capabilities provide a good fit of the results and hence highly accurate predictions. As it is seen in Figure 2, for the Applu application NN-E achieves 1.8% error rate when 1% of the design spaced is used in training. This rate drops below 1% error as the training data set size is increased to 2%. For NN-E, we observe a similar behavior for Equake (Figure 3) and Mcf (Figure 5) applications. On the other hand, Gcc (Figure 4) and Mesa (Figure 6) exhibit higher error rates. An interesting observation is that the accuracy of Neural Network models increases while the training data increases and very little change occurs for linear regression models. On average (over the all applications), NN-S method achieves 94.06% estimation accuracy at 1% sampling rate and the accuracy goes up to 98.22% when 3% sampling rate is used. NN-E, on the other hand, achieves 96.52% estimation accuracy on 1% sampling rate, which goes up to 99.08% at 3% sampling rate. Note that the NN-S method is similar to the model used by Ipek et al. [2].

Another observation is that the difference between the estimated error and the true error rates is generally small. The estimated error is smaller than the true error in some cases. They become very close to the true error rates after 3% sampling of the whole design space.

4.3. Chronological Predictive Modeling

In the previous section, we have shown that the predictive models achieve high prediction accuracies for estimating the performance of simulations. In this section, we present the success of the models when applied to performance prediction of manufactured systems. This section presents the results for chronological predictive models for single processor and multiprocessor based systems, which use historical performance announcements to predict the performance of future systems. We used the published results in 2005 to predict the performance of the systems that were built and reported in 2006.

In Figure 7, we present the prediction error for different Linear Regression and Neural Network models for Intel Xeon, Intel Pentium 4, and Intel Pentium D based systems. The percentage error is calculated as described in the previous section. The mean and standard deviation of the percentage prediction error are shown by circles and error bars, respectively. In general, we see that Linear Regression models perform better than Neural Networks. One of the main reasons for this is the fact that neural networks tend to over-fit to the data. In our case, the model built using 2005 data is very accurate for predicting 2005 data, however when we try to predict 2006, the over-fitting causes larger errors in estimations. However, linear regression does not have this problem and is successful in predicting 2006 results. In Figure 7, we see the best accuracy is achieved using with linear regression enter (LR-E) method of linear regression with an error rate of 2.1%, 1.5%, and 2.2% for Intel Xeon, Pentium 4, and Pentium D based systems, respectively. Figure 8a shows the results for Opteron based systems. The accuracies of the models are similar to the other single processor families. For Pentium D (Figure 7c), all the models perform about the same and produce roughly 2% error rate. The reason for this is that the data points are very similar to each other, because Pentium D results contain less than 2 years of data. Therefore, there have not been major changes in the systems and as a result, the neural network models are as successful as the linear regression models.

In Figure 8, we present the 1, 2, 4, 8 processor results for AMD Opteron based systems. The prediction accuracy results for the multiprocessor systems are similar to the single processor cases. A trend that we observe is that going to more complex systems, Opteron-2 (Figure 8b) to Opteron-4 (Figure 8c) to Opteron-8 (Figure 8d), we have a slightly higher minimum error rate of 3.1%, 3.2%, and 3.5%, respectively. These minimum error rates are achieved with the stepwise (LR-S) and backward (LR-B) methods. Here we see that LR-S/LR-B methods perform significantly better than the LR-E method. The reason for this is LR-E method uses all predictors as input and hence the model has a tighter fit to the training data, while LR-S method only adds a predictor to the model if it improves the quality of the model significantly. Likewise, LR-B method removes predictors if the predictor does not improve the quality of the model above a specified threshold. Hence, LR-S and LR-B methods converge to the same model in these cases. In these examples we see that LR-S/LR-B methods use lesser predictors than LR-E and perform better on the test (future) data. Another observation is that the neural networks perform poorer than linear regression models. Their prediction rate seems to get highly inaccurate as the number of processors in the system increases.

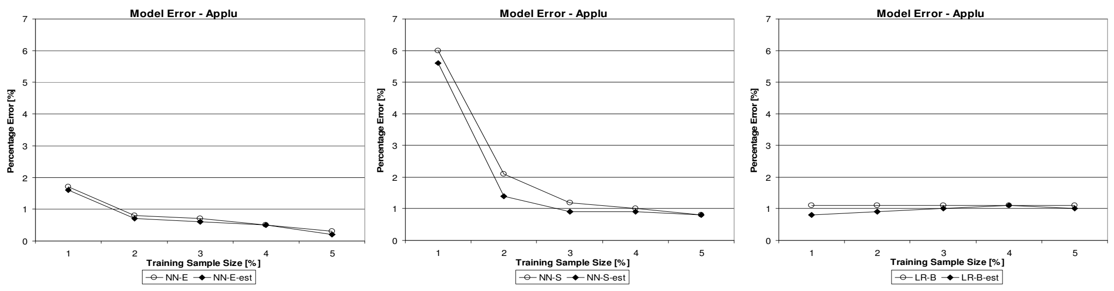


Figure 2. Estimated vs. true error rates for Applu application: NN-E (L), NN-S (M), LR-B (R)

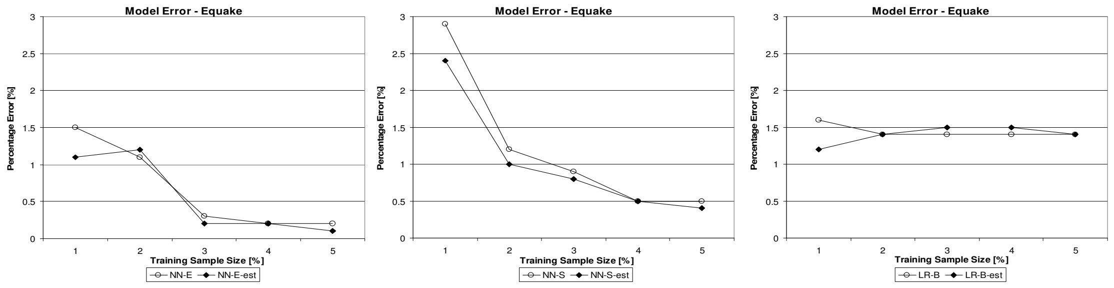


Figure 3. Estimated vs. true error rates for Equake application: NN-E (L), NN-S (M), LR-B (R)

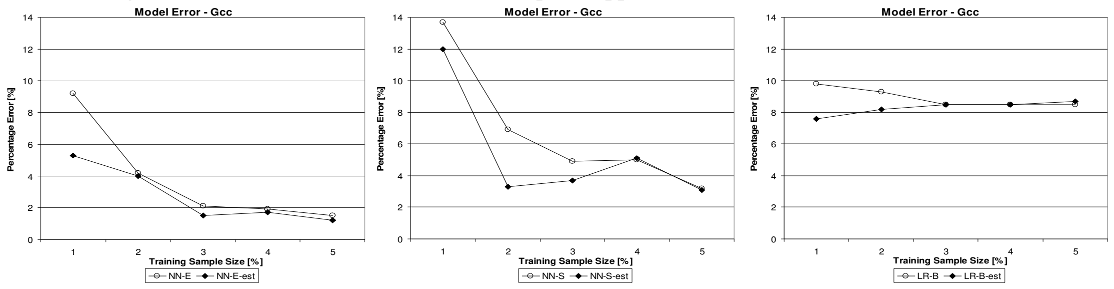


Figure 4. Estimated vs. true error rates for Gcc application: NN-E (L), NN-S (M), LR-B (R)

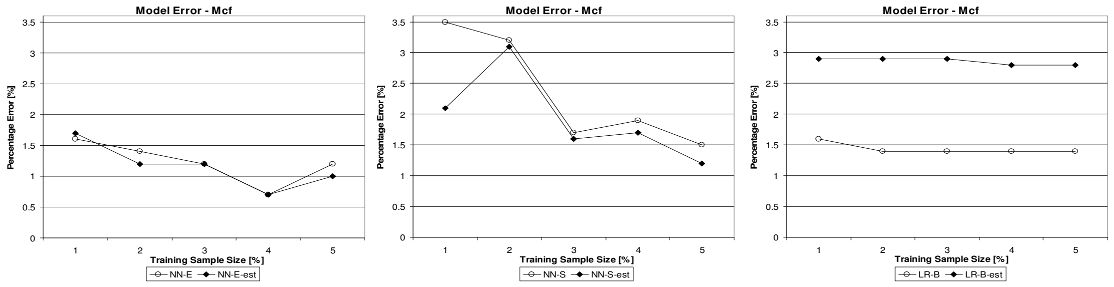


Figure 5. Estimated vs. true error rates for Mcf application: NN-E (L), NN-S (M), LR-B (R)

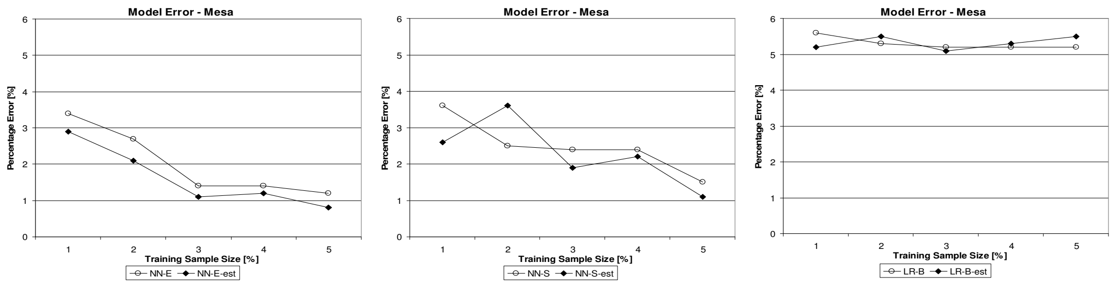
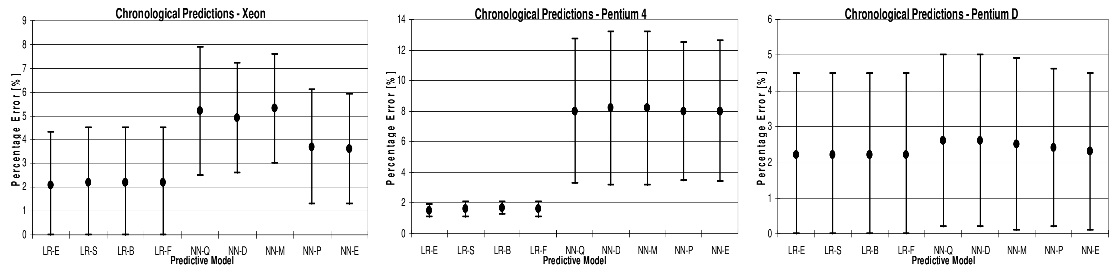
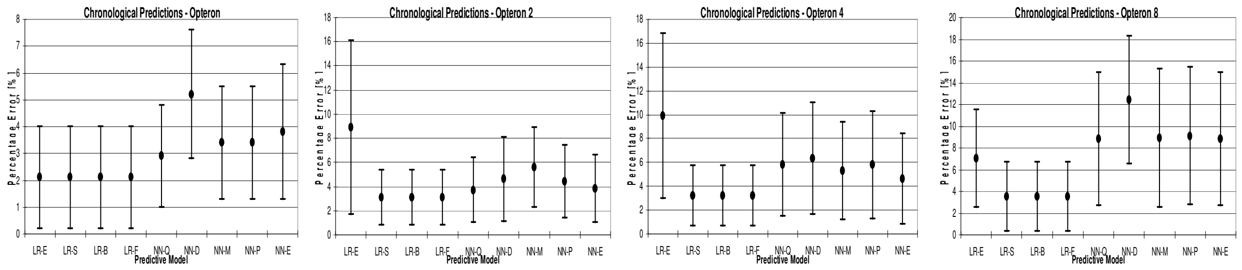


Figure 6. Estimated vs. true error rates for Mesa application: NN-E (L), NN-S (M), LR-B (R)



(a) **(b)** **(c)**

Figure 7. Chronological predictions for Xeon (a), Pentium 4 (b), Pentium D (c) based systems



(a) **(b)** **(c)** **(d)**

Figure 8. Chronological predictions for Opteron based multiprocessor systems: (a) one (b) two (c) four, and (d) eight processors

Table 2. The best accuracy and the model that achieves this for single and multi-processor chronological predictive modeling.

Prediction	Xeon	Pentium D	Pentium 4	Opteron	Opteron 2	Opteron 4	Opteron 8
Accuracy	2.1	2.2	1.5	2.1	3.1	3.2	3.5
Method	LR-E	LR-E	LR-E	LR-B/LR-S	LR-B/LR-S	LR-B/LR-S	LR-B/LR-S

Table 3. Average accuracy results from SPEC simulations

Statistics	1%	2%	3%	4%	5%
LR-B	4.2	4	3.82	3.8	3.8
NN-E	3.48	2.04	1.14	0.94	0.88
NN-S	5.94	3.18	2.22	1.16	1.5
Select	3.4	2.6	1.14	0.94	0.88

4.4. Summary

In the previous sections, we have presented the results for SPEC benchmark simulation results for the microprocessors and SPEC published results for real systems. For the sampled design space exploration using simulation results (Section 4.2), we see that generally neural network methods perform better than linear regression methods. When we compare the neural network models in themselves, exhaustive prune method has the best prediction accuracy. The summary of the results presented in Section 4.2 are shown in Table 3. The last row, select method, shows the error rates that would be achieved if the method that gives the best result on the estimation is used for predicting the whole data set. The results reveal that select method successfully finds the best model and uses it for the predictions. Note that for the 1% sampling rate, select method performs better than the NN-E. The reason for this is the Applu application; select method uses LR-B, which gives a better accuracy than NN-E. Therefore, the average accuracy of the select method is better than the NN-E method.

Table 2 summarizes the results presented in Section 4.3. In this table, we present the best accuracy achieved and the method that achieves it for different systems. As mentioned previously, linear regression models perform well, achieving error rates ranging between 1.5% and 3.5% on single and multiprocessor systems. The models described in Section 4.3 include 3 to 10 predictor variables. Within these, there are generally two or three factors that are significantly more important than others. The important factors and their order of importance change from a processor family to another. For example, for the Opteron systems, (Figure 8) the most important parameters for neural networks (with their relative importance presented in parenthesis) are processor speed (0.659), memory frequency (0.154), L2 being on chip or off chip (0.147), and L1 data cache size (0.139). Note that the importance factor denotes the relative importance of the input factor (0 denoting that the field has no effect on the prediction and 1.0 denoting that the field completely determines the prediction). For the same predictions, the linear regression model included processor speed and memory size with standardized beta coefficients of 0.915 and 0.119, respectively. Standardized beta coefficients show the relative importance of the predictor variable. While for Pentium D based systems (Figure 7c), the important factors used in the neural network model are processor speed (0.570), L2 cache size (0.500), L1 cache being shared or not (0.206), L2 cache being shared or not (0.154), L1 data cache size (0.145), and bus frequency (0.120). Linear regression, on the other hand, uses processor speed (0.733), L2 cache size (0.583), memory size (0.001), memory frequency (0.094), and L1 cache size (0.297). Overall, the combinations of all parameters aid in the high prediction rates we observe.

An important difference between the chronological and simulation analysis is that the number of available data points is usually small for chronological analysis. Hence the diversity for many of the components is hard to capture in the created models, higher error rates are seen. In simulation data, the huge number of points is created by keeping the input parameters as constant and changing only one parameter at a time. Even with sampling 1% of the data, the diversity can be captured, and it is easier for the predictive models to achieve higher accuracy on simulation data.

5. Related Work

There have been numerous works done in the area of design space exploration. Eyerman et al. [9] uses a different heuristics to model the shape of the design space of superscalar out-of-order processor. Ipek et al. [2] use artificial neural networks (with cross-validation to calculate their prediction accuracy) to predict the performance of memory, processor and CMP design spaces. Meanwhile, Lee et al. [3] use regression models to predict performance and power usage of the applications found in the SPECjbb and SPEC2000 benchmarks. As in the previous reference, the data points are created using simulations. Kahn et al. [15] uses predictive modeling, a machine learning technique to tackle the problem of accurately predicting the behavior of unseen configurations in CMP environment. Ghosh et al. [10] have presented an analytical approach to the design space exploration of caches that avoids exhaustive simulation. The problem that they are trying to solve (only varying cache size and associativity) is very small compared to the ones that other researches are trying to solve. Dubach et al. [16] has used a combination of linear regressor models in conjunction with neural networks to a model that can predict the performance of programs on any microarchitectural configuration with only using 32 further simulations. In this work, we target system performance rather than processor performance. All of these works have based their models on simulation while our results use simulation results and already built existing computer systems. To our knowledge there has not been any work done in this area. The closest work is by Ipek et al. [11], where they use artificial neural networks to predict the performance of SMG2000 applications run on multi-processor systems. The application inputs and the number of processors the application runs on are changed during their analysis. Their accuracy results are around 12.3% when they have 250 data points for training. However, we must point that they also do not estimate the performance of the systems but rather simulate the execution of an application on one system.

6. Conclusion

In this work, we have used two different machine learning techniques, linear regression and neural network, in the area of design space exploration. The design space exploration is an important task for all system manufacturers. The possible combinations of system parameters that can be set in design space exploration is generally huge and the models we generate in this work can be used to estimate the performance of various systems by using a small fraction of the design space as a training set. As a result, these models will reduce design and development cost. In our work, we have used the results from the SPEC2000 benchmark. We have first used a simulator to generate performance numbers for a selected number of applications from the benchmarks, and then used predictive modeling techniques to show that the design space can be efficiently modeled. Neural networks are generally better for this goal; the best neural network model (NN-E) achieves 96.5% accuracy on average when 1% of the design space is used. If error estimations are used to guide the selection for the prediction model, 96.6% accuracy is achieved for the same design space. Then, to show that our predictive models can be applied to the real world systems, we have used the published SPECint2000 numbers from the SPEC website. Using these performance numbers and the chronological prediction technique, we have shown that performance of future systems can be estimated with less than 2.2% error rate on average by using the data from previous systems. These results indicate that the designers can select a small portion of the design space to estimate the performance of new systems. Our models are also successful for estimating future system performance when limited data is available for the already built systems.

7. Acknowledgements

This work was supported in part by NSF grants CNS-0406341, CNS-0551639, IIS-0536994, CCR-0325207, CNS-0720691, IIS-0613568, CCF-0541337, and DOE's SCIDAC and FASTOS programs.

8. References

[1] Moya F., Moya J. M. and Lopez J. C., Evaluation of Design Space Exploration Strategies. In Proc. of the EUROMICRO Conference, Sep 1999, York, England
[2] Ipek E., McKee S. A., deSupinski B. R., Schultz M. and Caruana R. Efficiently Exploring Architectural Design Spaces via Predictive Modeling. In Proc. of the ASPLOS, Oct. 2006, San Jose, CA.
[3] Lee B. C. and Brooks D. M. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In Proc. of the ASPLOS, Oct 2006, San Jose, CA.
[4] The Standard Performance Evaluation Corporation, http://spec.org
[5] Tan P., Steinbach M. and Vipin K. Introduction to Data mining. Addison-Wesley, 2005.
[6] Ramakrishnan R. and Gehrke J. Database Management Systems. McGraw-Hill, 2000.
[7] Montgomery D. C., Peck E. A. and Vining G. C. Introduction to Linear Regression Analysis. Wiley, New York, NY, 2001.
[8] SPSS Clementine version 11, http://www.spss.com/clementine
[9] Eyerman S., Eeckhout L. and Bosschere K. D. The Shape of the Processor Design Space and its Implications for Early Stage Explorations. In Proc. of the Int. Conf. on ACMOS. Mar 2005, Prague, Czech Republic.
[10] Ghosh A. and Givargis T. Analytical Design Space Exploration of Caches for Embedded Systems. In the Proc. of the DATE Conference. Mar 2003, Munich Germany.
[11] Ipek E., de Supinski B. R., Schulz M. and McKee S. A. An Approach to Performance Prediction for Parallel Applications. In Proc. of the Euro-Par. May 2005, Monte de Caparica, Portugal.
[12] SimpleScalar Tool Set, http://www.simplescalar.com/
[13] Sherwood T., Perelman E., Hammerly G., and Calder B. Automaticall characterizing large scale program behaivour. In the Proc. of the ASPLOS. Oct 2002, San Jose, CA.
[14] Phansdalkar A., Joshi A., Eeckhout L., and John L. Measuring program similarity: Experiments with SPEC CPU benchmark suites. In Proc. of IEEE ISPASS, Mar 2005, Austin, TX.
[15] Khan S., Xekalakis P., Cavazos J. and Cintra M. Using Predictive Modeling for Cross-Program Design Space Exploration in Multicore Systems. In Proc. of the Int. Conf. on PACT, Sep 2007, Brasov, Romania.
[16] Dubach C., Jones T. M. and O'Boyle M. F. P. Microarchitectural Design Space Exploration Using An Architecture-Centric Approach. In Proc. of the Int. Symp. on Microarchitecture, Dec 2007, Chicago, IL.