

Efficient Architectural Design Space Exploration via Predictive Modeling

ENGIN İPEK, SALLY A. McKEE, KARAN SINGH and RICH CARUANA

Cornell University

and

BRONIS R. de SUPINSKI and MARTIN SCHULZ

Lawrence Livermore National Laboratory

Efficiently exploring exponential-size architectural design spaces with many interacting parameters remains an open problem: the sheer number of experiments required renders detailed simulation intractable. We attack this via an automated approach that builds accurate predictive models. We simulate sampled points, using results to teach our models the function describing relationships among design parameters. The models can be queried and are very fast, enabling efficient design tradeoff discovery. We validate our approach via two uniprocessor sensitivity studies, predicting IPC with only 1-2% error. In an experimental study using the approach, training on 1% of a 250K-point CMP design space allows our models to predict performance with only 4-5% error. Our predictive modeling combines well with techniques that reduce the time taken by each simulation experiment, achieving net time savings of three-four orders of magnitude.

Categories and Subject Descriptors: I.6.5 Computing Methodologies [**Simulation and Modeling**]: Model Development; B.8.2 Hardware [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms: Design, Experimentation, Measurement

Additional Key Words and Phrases: Artificial neural networks, design space exploration, performance prediction, sensitivity studies

1. INTRODUCTION

Quantifying the impact of design parameters on evaluation metrics and understanding tradeoffs and interactions among such parameters permeates the foundation of computer architecture. Architects rely on this understanding to perform cost-benefit analyses among alternative design options. Such analyses usually employ cycle-by-cycle simulation of a target machine either to predict performance impacts of architectural changes, or to find promising design subspaces satisfying different performance/cost/complexity/power constraints. Several factors have unacceptably increased the time and resources required for

Authors' addresses: Engin İpek, Sally A. McKee, and Karan Singh, Computer Systems Laboratory, Cornell University, Ithaca, NY 14853 USA; email {engin, sam, karan}@csl.cornell.edu; Rich Caruana, Computer Science Department, Cornell University, Ithaca, NY 14853 USA; email caruana@cs.cornell.edu; Bronis R. de Supinski and Martin Schulz, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Box 808, L-560, Livermore, CA 94551, USA; email {bronis, schulzm}@llnl.gov.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0000-0000/20YY/0000-0001 \$5.00

the latter task, including the desire to model more realistic workloads, the increasing complexity of modeled architectures, and the exponential design spaces spanned by many independent parameters. Thorough study of even relatively modest design spaces becomes challenging, if not infeasible [Martonosi and Skadron 2001; Jacob 2003; Davis et al. 2005].

Nonetheless, sensitivity studies of large design spaces are often essential to making good choices: for instance, Kumar et al. [2005] find that design decisions not accounting for interactions with the interconnect in a chip multiprocessor (CMP) are often opposite to those indicated when such factors are considered. Research on reducing the time required for individual experiments or on identifying the most important subspaces to explore within a full parameter space has significantly improved our ability to conduct more thorough studies. Even so, simulation times for thorough design space exploration remain intractable for most researchers.

One key point to take away from this work is that sampling design spaces at too coarse a granularity can lead us to draw inappropriate conclusions. The community inevitably relies on experience and intuition about architectural design spaces in choosing subspaces to study or in deciding how to apply the various tools at our disposal, but rapid changes in technologies and the growing complexities of both the architectural design spaces and workload configuration spaces suggest that our historical experiences and intuitions may no longer apply in the same ways. We need tools and methodologies to validate (or refute) our intuitions as we move into an era of multi-clock domain, many-core systems (for instance) that execute increasingly complex software stacks.

We attack these design space problems by using artificial neural networks (ANNs) to predict performance for most points in very large design spaces. We view the architecture simulator as a nonlinear function of its M -parameter configuration: $SIM(p_0, p_1, \dots, p_M)$. Instead of sampling this function at every point (parameter vector) of interest, we employ powerful nonlinear regression to approximate it. We repeatedly sample small numbers of points in the design space, simulate them, and use the results to teach the ANNs to approximate the function. At each teaching (training) step, we obtain highly accurate error estimates of our approximation for the full space. We continue refining the approximation by training the ANNs on further sample points until error estimates drop sufficiently low.

By training the ANNs on 1-2% of a design space, we predict results for other design points with high accuracies (98-99% for our uniprocessor studies, and 95-96% for our CMP study). The ANNs are extremely fast compared to simulation (training typically takes a few minutes), and our approach is fully automated. Combining our models with SimPoint [Sherwood et al. 2002] reduces required CPU time by three-four orders of magnitude, enabling detailed study of architectural design spaces previously beyond the reach of current simulation technology. This allows the architect to purge uninteresting design points quickly and focus detailed simulation on the most promising design subspaces.

Most importantly, our approach fundamentally differs from heuristic search algorithms in scope and use. It can certainly be used for optimization (predicted optimum with 1% sampling of the design spaces is within 3% of globally optimal performance for applications in our processor and memory system studies), but we provide a superset of the capabilities of heuristics that intelligently search design spaces to optimize an objective function (e.g., those studied by Eyerman et al. [2005]). Specifically, our technique:

- generates accurate predictions *for all points in the design space*. Unlike heuristic search techniques, our models can be queried to predict performance impacts of archi-

tectural changes, enabling efficient discovery of tradeoffs among parameters in different regions.

- provides highly accurate (typically within 1-2%) error estimates. These increase confidence in results, and provide a well crafted knob for the architect to control the accuracy-speed tradeoff inherent in architectural modeling.

- verifies that apparent performance gains from a novel proposal are not mere artifacts of other parameters chosen.

- allows architects to observe the sensitivity of a proposal to interacting parameters. This allows more thorough evaluation, increasing confidence in novel architectural ideas.

Since all regression methods (including ANNs) ultimately rely on interpolation between training samples, a global optimum that is an outlier with much higher performance than any of the training data is unlikely to be predicted with high accuracy. This limitation does not constitute an important drawback to our approach: our primary goal is not finding a single point that optimizes an objective function, but rather building models that can predict performance for *all* points in the design space. This allows architects to:

- compute correlations among parameters in different regions using statistical tests.

- query the models to find high-performance regions of the space that satisfy a cost or power budget.

- visualize query results to inspect trends, trade-offs, and performance sensitivity across the design space.

- calculate interaction costs Fields et al. [2004] among all parameters with high accuracy.

- evaluate novel architectural features across large design spaces, leading to more thorough and objective comparisons among a multitude of alternatives.

- observe performance plateaus to find the right combination of architectural features that can be appropriately downsized to reduce area overheads with minimal performance loss.

- diagnose performance bottlenecks (possibly involving multiple parameters) rapidly.

After training on 2% of our design spaces, querying our models to identify design points within 10% of the predicted optimal IPC purges over 80% of the design points. Querying again to identify points within a given power budget, for instance, could eliminate comparable portions of remaining subspaces. Inspection of these spaces can then provide insight to guide subsequent design decisions.

Our approach is truly a design space exploration method, not simply an optimization technique (many good such techniques already exist). Given appropriate training data, our models can answer queries such as “which points in the design space are likely to provide a given level of performance, not exceed a given power budget, not exceed a given cost, fit in a specified footprint, and never exceed a given temperature?” We expect that many quite different design points will meet such criteria, but here we present the validation of our approach. Future work will use the approach to answer such questions in the design of next-generation systems.

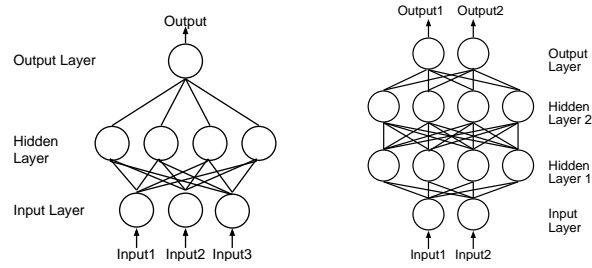


Fig. 1. Simplified diagrams of fully connected, feed-forward ANNs

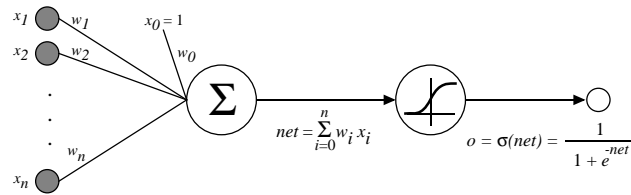


Fig. 2. Example of a hidden unit with a sigmoid activation function

2. ANN MODELING OF DESIGN SPACES

Artificial neural networks (ANNs) are machine learning models that automatically learn to predict targets (here, simulation results) from a set of inputs. ANNs constitute a powerful, flexible method for generalized nonlinear regression. They have been used in research and commercially to guide autonomous vehicles [Pomerleau 1993], to play backgammon [Tesauro 1995] and to predict weather [Marzban 2000], stock prices, medical patient outcomes, and horse races. With respect to computer systems, Bigus [1994b] uses them to predict job class response times in a simulated multiprogrammed environment and to tune system performance [Bigus 1994a], and Yu et al. [2002] automatically optimize I/O request handling. The representational power of ANNs is rich enough to express complex interactions among variables: any function can be approximated to arbitrary precision by a three-layer ANN [Mitchell 1997]. We choose ANNs over other predictive approaches (such as linear or polynomial regression and Support Vector Machines [SVMs]) for modeling parameter spaces in computer architecture because:

- (1) they represent a mature, already commercialized technology;
- (2) they do not require the form of the functional relationship between inputs and target values to be known;
- (3) they operate with real-, discrete-, cardinal-, and boolean-valued inputs and outputs, and thus can represent parameters of interest to an architect; and
- (4) they work well with noisy data, and thus can successfully be combined with existing mechanisms that reduce the time simulation experiments take at the expense of introducing noise.

Figure 1 shows the basic organization of simple *fully connected, feed-forward* ANNs. Networks consist of an *input layer*, *output layer*, and one or more *hidden layers*. Input

values are presented at the input layer; predictions are obtained from the output layer. Each unit operates on its inputs to produce an output that it passes to the next layer. In fully connected feed-forward ANNs, weighted edges connect every unit in each layer to all units in the next layer, communicating outputs to other units downstream. A unit calculates its output by applying its *activation function* to the weighted sum (based on edge weights) of all inputs. Figure 2 (borrowed from Mitchell [1997]) depicts a hidden unit using a sigmoid activation function. In general, activation functions need not be sigmoid, but must be nonlinear, monotonic, and differentiable. Our models use sigmoid activation functions.

Model edge weights are updated via backpropagation, using gradient descent in weight space to minimize squared error between simulation results and model predictions. Weights are initialized near zero, causing the network to act at first like a linear model. During training, examples are repeatedly presented at the inputs, differences between network outputs and target values are calculated, and weights are updated by taking a small step in the direction of steepest decrease in error. As its weights grow, an ANN becomes increasingly nonlinear. Every network weight $w_{i,j}$ (where i and j correspond to processing units) is updated according to Equation 1, where E stands for squared-error and η is a small *learning rate* constant (effectively the gradient descent step size).

$$w_{i,j} \leftarrow w_{i,j} - \eta \frac{\partial E}{\partial w_{i,j}} \quad (1)$$

A *momentum* term in the backpropagation update rule of Equation 2 causes weight updates in the current gradient descent iteration to include a fraction of the previous iteration’s update. This helps the search continue “rolling downhill” past inferior local minima. Momentum accelerates gradient descent in low-gradient regions and damps oscillations in highly nonlinear regions.

$$w_{i,j} \leftarrow w_{i,j} - \left(\eta \frac{\partial E}{\partial w_{i,j}} + \alpha \Delta w_{i,j} (n - 1) \right) \quad (2)$$

2.1 Training

Machine learning models require some type of training experience from which to learn. Here we use *direct* training examples consisting of the design space parameter vector (to the simulator function) along with IPCs from simulation results. Training an ANN involves learning edge weights from these examples. The weights associated with each edge in an ANN define the functional relationship between input and output values. For example, to learn to predict IPC from L1 and L2 cache sizes and front-side bus bandwidth, the architect runs a number of cycle-by-cycle simulations for combinations of these parameters and collects the parameters and resulting IPCs into a *training* dataset. ANN weights are adjusted based on these data until the ANN accurately predicts IPC from input parameters. Obviously, a good model must accurately predict parameter combinations on which it was not trained. Equally obvious is the need to train and test on separate datasets.

The ANN parameters that most impact learning are number of hidden layers, number of hidden units per layer, learning rate (gradient descent step size), momentum, and distribution of initial weights. Finding settings that perform well is typically straightforward, requiring a small amount of experimentation around typical values. Across all benchmarks, we use one 16-unit hidden layer, learning rate of 0.001, and momentum value of 0.5, initializing weights uniformly on $[-0.01, +0.01]$. These parameters can be tuned automatically.

MODEL	FOLD									
	1	2	3	4	5	6	7	8	9	10
1	ES	TS	TR	TR	TR	TR	TR	TR	TR	TR
2	TR	ES	TS	TR	TR	TR	TR	TR	TR	TR
3	TR	TR	ES	TS	TR	TR	TR	TR	TR	TR
4	TR	TR	TR	ES	TS	TR	TR	TR	TR	TR
5	TR	TR	TR	TR	ES	TS	TR	TR	TR	TR
6	TR	TR	TR	TR	TR	ES	TS	TR	TR	TR
7	TR	TR	TR	TR	TR	TR	ES	TS	TR	TR
8	TR	TR	TR	TR	TR	TR	TR	ES	TS	TR
9	TR	TR	TR	TR	TR	TR	TR	TR	ES	TS
10	TS	TR	TR	TR	TR	TR	TR	TR	TR	ES

Fig. 3. Example of 10-fold cross validation ensemble on 1K training points—ES/TS/TR indicate early stopping, test, and training sets, respectively

2.2 Cross Validation

In polynomial curve fitting, polynomials of high degree yield models that have excellent fit to the training samples yet interpolate poorly; likewise, ANNs may *overfit* to training data. Overfitting yields models that generalize poorly to new data even though they perform well on training data. In contrast to polynomial curve fitting, where model complexity is reduced by decreasing polynomial degree, larger networks for which training is halted *before* gradient descent reaches the minimum error on the training set generally make better predictions (this is called *early stopping*) [Caruana et al. 2000]. We address this by using an ensemble of models. Each is initialized with random weights near 0, but each is trained and tested on different datasets, as described below.

We reserve a portion of the training set and prevent overfitting by stopping gradient descent when squared error on this unbiased sample stops improving. To do this, we allocate part of our data as the “early stopping set”, and train on the remaining points for a fixed number of passes (called “epochs”, 30K in our case) through the training set. At the end of each epoch, we evaluate each ANN’s performance on the early stopping set, retaining individual models with lowest squared error at each step along the process.

Unfortunately, if 25% of the data are used as this early stopping set, the training set for gradient descent is 25% smaller, and as with other regression methods, ANNs learn less accurate models from reduced training samples. *Cross validation* both avoids this problem and allows us to estimate model accuracy.

In cross validation, the training sample is split into multiple subsets or *folds*. In our case, we divide training samples into 10 folds, each containing 10% of the training data. Folds 1-8 (80% of the data) are used for training a first ANN; fold 9 (10% of the data) is used for early stopping to avoid overfitting to the training data; and fold 10 (also 10% of the data) is used for estimating performance of the trained model. We train a second ANN on folds 2-9, use fold 10 for early stopping, and use fold 1 to estimate accuracy. This process is repeated for other ANNs, using the data in each fold successively as training, early stopping, and test sets. Figure 3 shows the assignments of folds to ANNs in the modeling process.

We combine the resulting 10 networks into an ensemble, averaging their predictions. Each ANN is trained on 80% of the data, but all data are used to train models in the final ensemble. This ensemble performs similarly to models trained on all data, yet held-aside

data are available for early stopping and unbiased error estimation. Using ensembles (i.e., averaging multiple models) usually performs better than using a single ANN. Means and standard deviations of model error on the test folds are used to estimate ensemble accuracy, allowing the architect to determine when the models are accurate enough to be useful. Such cross validation is considered “best practice” in the Machine Learning community.

2.3 Modeling Architectural Design Spaces

Parameters in architectural design spaces can be grouped into a few broad categories. *Cardinal* parameters indicate quantitative relationships (e.g., cache sizes, or number of ROB entries). *Nominal* parameters identify choices, but lack quantifiable properties among their values (e.g., front-end fetch policy in SMTs, or type of coherence protocol in CMPs). Continuous (e.g., frequency) and boolean (e.g., on/off states of power-saving optimizations) parameters are also possible. The encoding of these parameters and how they are presented to ANNs as inputs significantly impact model accuracy. We encode each cardinal or continuous parameter as a real number in $[0,1]$, normalizing with minmax scaling via minimum and maximum values over the design space. Using a single input facilitates learning functional relationships involving different regions in the parameter’s range, and normalization prevents placing more emphasis on parameters with broader ranges. We represent nominal parameters with one-hot encoding: we allocate an input unit for each parameter setting, making the input corresponding to the desired setting 1 and those corresponding to other settings 0. This avoids erroneous encoding of range information where none exists. We represent boolean parameters as single inputs with 0/1 values. Target values (simulation results) for model training are encoded like inputs. We scale normalized IPC predictions back to the actual range. When reporting error rates, we perform calculations based on actual (not normalized) values.

When exploring a design space, absolute value of the model error is of little use. For instance, in predicting execution time, erring by one second is negligible for actual time of an hour but significant for actual time of two seconds. When absolute errors have differing costs, ANNs can be trained by presenting points with higher costs more often. *Stratification* replicates each point in the dataset by a factor proportional to the inverse of its target value so that the network sees training points with small target values many more times than it sees those with large absolute values. The training algorithm will then put varying amounts of emphasis on different regions of the search space, making the right tradeoffs when setting weights to minimize percentage error. We stratify before normalizing, and we perform early stopping based on percentage error.

2.4 Intelligent Sampling

Sample points for training models affect how quickly they learn to approximate the simulator function accurately. Randomly sampling simulation data from the design space yields good performance, but choosing sample points intelligently yields better predictions. *Active learning* [Saar-Tsechansky and Provost 2001] is a general class of algorithms that aim for a given accuracy with fewest samples by selecting the points from which the model is likely to derive the most benefit. We seek to identify samples on which the model makes the greatest error, since learning these points is most likely to improve model accuracy. Unfortunately, assessing model error on any point requires knowing simulation results for that point. Since results are unavailable before simulation, identifying points likely to have highest error requires an alternative strategy. We thus measure the variance

- (1) Identify important design parameters.
- (2) Perform a set of simulations for N combinations of parameter settings, possibly reducing the time for each simulation by using statistical simulation techniques (e.g., SimPoint).
- (3) Normalize inputs and outputs. Encode nominal parameters with one-hot encoding, booleans as 0-1, and others as real values in the normalized 0-1 range. Collect the results in a data set.
- (4) Divide data set into k folds.
- (5) Train k neural nets with k -fold cross validation. During training, present each data point to the ANNs at a frequency proportional to the inverse of its IPC (we assume the target to be predicted is IPC; other targets are similar). Perform early stopping based on percentage error.
- (6) Estimate average and standard deviation of error from cross validation.
- (7) Repeat 2-6 with N additional simulations if estimated error is too high.
- (8) Predict any point in the parameter space by placing the parameters at the input layers of all ANNs in the ensemble, and averaging predictions of all models.

Fig. 4. Summary of steps in modeling mechanism

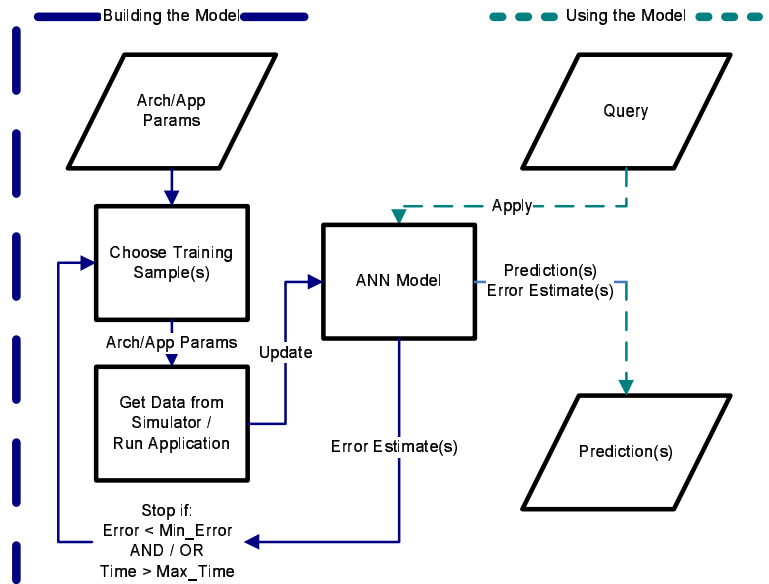


Fig. 5. Pictorial representation of modeling approach (training versus testing)

of the predictions for all ANNs in our cross validation ensemble. After each training round, we query all ANNs for predictions on every point in the design space (testing takes under 10 seconds for 20K+ points). We calculate the variance of model predictions, the mean prediction, and the prediction's coefficient of variance (CoV, or the ratio of the standard deviation to mean). Points with high prediction CoV values are those for which disagreement among the ANNs is largest. Since the ensemble's ANNs differ primarily by the samples (folds) on which they are trained, high disagreement indicates that including the point in the sample set can lower model variance (and thus error). In effect, active learning assigns confidence values based on predictions of the 10 models from our 10-fold cross validation, and then samples the least confident points.

Impacts of different points on model accuracy are dependent on one another. Sorting and sampling strictly according to CoV values can yield a dataset with redundant points. One point may improve the variance of another if these points lie close together in the

Table I. Performance swings across processor and memory system design spaces for each application. Each entry shows the ratio of the maximum and minimum IPC values.

Application	Sensitivity Study	
	Memory System	Processor
gcc	1.62	1.67
mcf	3.36	2.59
crafty	1.85	1.68
vortex	2.28	1.69
bzip2	1.57	1.78
twolf	3.40	2.44
swim	1.60	2.54
mgrid	2.28	3.38
applu	2.16	2.45
mesa	1.73	1.33
art	14.18	4.31
equake	1.77	1.80

Table II. Performance swings across the CMP design space for each application. Each entry shows the ratio of the maximum and minimum performance values.

Application	CMP Study
swim	41
art	25
mg	55
fft	37
radix	65
ocean	35

design space: sampling both such points is inefficient, since the second presents little additional information. We address this by iteratively choosing samples from the sorted points. We first include the least confident point. We include the next least confident point if its euclidian distance (in the design space) to the first point is above a certain threshold. We include each point considered if its distance to currently selected points is above threshold. If we do not find enough points satisfying this constraint, we lower the threshold and reconsider rejected points. The threshold is initialized to the maximum distance between any two points in the space. Once we have enough sample points, we simulate them and train new models. If our error estimate is too high, we calculate a new sample set through the active learning mechanism. Figure 4 summarizes our modeling mechanism for random sampling or active learning. Figure 5 provides a graphic perspective to differentiate steps in training versus using the model.

3. EXPERIMENTAL SETUP

We conduct performance sensitivity studies on memory system, microprocessor, and multi-threaded chip multiprocessor (CMP) parameters via detailed simulation of an out-of-order processor and its memory subsystem (SESC) [Renau 2002]. We model contention and latency at all levels. Phansalkar et al. [2005] use principal component analysis to group SPEC CPU 2000 applications according to metric similarities like instruction mix, branch direction, and data locality. For the memory system and processor studies, we choose codes

(gcc, mcf, crafty, vortex, bzip2, twolf, swim, mgrid, applu, mesa, art, and quake) to cover the clusters they identify in an effort to model the diverse program behaviors represented by the suite. Since we run over 500K simulations, we choose MinneSPEC [KleinOsowski and Lilja 2002] reduced reference inputs to validate our approach (results for SPEC reference inputs would be even more impressive in terms of time saved). We run all simulations, then sample these incrementally to train the ANNs, predicting remaining points in the space at each round, and validating predictions against the simulation results. For the CMP study, we use two applications (swim and art) from the SPEC OMP V3.0 suite and four applications (mg, fft, ocean, and radix) from the parallel NAS benchmarks. We use MinneSPEC and Class-S inputs for SPEC OMP and NAS benchmarks, respectively. We train our models on 1.03% of the design space (2500 simulations), and report accuracies obtained on an independently sampled set of 500 points.

We assume a 90nm technology for the processor and memory system studies and a 65nm technology for the CMP study. We derive all cache latencies with CACTI3.2 [Wilton and Jouppi 1996]. Table III shows parameters in the memory hierarchy study, the design space for which spans the cross product of all parameter values and requires 23,040 simulations per benchmark. Core frequency is 4GHz. The L2 bus runs at core frequency and the front-side bus is 64 bits. Table IV shows parameters in the microprocessor study, which requires 20,736 simulations per benchmark. We use core frequencies of 2GHz and 4GHz, and calculate cache and SDRAM latencies and branch misprediction penalties based on these. We use 11- and 20-cycle minimum latencies for branch misprediction penalties in the 2GHz and 4GHz cases, respectively. For register files, we choose two of the four sizes in Table IV based on ROB size (e.g., a 96 entry ROB makes little sense with 112 integer/fp registers). When choosing the number of functional units, we choose two sizes from Table IV based on issue width. The number of load, store and branch units is the same as the number of floating point units. SDRAM latency is 100ns, and we simulate a 64-bit front-side bus at 800MHz. The design space for the CMP study spans the cross product of all parameters in Table V, which would require 241,920 simulations per benchmark for a full sensitivity study (unlike the other spaces, we have not simulated this immense design space exhaustively). We vary number of cores and SMT contexts, microarchitectural parameters, and parameters of the shared-memory subsystem. These parameters cover a wide range of current and upcoming system organizations [Kongetira et al. 2005; Ramanathan 2006; Borkar et al. 2006]. Table I and Table II show performance swings across processor, memory system, and CMP design spaces for all applications under study.

Partial simulation techniques (modeling only certain execution intervals or simulation points) reduce experiment time at the expense of slight losses in accuracy. In SimPoint, Sherwood et al. [2002] combine *basic block distribution analysis* with clustering to summarize behavior of sections of program execution; this guides selection of representative samples to simulate in detail. Such techniques induce slight errors in results, the amount of which varies across the parameter space. When combining techniques like SimPoint with our approach, the ANNs see noisy results where the precise amount of error depends on the simulation technique, its parameters, and design space parameters. To evaluate our approach with noisy (but quickly obtained) simulation results, we repeat the processor study with SimPoint and our four longest-running applications (mesa, quake, mcf, crafty). We scale SimPoint intervals from 100M to 10M dynamic instructions to adjust for shorter run-times from using the MinneSPEC input sets. Otherwise, we run SimPoint out-of-the-box.

Table III. Parameter values in memory system study

Variable Parameters	Values
L1 DCache Size	8,16,32,64 KB
L1 DCache Block Size	32,64 B
L1 DCache Associativity	1,2,4,8 Way
L1 Write Policy	WT,WB
L2 Cache Size	256,512,1024,2048 KB
L2 Cache Block Size	64,128 B
L2 Cache Associativity	1,2,4,8,16 Way
L2 Bus Width	8,16,32 B
Front Side Bus Frequency	0.533,0.8,1.4 GHz
Fixed Parameters	Value
Frequency	4GHz
Fetch/Issue/Commit Width	4
LD/ST Units	2/2
ROB Size	128 Entries
Register File	96 Integer/96 FP
LSQ Entries	48/48
SDRAM 100 ns	64 bit FSB
L1 ICache	32 KB/2 Cycles
Branch Predictor	Tournament (21264)

Table IV. Parameter values in the processor study

Variable Parameters	Values
Fetch/Issue/Commit Width	4,6,8 Instructions
Frequency	2,4 GHz (affects Cache/DRAM/ Branch Misprediction Latencies)
Max Branches	8,32
Branch Predictor	1K,2K,4K Entries (21264)
Branch Target Buffer	1K,2K Sets (2-Way)
ALUs/FPUs	2/1,4/2,3/1,6/3,4/2,8/4 (2 choices per Issue Width)
ROB Size	96,128,160
Register File	64,80,96,112 (2 choices per ROB Size)
Ld/St Queue	16/16,24/24,32/32
L1 ICache	8,32KB
L1 DCache	8,32KB
L2 Cache	256,1024KB
Fixed Parameters	Value
L1 DCache Associativity	1,2 Way (depends on L1 DCache Size)
L1 DCache Block Size	32B
L1 DCache Write Policy	WB
L1 ICache Associativity	1,2 Way (depends on L1 ICache Size)
L1 ICache Block Size	32B
L2 Cache Associativity	4,8 Way (depends on L2 Cache Size)
L2 Cache Block Size	64B
L2 Cache Write Policy	WB
Replacement Policies	LRU
L2 Bus	32B/Core Frequency
FSB	64bits/800 MHz
SDRAM	100ns

Table V. Parameter values in the CMP study

Variable Parameters	Values
Core Configuration	In-Order, Out-of-Order
Issue Width	1,2,4
Number of Cores	1,2,4,8
SMT Contexts per Core	1,2,4
Off-Chip Bandwidth	8,16,24,32,40,48,56,64 GB/s
Frequency	1,1.5,2,2.5,3,3.5,4 GHz (affects Cache/DRAM/ Branch Misprediction Latencies)
L2 Cache Size	1,2,4,8MB
L2 Cache Block Size	32,64,128B
L2 Cache Associativity	1,2,4,8,16 Way
Fixed Parameters	Value
ROB Size	24,48,96 (depends on Issue Width)
Int/FP Issue Queue Sizes	12/12,24/24,48/48 (depends on Issue Width)
LD/ST Queue Sizes	6/6,12/12,24/24 (depends on Issue Width)
Int/FP Rename Registers	24/24,48/48,96/96 (depends on Issue Width)
L2 Bus	64B/Core Frequency
L1 ICache	32KB,2 Way,32B,LRU (latency depends on Core Frequency)
L1 DCache	32KB,2 Way,64B,LRU (latency depends on Core Frequency)
L2 Cache	Shared,Unified,8 Banks (latency depends on Core Frequency and L2 Parameters)
Branch Predictor	Tournament
SMT Fetch Policy	Round Robin
SDRAM	80ns Uncontended

4. EVALUATION

We address four questions:

- (1) How much of the design space must we simulate to train our models?
- (2) How accurate and robust are our predictions versus full simulation?
- (3) How fast can we train the models?
- (4) How well does our approach integrate with other approaches to reduce time per simulation?

We answer these via four sensitivity studies: the memory hierarchy and processor studies with random sampling; the memory study with active sampling; and the processor study with SimPoint. We also apply our approach to the CMP design space described in Section 3. Here we predict performance because it's well understood, but our approach is general enough to apply to other metrics, even multiple metrics at once.

Several mechanisms demonstrate that our design spaces are complex and nonlinear. First, linear regression yields high error (15-20%) for even dense (8%) samplings: simple linear models are inadequate. Second, we investigate the learned model: edge weights deviate greatly from their initial values near zero, indicating the model is a highly nonlinear

Table VI. Memory hierarchy study results (random sampling)

	Memory System Study											
	1.08% Sample				2.17% Sample				4.12% Sample			
	Mean Error %		SD of Error %		Mean Error %		SD of Error %		Mean Error %		SD of Error %	
App.	True	Est.	True	Est.	True	Est.	True	Est.	True	Est.	True	Est.
gcc	3.69	4.13	4.02	5.46	1.50	1.49	1.44	1.30	1.13	1.14	0.97	1.09
mcf	4.61	4.53	5.6	5.73	2.84	3.06	2.94	3.61	1.74	1.77	1.59	1.68
crafty	2.16	2.45	2.10	2.38	1.17	1.29	1.10	1.33	0.87	0.96	0.77	0.91
vortex	4.53	4.65	4.63	5.19	2.90	2.96	3.07	2.77	1.90	2.07	1.99	2.14
bzip2	1.95	1.90	1.84	1.82	0.97	0.94	0.86	0.81	0.59	0.61	0.48	0.52
twolf	2.25	2.40	3.57	3.25	1.10	1.15	1.15	1.31	0.70	0.75	0.74	0.82
swim	0.66	0.75	0.52	0.55	0.57	0.55	1.50	0.48	0.54	0.54	0.45	0.46
mgrid	4.96	5.19	6.12	6.43	1.53	1.52	1.40	1.79	0.83	0.85	0.74	0.75
applu	3.11	2.97	2.74	2.79	2.35	2.57	1.90	2.32	1.28	1.31	1.04	1.21
mesa	2.85	2.8	4.27	5.24	2.69	2.73	4.16	4.77	1.97	2.15	2.87	3.79
art	6.63	6.83	5.23	5.99	4.69	4.82	4.29	4.45	2.92	3.05	2.86	3.09
equake	2.32	2.47	3.28	4.58	1.40	1.39	1.81	1.61	0.92	0.92	0.97	0.98

Table VII. Microprocessor study results (random sampling)

	Processor Study											
	0.96% Sample				1.93% Sample				4.10% Sample			
	Mean Error %		SD of Error %		Mean Error %		SD of Error %		Mean Error %		SD of Error %	
App.	True	Est.	True	Est.	True	Est.	True	Est.	True	Est.	True	Est.
gcc	1.88	1.97	1.48	1.31	1.09	1.05	0.88	0.95	0.59	0.59	0.49	0.52
mcf	1.67	1.71	1.38	1.51	1.20	1.26	0.99	1.13	0.94	0.99	0.83	0.90
crafty	2.65	2.75	2.03	2.07	1.53	1.61	1.25	1.34	0.78	0.81	0.66	0.65
vortex	2.90	3.46	2.17	2.68	1.39	1.52	1.11	1.34	0.88	0.93	0.71	0.78
bzip2	1.30	1.50	0.95	1.16	0.79	1.86	0.61	0.66	0.56	0.57	0.44	0.46
twolf	1.31	1.40	1.02	1.05	0.85	0.99	0.66	0.79	0.57	0.61	0.44	0.50
swim	2.65	1.49	2.05	1.88	1.22	1.45	0.94	1.31	0.59	0.57	0.49	0.49
mgrid	1.39	1.16	1.13	0.89	0.99	1.00	0.75	0.78	0.74	0.80	0.59	0.66
applu	1.94	1.85	1.45	1.43	1.30	1.29	0.99	1.04	0.87	0.89	0.72	0.82
mesa	2.57	2.851	1.96	2.06	1.27	1.33	0.99	1.05	0.87	0.94	0.69	0.79
art	2.41	2.34	1.91	2.00	1.67	1.74	1.45	1.53	1.29	1.29	1.12	1.18
equake	1.80	1.89	1.39	1.47	1.15	1.99	0.94	1.00	0.72	0.73	0.59	0.64

function of the design parameters. Third, we perform multi-dimensional scaling (MDS) analysis on our datasets [Hastie et al. 2001], finding many local minima with respect to error predictions in different regions (many over 90% of the globally optimal IPCs).

4.1 Training Set Size

Like other regression methods, ANNs typically predict better when trained on more data. Data collection in architecture design space exploration is expensive: a tradeoff exists between number of simulations and model accuracy. For the memory and processor studies, we increment our datasets by 50 simulations at each training step. After each training round, we test the ANNs on remaining points in the space, recording mean percentage error and standard deviation of error and tracking cross-validation estimates for these metrics.

Table VI through Table VII summarize results for the memory system and microprocessor studies with randomly selected training samples. We show mean and standard deviation of error with cross-validation estimates for training sets corresponding roughly to 1%, 2%,

Table VIII. Reduction factors in number of sample points (active sampling vs. random sampling)

Accuracy	Application								
	applu	mcf	mgrid	mesa	equake	crafty	vortex	bzip2	gcc
98%	1.09	1.23	1.17	1.17	1.15	1.18	1.32	1.0	1.0
97%	1.51	1.29	1.13	2.55	1.0	1.0	1.0	1.15	1.15

and 4% of the full spaces. Preliminary experiments indicate that uniformly sampling the space tends to deliver slightly better results than random sampling, but not as good as intelligent sampling via active learning. Table VIII shows the reduction factors in numbers of required samples when using active learning versus random sampling and training the ANNs within error rates of 2-3% for nine applications. For tighter accuracy requirements, we expect active sampling to outperform random: the latter generally requires more samples to deliver comparable results. In most cases we reduce the number of samples, but sometimes there is no change. Nonetheless, active learning never increases sampling requirements, and studying intelligent sampling strategies is part of ongoing research.

4.2 Accuracy for the Uniprocessor Studies

Learning curves in Figures 6 and 7 illustrate percentage error rate decreases in the memory system and processor studies as training set sizes increase (via random sampling) for our integer and floating point applications, respectively. The x axes show portions of the full parameter spaces simulated to form training sets, and the y axes show percentage error across the design spaces. Error bars indicate ± 1 standard deviation of the averages.

For the memory system study, training on 0.22% of the full design space (50 training examples) yields average error between 5-10%, with standard deviation of error typically between 10-15% across all applications. These are unacceptably high. The small training set includes insufficient information to capture the functional relationship between design parameters and performance. Standard deviation of error is high, and model accuracy varies significantly from one region of the design space to another, indicating that sampling is too sparse. For the applications in Figures 6 and 7, error rates improve dramatically as more data are added to the training sets. When training on roughly 1% of the full space, average error and standard deviation drop to about 0.7-6.7%. Randomly sampling 1% more brings error rates down to 0.6-4.7%. Rates reach an asymptote at sample sizes of about 4%: models for these applications exhibit less than 2% average error.

Learning curves for the processor parameter study follow similar trends. When simulating only 0.24% of the full design space (50 training examples), the data contain too little information to train accurate models. Depending on the application, average error rate varies between 2.5-6.4%, while standard deviation falls in the 1.8-5.2% range. As more data are sampled, model accuracy improves rapidly. When training set size reaches 1% of the full space, models for the applications in Figure 7 reach average error rates up to 2.9% with standard deviations less than 2.2%. Table VII shows that models for all applications maintain average error rates of 1.3-4.9% with standard deviations of 1.0-4.8% at this sampling rate. When training set size increases to 2% of the full space, models for all applications achieve error rates less than 2%.

Figures 8 and 9 show empirical cumulative distribution function (CDF) plots for mcf, twolf, art and mesa from the memory system and processor studies. We choose mcf, twolf, and art because they show the largest ranges in IPCs of all applications in both studies.

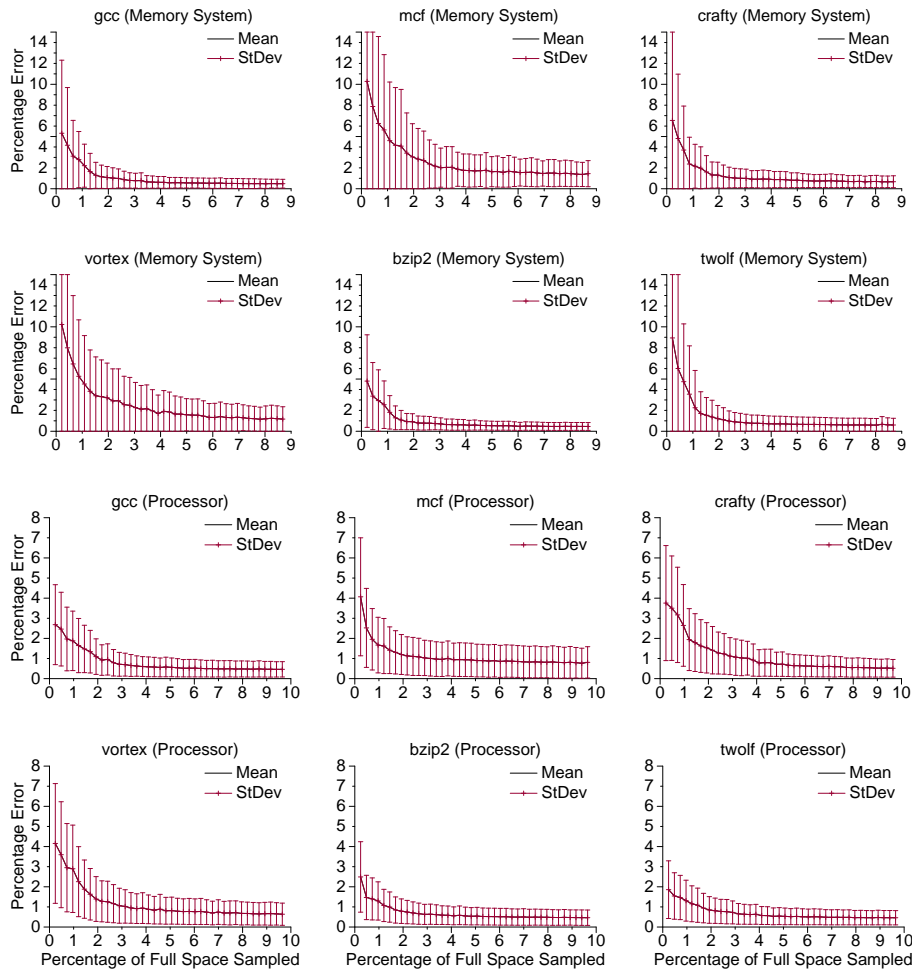


Fig. 6. Accuracy of the models on the design spaces for integer applications (random sampling). Training sets vary from 50 to 2000 points in increments of 50.

We choose mesa because its behavior is representative of many remaining applications. The x axes show percentage error, and the y axes show the percentage of data points that achieve error less than each x value. Columns show plots for training set sizes of about 1%, 2%, and 4% of the design spaces.

To evaluate the robustness of our error estimation, we compare estimated and true mean error and standard deviation. Figures 10 and 11 illustrate these metrics on the memory and processor design spaces as a function of training set size. For almost all codes, cross validation estimates are within 0.5% of actual values for sample sizes greater than or equal to 1%. For sample size less than 1%, differences between estimated and actual errors vary between 0.5-4%, but estimates are conservative in this regime. Cross validation estimates error rates from individual ANNs in the ensemble. Final predictions, however, average predictions of all ANNs, typically yielding lower error rates. Cross validation thus slightly

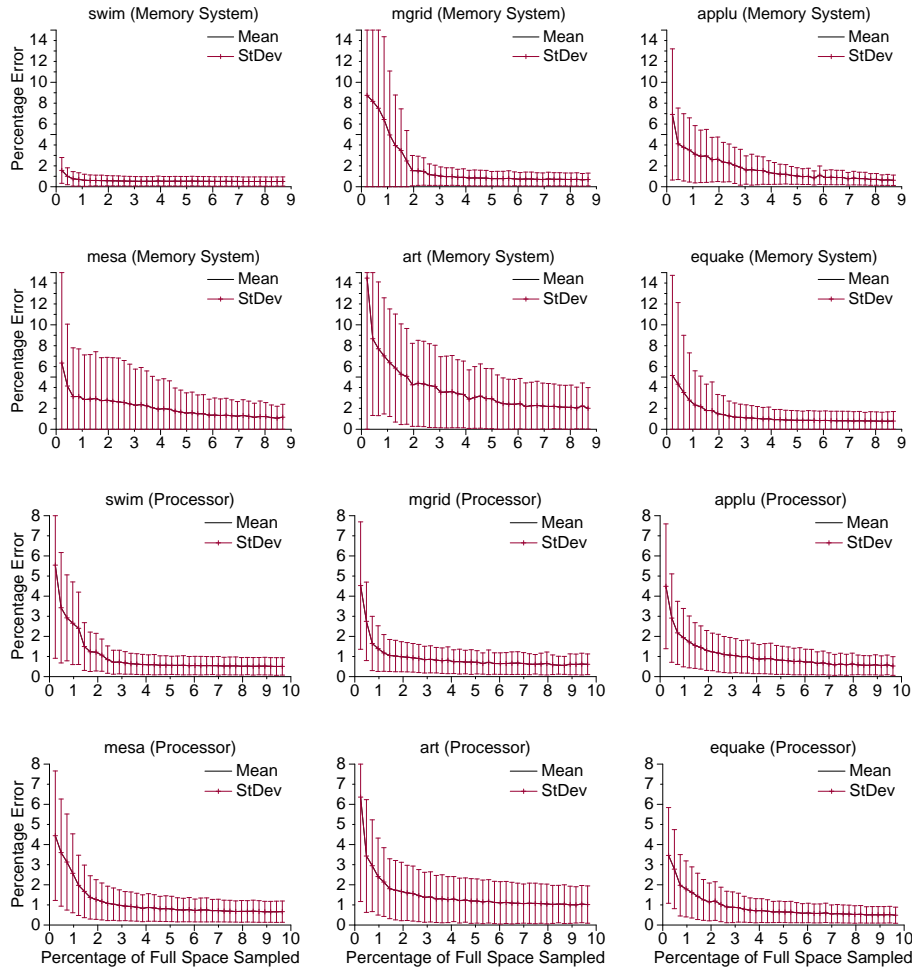


Fig. 7. Accuracy of the models on the design spaces for floating point applications (random sampling). Training sets vary from 50 to 2000 points in increments of 50.

overestimates actual error, providing a conservative estimate of average prediction accuracy and standard deviation. With sample size above 1%, differences between true and estimated error rates become negligible. Accuracy of these estimates allows the architect to stop collecting simulation results as soon as error rates become acceptable. In our experiments, cross validation almost never underestimates error.

The top of Table IX shows the best three configurations for `bzip2` from the memory hierarchy study. The bottom shows the best predicted configurations for a 2% sampling of the design space. The predicted configurations point to the same narrow area of the design space as the simulation results: L1 cache size, write policy, and L2 size match, and L2 block size differs in only one configuration. For other parameters, the model chooses slightly different tradeoffs from the optimal settings found by simulation, but these do not yield significant performance differences for this application and this particular (optimum)

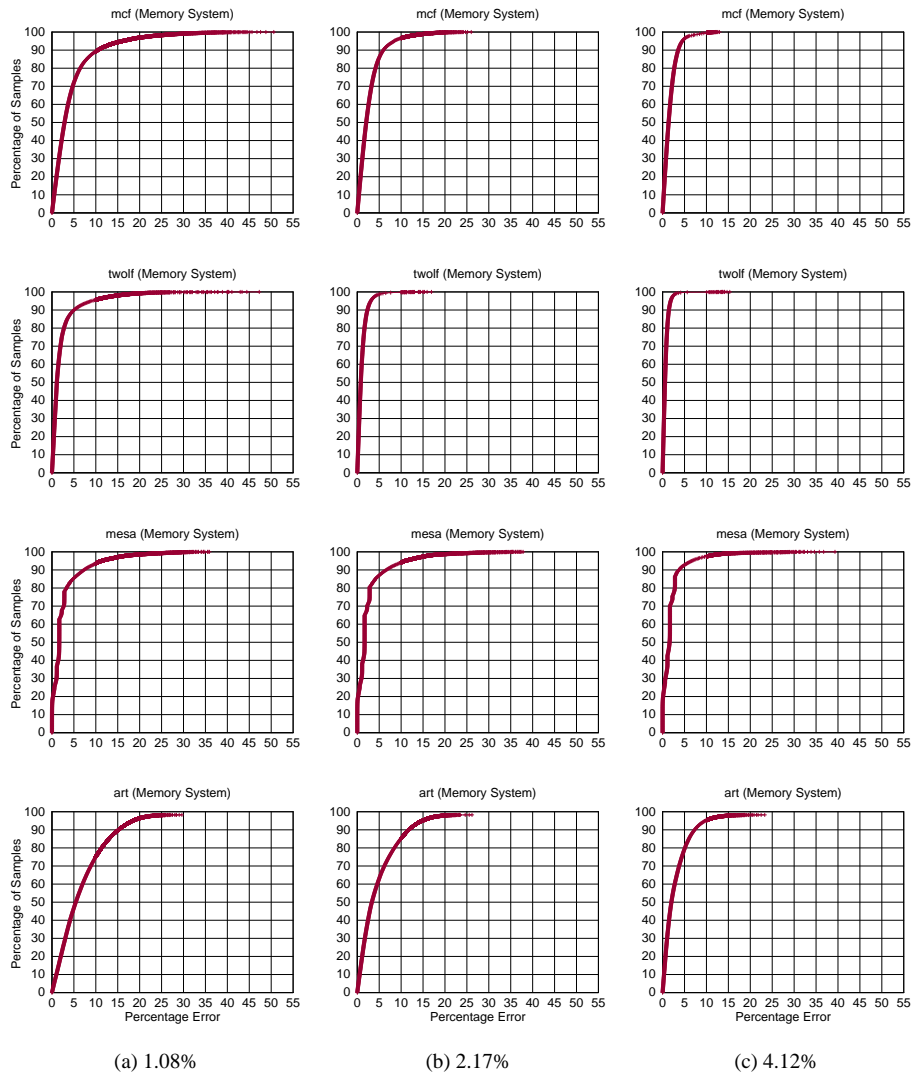


Fig. 8. Empirical CDF plots of error on Memory System study: x axes show percentage error, and the y axes show the percentage of data points that achieve error less than each x value.

region of the design space. The model comes close to finding the global optimum, and predicted performance characteristics of the region it finds are similar to the actual performance characteristics. Combining this with an approach that ranks parameter importance or characterizes interactions [Yi et al. 2003; Joseph et al. 2006b] guides the architect in exploring design tradeoffs in the region.

4.3 Accuracy for the CMP Study

For the CMP study, we use two SPEC OMP benchmarks (art-OMP and swim-OMP), one parallel NAS benchmark (mg), and three SPLASH-2 benchmarks (fft, ocean, radix). Fig-

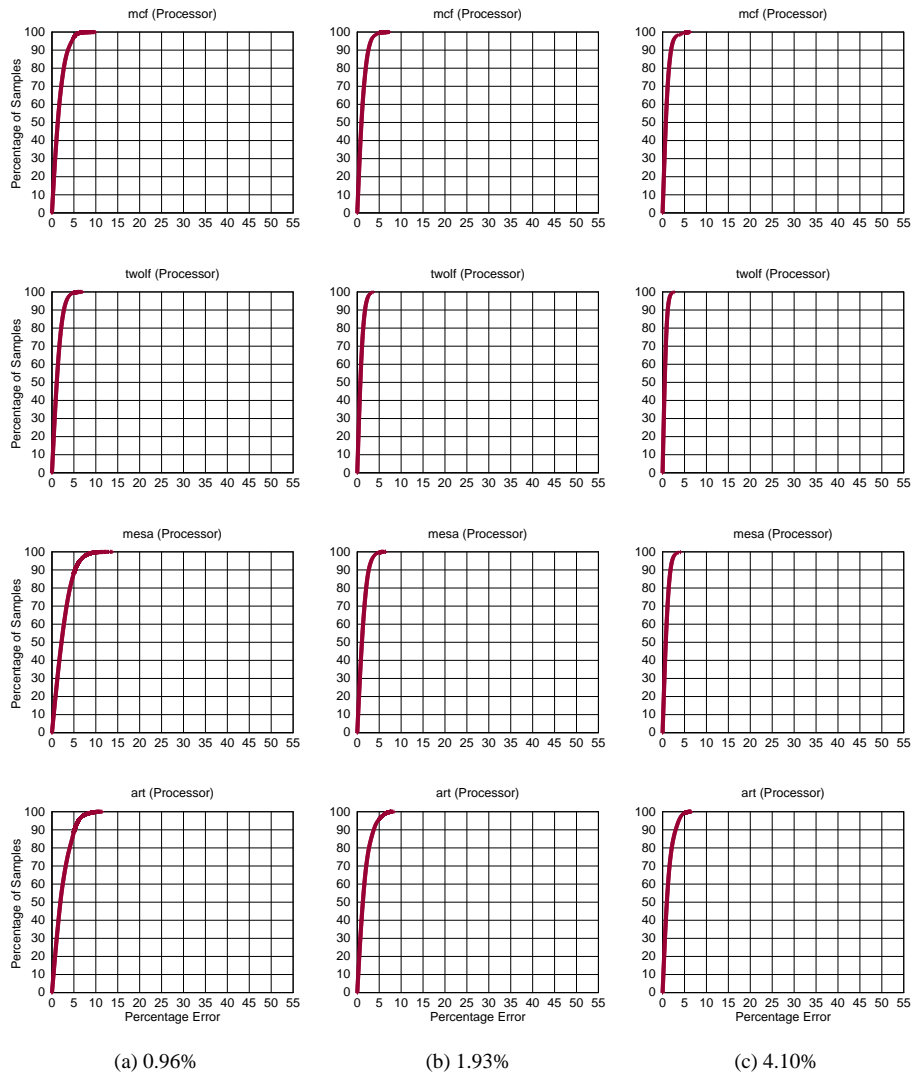


Fig. 9. Empirical CDF plots of error on Processor study: x axes show percentage error, and the y axes show the percentage of data points that achieve error less than each x value.

ure 12 shows learning curves for this study. Simulating 0.1% of the design space yields mean errors between 8.7% and 11.6%, with standard deviation between 8.1% and 18.5%. As expected, training on more samples improves model accuracy. At about 1% of the full space, the models reach mean error rates of 3.9-5.3%, with standard deviations of only 4.3-7.1%. All applications show similar learning curves. Table X summarizes results for all applications in the CMP study. CDFs for all studies at about 1% look similar, giving us confidence that adding more training samples will yield similar accuracies for this study.

Table X lists the mean error and the standard deviation of error obtained on the CMP study at a 1.03% sampling of the design space. Models for art, swim, mg, fft, ocean, and

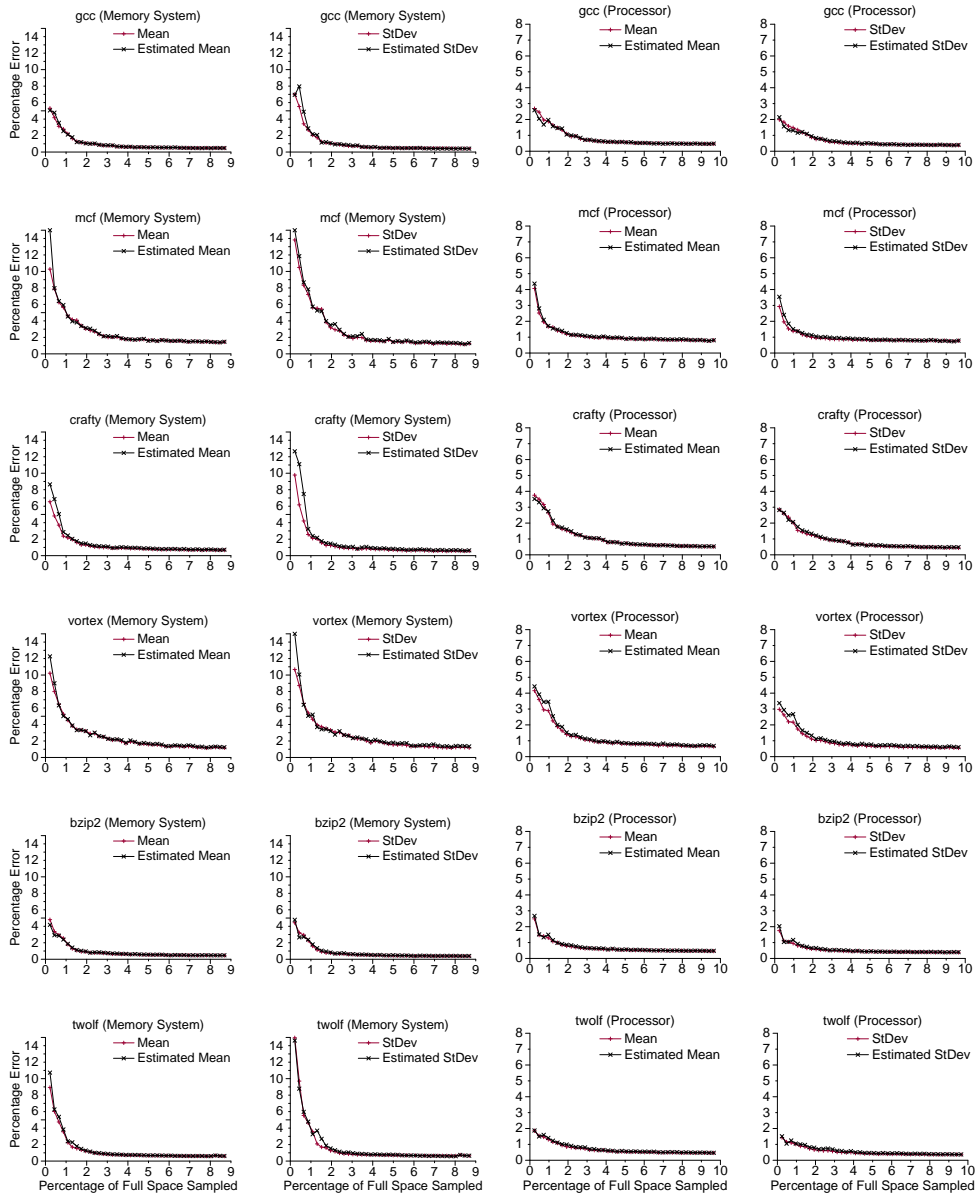


Fig. 10. Estimated and true means and standard deviations for percentage error for integer applications (random sampling). Training sets vary from 50 to 2000 points in increments of 50.

radix obtain average error rates of 4.66%, 4.44%, 5.27%, 4.79%, 3.89%, and 4.74%, and standard deviations of 5.03%, 4.87%, 6.41%, 5.25%, 4.53%, and 6.25%, respectively. Note that system performance varies widely across the CMP design space we study (by factors of 25, 41, 55, 37, 35, and 65 for art, swim, mg, fft, ocean, and radix, respectively), and hence these error rates are negligible compared to the performance impact of the parameters

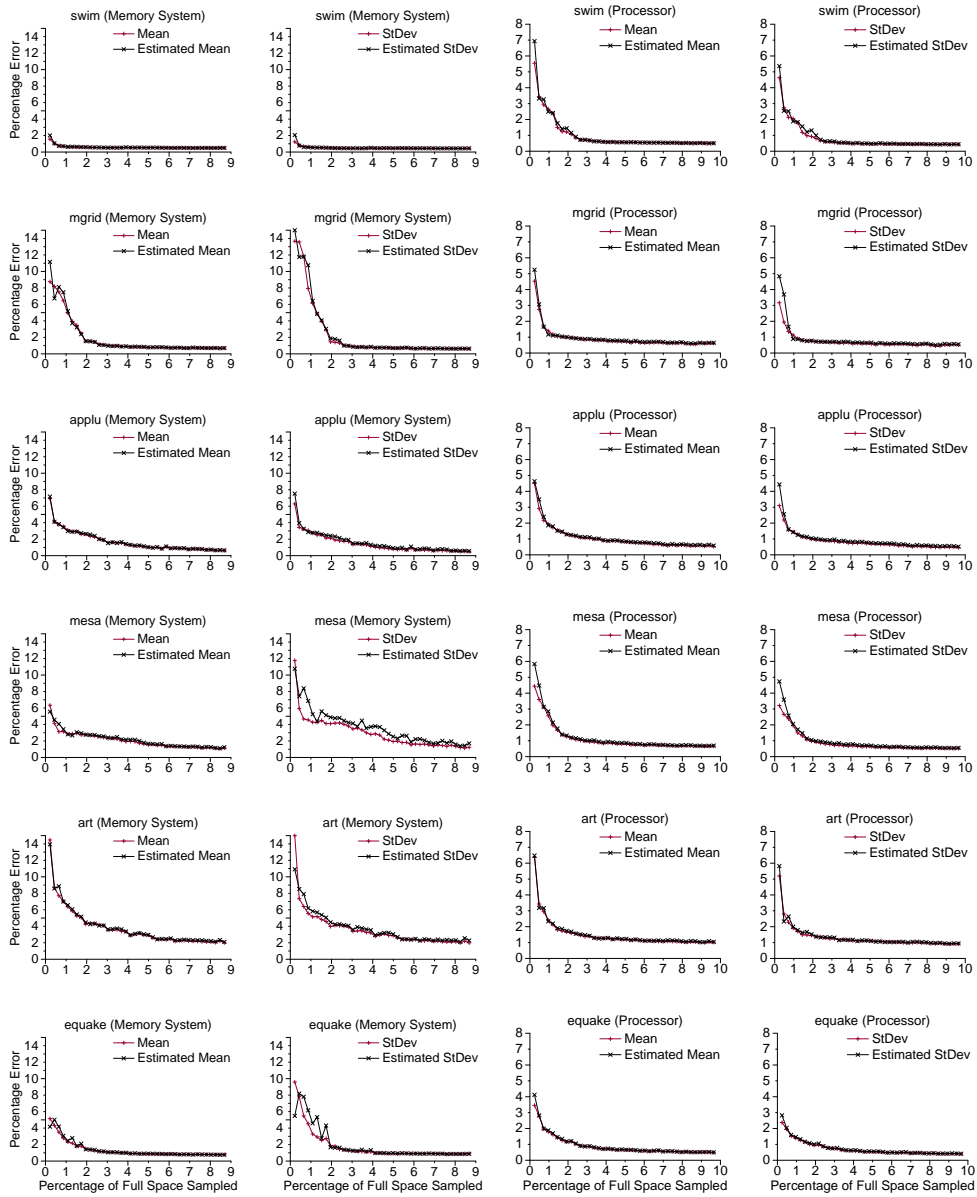


Fig. 11. Estimated and true means and standard deviations for percentage error for floating point applications (random sampling). Training sets vary from 50 to 2000 points in increments of 50.

we vary. The CDF plots in Figure 13 show the distribution of error. 90% of the points are predicted with less than 12.2-9.2% error, and 75% of the points are predicted with less than 5.5-6.5% error. 65-72% of the points are predicted with less than 5% error. These results suggest that ANNs can handle design spaces with widely varying target values and still deliver low percentage error when trained for the right metric (as explained in Section 2.3).

Table IX. Similarity in best configurations for bzip2

L1 size (KB)	L1 block size (B)	L1 ways	L1 write policy	L2 size (KB)	L2 block size (B)	L2 ways	L2 bus width	FSB frequency (GHz)	IPC
Best Simulation Configurations									
16	64	2	WB	1K	128	16	32	1.4	1.10
16	64	2	WB	1K	128	16	16	1.4	1.10
16	64	2	WB	1K	128	8	32	1.4	1.10
Best Predicted Configurations									
16	32	2	WB	1K	128	16	32	1.4	1.09
16	32	4	WB	1K	128	16	32	1.4	1.08
16	64	1	WB	1K	128	16	8	0.8	1.08

Table X. CMP study results (random sampling).

	CMP Study											
	0.31% Sample				0.52% Sample				1.03% Sample			
	Mean Error %		SD of Error %		Mean Error %		SD of Error %		Mean Error %		SD of Error %	
App.	True	Est.	True	Est.	True	Est.	True	Est.	True	Est.	True	Est.
art-OMP	6.13	6.84	6.99	5.68	5.25	6.37	5.94	5.86	4.66	5.03	6.05	5.50
swim-OMP	5.10	6.13	5.41	7.24	4.95	5.89	5.40	6.15	4.44	4.87	4.65	3.90
mg	6.98	6.49	9.06	6.09	5.72	6.29	6.81	5.91	5.27	6.41	4.76	4.39
fft	6.09	7.59	5.84	5.95	5.11	6.37	4.99	5.72	4.79	5.25	4.79	4.78
ocean	5.07	6.42	5.24	5.76	4.49	5.93	4.46	5.97	3.89	4.53	4.26	4.37
radix	6.23	10.08	8.16	9.06	5.69	7.15	7.51	6.43	4.74	6.25	7.14	7.08

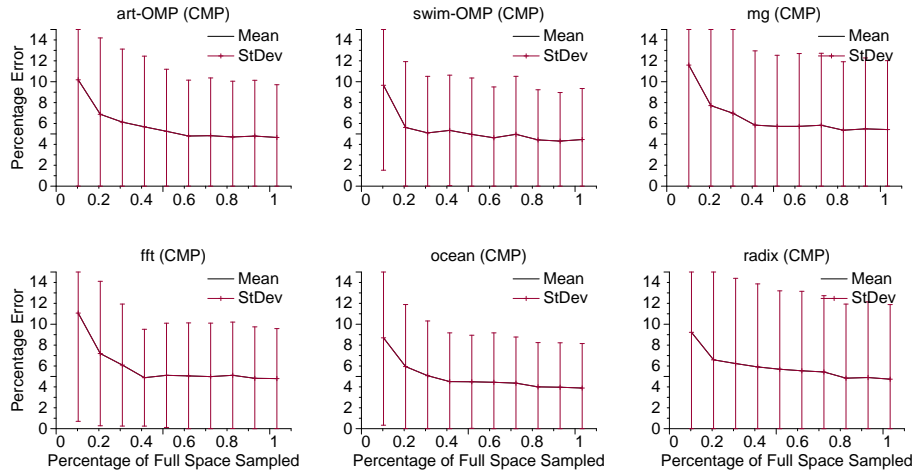


Fig. 12. Accuracy of the models on the CMP design space (random sampling). Training sets vary from 250 to 2500 points in increments of 250.

4.4 Training Times

If predictive models are to enable large design space exploration with reasonable time and computation requirements, it is critical that training and querying the models be *much* faster than architectural simulation time. ANNs respond to queries rapidly once built (less than 0.5ms per query in our experiments); we evaluate training times in this section.

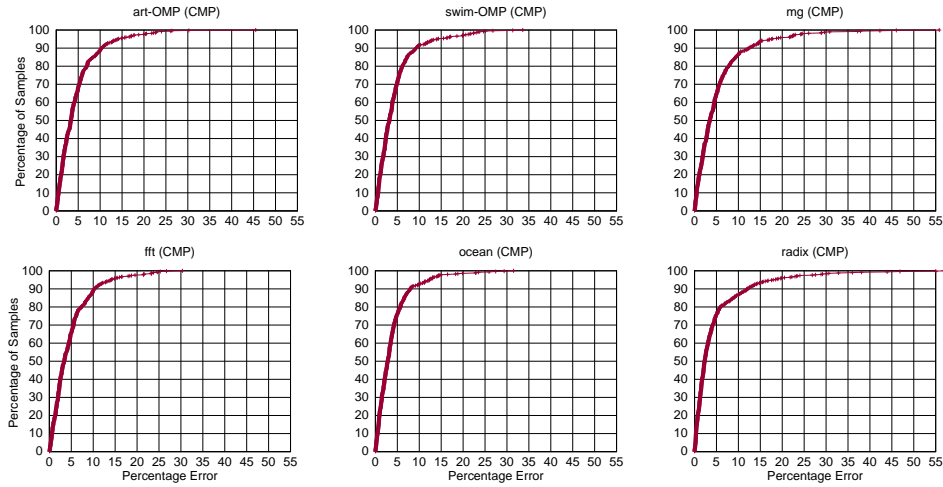


Fig. 13. Empirical CDF plots of error on CMP study: x axes show percentage error, and the y axes show the percentage of data points that achieve error less than each x value

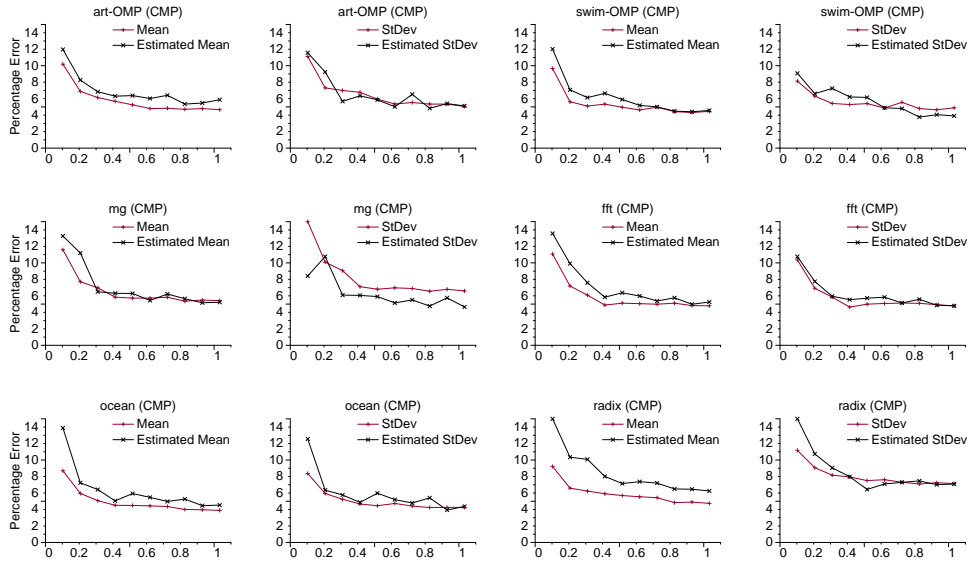


Fig. 14. Estimated and true means and standard deviations for percentage error for the CMP study (random sampling). Training sets vary from 250 to 2500 points in increments of 250.

Since the number training epochs is fixed at 30K in our experiments (Section 2.2), the only variable influencing the computational cost of training in our experiments is the size of the training sets. Figure 15 shows time required to train our models as a function of training set size. The ANNs in the 10-fold cross validation ensemble are trained in parallel on 10 cluster nodes with 3GHz Intel Pentium 4TM CPUs and 1GB DRAM. Each point

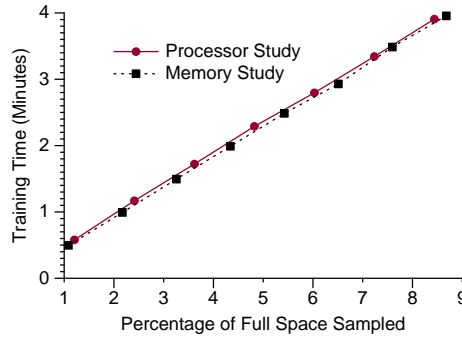


Fig. 15. Training times

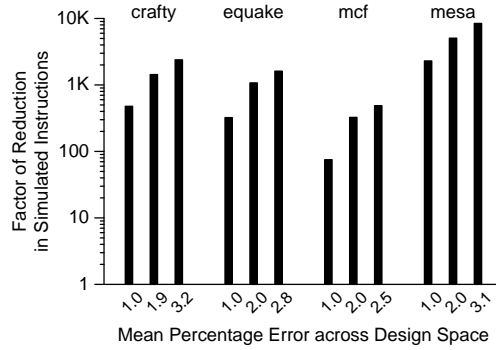


Fig. 16. Gains from combining ANNs+SimPoint

represents the average of three measurements. As training sets increase from 1-9% of the parameter space, training times scale linearly from 30 seconds to four minutes¹. Since simulation results are collected in batches, each round of training is amortized over multiple simulations. Learning curves presented in these studies typically level off at training set sizes of 2-4% of the full space, requiring less than two minutes per training step (compared to hours, days, or weeks of simulation per design point).

4.5 Integration with SimPoint

Our predictive modeling directly targets large parameter spaces and is orthogonal to techniques that reduce times for single simulations. This orthogonality does not necessarily imply that multiple techniques can be combined successfully. For instance, SimPoint reduces experiment time at the expense of some loss in accuracy. If the two approaches are combined to build a predictive model based on SimPoint samples, that model must handle input imprecision (which acts like noise), and not amplify error. Fortunately, ANNs work well in the presence of noise. To verify the robustness of our models, we repeat the processor study (with random sampling) using SimPoint. After deriving SimPoints and

¹This result is expected, since the algorithmic complexity of training a neural network with a single hidden layer, H hidden units, I inputs, and O outputs on D data points for P passes through the training set is $O(H(I + O)PD)$.

corresponding weights, we collect results for each application on every point in the space by calculating SimPoint performance estimates per run. We train our ANNs on these noisy datasets, but measure accuracy *with respect to the complete simulations*. Figure 16 shows gains from combining approaches, while Figure 17 and Figure 18 show details. Figure 16 shows reduction in simulated instructions at average percentage errors between 1-4%. The combined approach yields impressive reductions in number of simulated instructions for design space exploration. Even at error rates as low as 1%, the combined approach reduces number of simulated instructions by two-three orders of magnitude. If 3.2% error rates can be tolerated, reductions reach three-four orders of magnitude. Of these gains, ANN modeling contributes factors of 40-200, while SimPoint contributes factors of 10-60.

The top of Figure 17 shows learning curves. When simulating only 0.24% of the parameter space, average error rate and standard deviation vary between 3.5-4.4% and 2.4-3.2%, respectively. Error rates steadily decrease as simulation results are added to the training sets. When 1% of the full space is simulated, average error rate drops to less than 2.7% and standard deviation drops to 1.4-2.0%. At this sampling rate, the models are accurate and perform consistently well in all regions of the design space, as indicated by lower standard deviation. When training sets contain 2% of the full space, average error falls between 1.1-1.4%. In this regime, standard deviation varies between 0.9-1.2%. Compared to using full simulations, training models with SimPoint results gives slightly higher error, but differences are negligible. The bottom of Figure 17 plots estimated and average error and its standard deviation as a function of training set size. Estimates are again accurate. One difference between these and the original results is that estimates provided by cross validation beyond a 1% sampling are consistently lower than actual error (differences are small). When cross validation calculates error estimates, it does so with respect to SimPoint results, unaware of the noise in those results. Note, however, that these estimates are never off by more than 1% in this regime. Figure 18 shows that our models' ability to predict performance is not adversely affected by using SimPoint: in all cases, the SimPoint learning curves and the full simulation learning curves are within 1%, and for three of four, the SimPoint predictions are conservative. Our results indicate that ANNs handle the inherent inaccuracies induced by SimPoint well. Typically, average error rates of below 2% are maintained below a 1% sampling of the full design space, and a 1% error rate is obtained by sampling about 2% of the space.

5. RELATED WORK

Several recent articles elucidate aspects of the design space problem. Martonosi and Skadron [2001] summarize an NSF workshop's conclusions: trends towards multiple cores and increasing on-chip heterogeneity will lead us to build systems that are difficult to simulate; we require research into abstractions and evaluation methodologies that make quantitative evaluations of complex systems manageable; existing tools dictate the majority of research, resulting in light exploration of difficult-to-model parts of the design space; and the community's emphasis on simulation may cause practitioners to overlook other useful and possibly more informative modeling techniques. Jacob [2003] spends six months simulation time to study a small part of a memory system design space. Davis et al. [2005] struggle with massive design spaces for in-order multithreaded CMP configurations: they employ industry guidelines to prune the space, but find the remaining find 13K design points for in-order CPUs alone unmanageable.

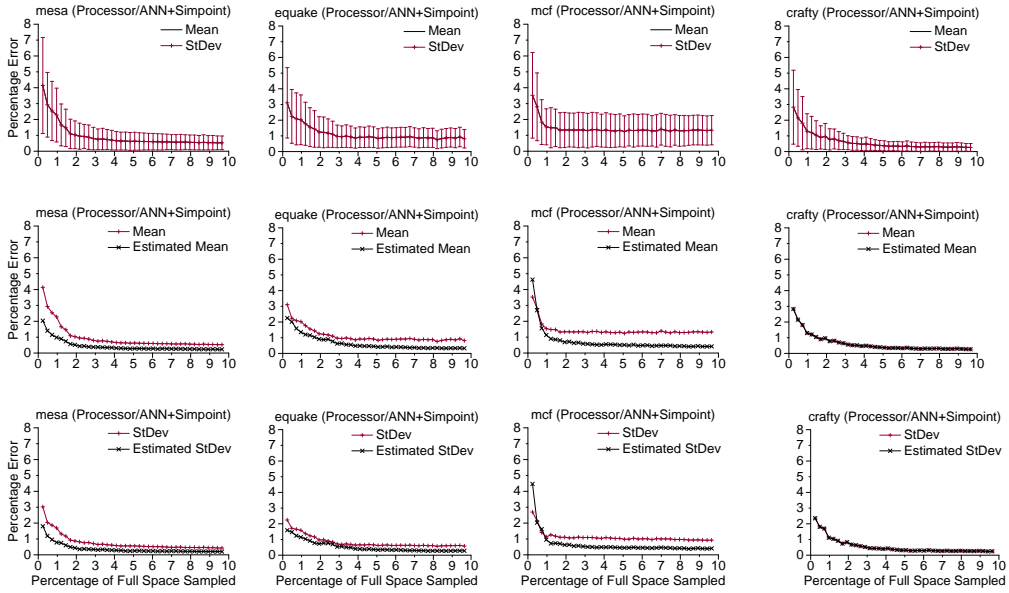


Fig. 17. SimPoint processor study results (random sampling). Training sets vary from 50 to 2000 points in increments of 50.

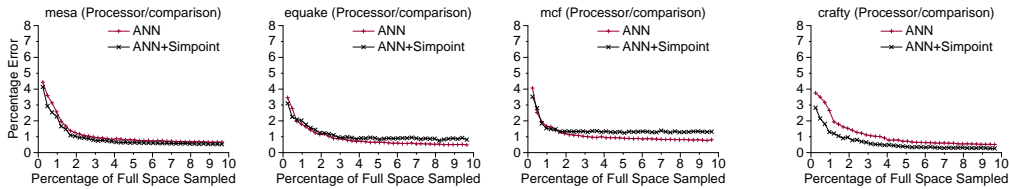


Fig. 18. Comparison of learning curves for full simulation runs vs. using SimPoints.

5.1 Exploration of Novel Design Spaces

Xie et al. [2006] explore 3D design spaces, giving an introduction to 3D integration technology, discussing the design tools to enable adoption of 3D integrated circuits, presenting the implementation of various microprocessor components using 3D technology, and ultimately rendering the design of Pentium 4TM processor in this technology. The authors focus more on tools (an adaptation of CACTI, a floorplanning algorithm, and a thermal estimator) and implementation, rather than thorough design space exploration. Nonetheless, they concisely describe many challenges posed by designs using 3D integrated circuits.

Li et al. [2006] explore the inter-related variables of core count, pipeline depth, superscalar width, L2 cache size, and operating voltage and frequency for CMPs under various area and thermal constraints. Like Kumar et al. [2005], their results emphasize the need to optimize parameters in tandem (rather than one at a time). They use a modeling methodology based on single-core traces and cache simulation (to model core interactions) augmented with analytic power and thermal models in order to explore as many parameters as possible. Their intent is to explore performance under various power, thermal, area, and pipeline depth constraints, which reduces the size of their design space.

Kunkel et al. [2000] also emphasize the importance of modeling the entire system when evaluating server performance, finding that interactions of subsystems for a given workload creates performance effects that can be subtle to analyze and that make the system design difficult to optimize. They employ both high-level and detailed models (as in Kumar and Davidson [1980]) that include queueing networks and a trace-based timing simulation model that is tunable with respect to the level of detail modeled. One option, for instance, is to speed experiments by modeling events probabilistically instead of deterministically. They resort to trace-based simulation due to the exorbitant times required to perform execution-based simulations for commercial workloads on server-class systems.

5.2 Analytic and Statistical Models

Noonburg and Shen [1994] use probability vectors to compose a set of components as linked Markov chain models solved iteratively, delivering 90-98% accuracy on a set of kernels. Karkhanis and Smith [2004] construct an intuitive first-order analytic model of superscalar microprocessors that estimates performance with 87-95% accuracy compared to detailed simulation. Fields et al. [2004] define *interaction costs* (icosts) to identify events affecting another event of interest. The technique requires hardware support, and the authors propose efficient hardware to enable sampling execution in sufficient detail to construct statistically representative microarchitecture graphs for computing icosts.

Yi et al. [2003] demonstrate Plackett and Burman fractional factorial design in prioritizing parameters for sensitivity studies. They model a high and low value for a set of N parameters, varying each independently to explore extremes of the design space in $2N$ simulations. By focusing on the most important parameters, PB analysis can reduce the simulations required to explore a large design space. Chow and Ding [1997] and Cai et al. [1998] apply principal components analysis and multivariate analysis to identify the most important parameters and their correlations for processor design. Eeckhout et al. [2003] and Phansalkar et al. [2005] use principal components analysis for workload composition and benchmark suite subsetting.

Muttreja et al. [2004; 2005] perform macromodeling, pre-characterizing reusable software components to construct high-level models to estimate performance and energy consumption. Symbolic regression filters irrelevant macromodel parameters, constructs macromodel functions, and derives optimal coefficient values to minimize fitting error. They apply their approach to simulation of several embedded benchmarks, yielding estimates with maximum 1.3% error. Joseph et al. [2006b] develop linear models primarily for identifying significant parameters and their interactions. The model is not intended to be predictive, and model prediction accuracy varies widely and depends on use of appropriate input transformations.

Lee and Brooks [2006] propose regression on cubic splines (piecewise polynomials) for predicting performance and power for applications executing on microprocessor configurations in a large microarchitectural design space. They address even larger design spaces than we do in our CMP study, and require lower sampling rates. Their approach is not automated, and requires some statistical intuition on the part of the modeler. Joseph et al. [2006a] develop a predictive performance model for superscalar processors. They require four times fewer samples to train their models, which are based on radial basis function (RBF) networks built from regression trees. Nonetheless, their example design space is smaller than the spaces examined in our work or in that of Lee and Brooks [2006], and it only includes numerical parameters. Furthermore, they test the accuracy of their predic-

tions against only 50 independent design points, whereas we test our model predictions for all design points in our sensitivity study design spaces, and we validate against 500 independent points in the exploratory CMP study. Lee and Brooks [2006] validate their predictions against 100 independent points. Finally, the RBF network approach is not automated.

Yoo et al. [2006] use ANNs to characterize multi-tiered web-service workloads, leveraging their predictions to guide application performance tuning. Their model achieves an average prediction accuracy (for multiple metrics inherent to the applications, e.g., effective transactions per second) of 95% on real commercial workloads. Furthermore, they clearly illustrate the highly nonlinear nature of the parameter space they address. İpek et al. [2005] also use ANN models to predict runtimes of parallel scientific workloads with very large parameter spaces, achieving average prediction accuracy of 93-95%.

5.3 Reduced Simulation Workloads

Statistical simulation [Eeckhout et al. 2003] represents an attractive alternative to full simulation for many purposes. The technique first derives application characteristics; generates a much smaller, synthetic trace exhibiting those characteristics; and then simulates that trace. Oskin et al. [2000] develop a hybrid simulator (HLS) that uses statistical profiles to model application instruction and data streams. HLS dynamically generates a code base and symbolically executes it on a superscalar microprocessor core much faster than detailed, observing average error within 5-7% of cycle-by-cycle simulation of a MIPS R10000. Iyengar et al. [1996] introduce the *R-metric* to evaluate *representativeness* of sampled, reduced traces (with respect to actual application workloads) and develop a graph-based heuristic to generate better synthetic traces. Eeckhout et al. [2004] build on this to generate statistical control flow graphs characterizing program execution, attaining 1.8% average error on 10 SPEC 2000 benchmarks.

5.4 Partial Simulation Techniques

Wunderlich et al. [2003] model minimal instruction stream subsets in SMARTS to achieve results within desired confidence intervals. The approach can deliver high accuracies, even with small sampling intervals. For large intervals, as in SimPoint [Sherwood et al. 2002], state warmup becomes less significant, but can still improve accuracy.

Conte et al. [1996] and Haskins and Skadron [2001] sample portions of application execution, performing *warmup* functional simulation to create correct cache and branch predictor state for portions of the application being simulated in detail. Haskins and Skadron exploit *Memory Reference Reuse Latencies* (MRRLs) to choose the number of warmup instructions to simulate functionally before a desired simulation point [Haskins and Skadron 2003]. This selection of warm-up periods roughly halves simulation time with minimal effect on IPC accuracy. Eeckhout et al. [2005] further reduce warmup periods with *Boundary Line Reuse Latencies* (BLRLs), in which they consider only reuse latencies that cross the boundary between warmup and sample. Van Biesbrouck et al. [2005] investigate warmup for both SimPoint and SMARTS, storing reduced *Touched Memory Images* (TMIs) and *Load Value Sequences* (LVSs) of data to be accessed in a simulation interval in conjunction with *Memory Hierarchy State* (MHS) collected through cache simulation of the target benchmark. They find MHS+LVS to be as accurate as MRRLs with faster simulation, and to require less storage than TurboSMARTS [Wenisch et al. 2005] checkpoints.

Rapaka and Marculescu [2003] use a hybrid simulation engine to detect code hotspots of

high temporal locality and use information from these to estimate statistics for the remaining code. This approach needs no application behavior characterization before simulation (as in Muttreja et al. [2004; 2005] and Van Biesbrouck et al. [2005]). Their adaptive profiling strategy predicts application performance and power with less than 2% error while speeding simulations by up to a factor of 12. This approach is not tied to a given architecture or application/input pairs, but requires modifying the simulation model.

6. CONCLUSIONS

We demonstrate that Artificial Neural Networks can model large design spaces with high accuracy and speed (training the ANNs on 1-2% of the space lets us predict results for other design points with 98-99% accuracy). Our approach is potentially of great value to computer architects, who rely on design space exploration to evaluate the sensitivity of a proposal to many interacting architectural parameters. We present a fully automated, general mechanism to build accurate models of architectural design spaces from limited simulation results. The approach is orthogonal to statistical techniques that reduce single simulation times or assess parameter importance, and we show that combining our models with one already widely used technique reduces number of simulated instructions by three-four orders of magnitude. We make several contributions:

- a general mechanism to build highly accurate, confident models of architectural design spaces, allowing architects to cull uninteresting design points quickly and focus on the most promising regions of the design space;
- a framework that incorporates additional simulation results incrementally and allows models to be queried to predict performance impacts of architectural changes, enabling efficient discovery of tradeoffs among parameters in different regions;
- an evaluation showing that training times are negligible compared to even individual architectural simulations; and
- an analysis of results showing that our approach can reduce simulation times for sensitivity studies by several orders of magnitude with almost no loss in accuracy.

We predict performance here, but our approach is sufficiently general to predict other statistics, even several at once. Our mechanism enables much faster exploration of design spaces of currently feasible sizes, and makes possible exploration of massive spaces outside the reach of current simulation infrastructures. We thus provide the architect with another tool to assist in the design and evaluation of systems. In so doing, we hope to increase understanding of design tradeoffs in a world of ever increasing system complexity.

ACKNOWLEDGMENTS

This paper extends “Efficiently Exploring Architectural Design Spaces via Predictive Modeling”, which appeared in the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII) in October, 2006 [İpek et al. 2006]. Part of this work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory (LLNL) under contract W-7405-Eng-48 (UCRL-JRNL-227222) and under National Science Foundation (NSF) grants CCF-0444413, OCI-0325536, and CNS-0509404. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or LLNL. The U.S. Government

retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

The authors thank editor David August, as well as Dave Albonesi, Kai Li, José Martínez, José Moreira, and the anonymous reviewers for feedback on this work. A research equipment grant from Intel Corp. provided computing resources that helped enable our design space studies.

REFERENCES

- BIGUS, J. 1994a. Applying neural networks to computer system performance tuning. In *Proc. IEEE International Conference on Neural Networks*.
- BIGUS, J. 1994b. Computer system performance modeling using neural networks. In *Proc. International Neural Network Society Conference*. 510–515.
- BORKAR, S., DUBEY, P., KAHN, K., KUCK, D., MULDER, H., PAWLOWSKI, S., AND RATTNER, J. 2006. Platform 2015: Intel processor and platform evolution for the next decade. White Paper, Intel Corporation.
- CAI, G., CHOW, K., NAKANISHI, T., HALL, J., , AND BARANY, M. 1998. Multivariate power/performance analysis for high performance mobile microprocessor design. In *Power Driven Microarchitecture Workshop*.
- CARUANA, R., LAWRENCE, S., AND GILES, C. 2000. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Proc. Neural Information Processing Systems Conference*.
- CHOW, K. AND DING, J. 1997. Multivariate analysis of Pentium Pro processor. In *Intel Software Developers Conference*. 84–91.
- CONTE, T., HIRSCH, M., AND MENEZES, K. 1996. Reducing state loss for effective trace sampling of superscalar processors. In *Proc. IEEE International Conference on Computer Design*. 468–477.
- DAVIS, J., LAUDON, J., AND OLUKOTUN, K. 2005. Maximizing CMP throughput with mediocre cores. In *Proc. IEEE/ACM International Conference on Parallel Architectures and Compilation Techniques*. 51–62.
- ECKHOUT, L., BELL, JR., R., STOUGIE, B., DE BOSSCHERE, K., AND JOHN, L. 2004. Control flow modeling in statistical simulation for accurate and efficient processor design studies. In *Proc. 31st IEEE/ACM International Symposium on Computer Architecture*. 350–361.
- ECKHOUT, L., LUO, Y., JOHN, L., AND DE BOSSCHERE, K. 2005. BLRL: Accurate and efficient warmup for sampled processor simulation. *The Computer Journal* 48, 4, 451–459.
- ECKHOUT, L., NUSSBAUM, S., SMITH, J., AND DE BOSSCHERE, K. 2003. Statistical simulation: Adding efficiency to the computer designer’s toolbox. *IEEE Micro* 23, 5, 26–38.
- ECKHOUT, L., VANDIERENDONCK, H., AND DE BOSSCHERE, K. 2003. Quantifying the impact of input data sets on program behavior and its applications. *Journal of Instruction Level Parallelism* 5, <http://www.jilp.org/vol5>.
- EYERMAN, S., ECKHOUT, L., AND BOSSCHERE, K. D. 2005. The shape of the processor design space and its implications for early stage explorations. In *Proc. 7th WSEAS International Conference on Automatic Control, Modeling and Simulation*. 395–400.
- FIELDS, B., BODICK, R., HILL, M., AND NEWBURN, C. 2004. Interaction cost and shotgun profiling. *ACM Transactions on Architecture and Code Optimization* 1, 3, 272–304.
- HASKINS, J. AND SKADRON, K. 2001. Minimal subset evaluation: Rapid warm-up for simulated hardware state. In *Proc. IEEE International Conference on Computer Design*. 32–39.
- HASKINS, J. AND SKADRON, K. 2003. Memory reference reuse latency: Accelerated sampled microarchitecture simulation. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*. 195–203.
- HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Verlag.
- İPEK, E., DE SUPINSKI, B., SCHULZ, M., AND MCKEE, S. 2005. An approach to performance prediction for parallel applications. In *Proc. ACM/IEEE Euro-Par International European Conference on Parallel Computing*. 196–205.
- İPEK, E., MCKEE, S., DE SUPINSKI, B., SCHULZ, M., AND CARUANA, R. 2006. Efficiently exploring architectural design spaces via predictive modeling. In *Proc. 12th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*. 195–206.

- IYENGAR, V., TREVILLYAN, L., AND BOSE, P. 1996. Representative traces for processor models with infinite cache. In *Proc. 2nd IEEE Symposium on High Performance Computer Architecture*. 62–73.
- JACOB, B. 2003. A case for studying DRAM issues at the system level. *IEEE Micro* 23, 4, 44–56.
- JOSEPH, P., VASWANI, K., AND THAZHUTHAVEETIL, M. 2006a. A predictive performance model for superscalar processors. 161–170.
- JOSEPH, P., VASWANI, K., AND THAZHUTHAVEETIL, M. 2006b. Use of linear regression models for processor performance analysis. In *Proc. 12th IEEE Symposium on High Performance Computer Architecture*. 99–108.
- KARKHANIS, T. AND SMITH, J. 2004. A 1st-order superscalar processor model. In *Proc. 31st IEEE/ACM International Symposium on Computer Architecture*. 338–349.
- KLEINOSOWSKI, A. AND LILJA, D. 2002. MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research. *Computer Architecture Letters* 1.
- KONGETIRA, P., AINGARAN, K., AND OLUKOTUN, K. 2005. Niagara: A 32-way multithreaded SPARC processor. *IEEE Micro* 25, 2 (Mar.), 21–29.
- KUMAR, B. AND DAVIDSON, E. 1980. Computer system design using a hierarchical approach to performance evaluation. *Communications of the ACM* 23, 9 (Sept.), 511–521.
- KUMAR, R., ZYUBAN, V., AND TULLSEN, D. 2005. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *Proc. 32nd IEEE/ACM International Symposium on Computer Architecture*. 408–419.
- KUNKEL, S., EICKEMEYER, R., LIPASTI, M., MULLINS, T., O’KRAFKA, B., ROSENBERG, H., VANDERWIEL, S., VITALE, P., AND WHITLEY, L. 2000. A performance methodology for commercial servers. *IBM Journal of Research and Development* 44, 6, 851–872.
- LEE, B. AND BROOKS, D. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proc. 12th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*. 185–194.
- LI, Y., LEE, B., BROOKS, D., HU, Z., AND SKADRON, K. 2006. CMP design space exploration subject to physical constraints. In *Proc. 12th IEEE Symposium on High Performance Computer Architecture*. 15–26.
- MARTONOSI, M. AND SKADRON, K. 2001. NSF computer performance evaluation workshop: Summary and action items. http://www.princeton.edu/~mrm/nsf_sim_final.pdf.
- MARZBAN, C. 2000. A neural network for tornado diagnosis. *Neural Computing and Applications* 9, 2, 133–141.
- MITCHELL, T. 1997. *Machine Learning*. WCB/McGraw Hill, Boston, MA.
- MUTTREJA, A., RAGHUNATHAN, A., RAVI, S., AND JHA, N. 2004. Automated energy/performance macro-modeling of embedded software. In *Proc. 41st ACM/IEEE Design Automation Conference*. 99–102.
- MUTTREJA, A., RAGHUNATHAN, A., RAVI, S., AND JHA, N. 2005. Hybrid simulation for embedded software energy estimation. In *Proc. 42nd ACM/IEEE Design Automation Conference*. 23–26.
- NOONBURG, D. AND SHEN, J. 1994. Theoretical modeling of superscalar processor performance. In *Proc. IEEE/ACM 27th International Symposium on Microarchitecture*. 53–62.
- OSKIN, M., CHONG, F., AND FARRENS, M. 2000. HLS: Combining statistical and symbolic simulation to guide microprocessor design. In *Proc. 27th IEEE/ACM International Symposium on Computer Architecture*. 71–82.
- PHANSALKAR, A., JOSHI, A., EECKHOUT, L., AND JOHN, L. 2005. Measuring program similarity: Experiments with SPEC CPU benchmark suites. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*. 10–20.
- POMERLEAU, D. 1993. Knowledge-based training of artificial neural networks for autonomous robot driving. In *Robot Learning*, J. Connell and S. Mahadevan, Eds. Kluwer Academic Press, Boston, 19–43.
- RAMANATHAN, R. 2006. Intel multi-core processors: Making the move to quad-core and beyond. White Paper, Intel Corporation.
- RAPAKA, V. AND MARCULESCU, D. 2003. Pre-characterization free, efficient power/performance analysis of embedded and general purpose software applications. In *Proc. ACM/IEEE Design, Automation and Test in Europe Conference and Exposition*. 10504–10509.
- RENAU, J. 2002. SESC. <http://sesc.sourceforge.net/index.html>.
- SAAR-TSECHANSKY, M. AND PROVOST, F. 2001. Active learning for class probability estimation and ranking. In *Proc. 17th International Joint Conference on Artificial Intelligence*. 911–920.

- SHERWOOD, T., PERELMAN, E., HAMERLY, G., AND CALDER, B. 2002. Automatically characterizing large scale program behavior. In *Proc. 10th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*. 45–57.
- TESAURO, G. 1995. Temporal difference learning and TD-Gammon. *Communications of the ACM* 38, 3 (Mar.), 58–68.
- VAN BIESBROUCK, M., EECKHOUT, L., AND CALDER, B. 2005. Efficient sampling startup for sampled processor simulation. In *Proc. 1st International Conference on High Performance Embedded Architectures and Compilers*. 47–67.
- WENISCH, T., WUNDERLICH, R., FALSAFI, B., AND HOE, J. 2005. TurboSMARTS: Accurate microarchitecture simulation sampling in minutes. *SIGMETRICS Performance Evaluation Review* 33, 1, 408–409.
- WILTON, S. AND JOUPPI, N. 1996. CACTI: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits* 31, 5 (May), 677–688.
- WUNDERLICH, R., WENISH, T., FALSAFI, B., AND HOE, J. 2003. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proc. 30th IEEE/ACM International Symposium on Computer Architecture*. 84–95.
- XIE, Y., LOH, G., BLACK, B., AND BERNSTEIN, K. 2006. Design space exploration for 3D architectures. *ACM Journal on Emerging Technologies in Computing Systems* 2, 2, 65–103.
- YI, J., LILJA, D., AND HAWKINS, D. 2003. A statistically-rigorous approach for improving simulation methodology. In *Proc. 9th IEEE Symposium on High Performance Computer Architecture*. 281–291.
- YOO, R., LEE, H., CHOW, K., AND LEE, H. 2006. Constructing a non-linear model with neural networks for workload characterization. In *Proc. IEEE International Symposium on Workload Characterization*. 150–159.
- YU, S., WINSLETT, M., LEE, J., AND MA, X. 2002. Automatic and portable performance modeling for parallel I/O: A machine-learning approach. *ACM SIGMETRICS Performance Evaluation Review* 30, 3 (Dec.), 3–5.