Genetically Programmed Response Surfaces for Efficient Design Space Exploration

Henry Cook, Kevin Skadron
Department of Computer Science
University of Virginia
Tech Report CS-2007-12
hcook@eecs.berkeley.edu skadron@cs.virginia.edu

August 2007

Abstract

In spite of many efforts to speed up cycle-accurate architecture simulation, exponential increases in architectural design complexity threaten to make traditional design optimization techniques completely intractable. *Response surface* methodologies address this challenge by transforming the optimization process from a lengthy series of detailed simulations into the tractable formulation and rapid evaluation of a marginally less accurate but easy to evaluate analytical expression—a predictive model. We propose *genetic programming* as a powerful method for creating these predictive response surface models out of sampled architectural performance data.

Genetically programmed response surfaces (GPRSs) allow the architect to make rapid design optimizations (because only a small number of detailed simulations are needed) while simultaneously obtaining insight into the problem domain (because the resulting response surface — a non–linear polynomial in our case — exposes relationships and relative weights among the design variables). We validate our methodology on realistic datasets and compare it to recently proposed techniques for predictive design space exploration. GPRSs are highly accurate when making global predictions about architectural performance behavior based on only small samples of performance data: global predictions of IPC incur less than 3% mean percentage error based on sample sizes of less than 1% of one target processor design space, and no worse than than mean 6% error at sample sizes as small as 0.0000002% out of over one billion possible design points from a second target space. GPRSs can therefore reduce required simulation costs by up to six orders of magnitude.

1 Introduction

Architects can no longer afford to rely solely on simulation to evaluate a design space. Design spaces are simply too large, and there are too many interactions among parameters [8, 12, 20]. Isolating a small number of variables to sweep neglects the fact that the values of many other parameters may change in order to keep a balanced organization. As a simple example, Skadron et al. [26] showed that better branch prediction changes the optimal values for ROB, LSQ, and cache size. The design space of a single core is already large; the multicore design space is nearly intractable, because core architecture (depth/width/structure sizes/etc.), core count, cache sizes, interconnect, energy efficiency, and thermal efficiency are *all* inter-related [20]. Even if all variables are swept, at a cost of O(n variables * m steps per variable) simulations, optimizing one variable at a time will stick in local minima and can produce dramatically sub-optimal designs. Analytical models can be developed that are trivial to solve, but a model simple enough to derive *a priori* typically cannot account for interactions, especially those involving multiple variables.

Techniques to speed up simulation time are not sufficient either, especially for multicore simulations, and especially for parallel programs, where an accurate simulation result for a single design point may still take a non-trivial time in order to properly account for thread interactions. Even if per simulation time is

dramatically reduced, exhaustive search of all possible parameter bindings may still be necessary, at a cost of O (n choose m). *What is required is an automatically generated analytical model that accounts for the necessary complexities but allows a large number of design points to be quickly evaluated.*

This is not a new problem, nor one unique to computer architecture optimization tasks. In an effort to address computational constraints on design space exploration, the modeling and simulation research community has developed the family of techniques known as *response surface methodology* [4, 31]. Given a sample of collected performance data, these techniques build approximation functions ("response surfaces") which accurately predict the performance of all the unsampled candidate designs. These functions serve to transform the optimization process from a lengthy series of detailed simulations into the tractable formulation and rapid evaluation of a marginally less accurate predictive model.

Techniques for creating globally accurate response surfaces range from simplistic linear regressions through highly non-linear techniques like trained artificial neural networks. We advocate the use of *genetic programming* [17] to create appropriate *non-linear, polynomial* approximation functions and fit them to the collected architectural performance data. Genetic programming is a technique, based on evolutionary biology, used to optimize a population of computer programs according to their ability to perform a computational task. In our case, the 'program' is an analytic equation whose evaluation embodies the response surface model, and the 'task' is to match the sample data points obtained from full–scale simulations [1].

The genetic programming (GP) algorithm operates on a population of expression trees, each of which represents a candidate response surface — specifically, a polynomial function which maps a subset of the design parameters to a performance measure. Individual functions deemed to provide the most accurate predictions are recombined using a set of evolutionary operators to form new generations of increasingly fit functions. Unlike a heuristic genetic search through the design space, GP uses the evolutionary process to improve the accuracy of a set of approximation functions, rather than to attempt to heuristically improve the performance of a set of designs. Creating an accurate approximation function and using it to predict which designs are optimal becomes increasingly efficient as design spaces continue to grow.

Genetic programming has been previously shown to create globally accurate response surfaces that successfully capture non-linear, highly multi-dimensional design spaces [1]. Because the predictive response surface is created based on a set of input data (e.g simulated performance or power), the quality of its output can only be as good as the simulator on which the data was collected. However, as the GPRSs are highly accurate, this approach is able to derive functions which can be trivially evaluated to rapidly estimate the result a long and costly cycle-accurate simulation would have produced. In fact, the time saved by using a GPRS means that the few detailed simulations that are needed for training can afford to be even more detailed, thus improving the overall efficiency and realism of the process. A final important property of GPRSs is that they produce fairly intuitive functions that clearly expose the relationship and relative importance of various configuration parameters.

Specifically, the main contributions of this paper are:

1. We prove that the GPRS technique is highly accurate when making global predictions about architectural performance behavior based on only small samples of performance data.

2. We demonstrate the usefulness of GP-provided explicit approximation functions in identifying important design parameters, uncovering variable relationships, and locating optimal design subsets.

3. We provide a direct comparison with recently published techniques and offer suggestions for to how these techniques could be integrated for maximum accuracy and efficiency.

On a superscalar processor design space consisting of over 20K design configurations, the genetic programming algorithm creates response surfaces which allow for global predictions of IPC with less than 3% mean percentage error and less than 9% worst case error, based on sample sizes of less than 1% of the total design space. Sample sizes as low as 0.1% of the global space still result in functions which incur less than 5% mean error, and less than 21% worst case error. Of particular importance, *the approximation equation provided by the GP process reveals subsets of the design space which in all but one of our tests cases contain 100% of the exhaustively determined global performance optima.* On a second, larger superscalar processor design space containing over one billion design points, we find that the median percentage error of the predictions range from 1.1% to 6.1% at sample sizes as low as 200 design points.

High accuracy at low sample sizes is not the only benefit conferred on an architect making use of GRPS. GPRS almost completely decouples the amount of time it takes to optimize a design from the amount of time

required to run a cycle–accurate simulation. Combined with a robust sampling method, GPRS collapses exponential design space growth caused by an increase in dimensionality back down to a linear increase in the number of fully simulated points in the sample set. Variables which have a significant impact on the target performance metric are explicitly identified. In addition, GPRSs allow us to predict the performance impact of a design change without actually simulating the adapted design. Once GPRS has explicitly defined the relationships among the design variables, we can evaluate the predicted change analytically. In sum, GPRS combines the automated ease of training neural networks [14] with the accuracy and insight provided by complex regression models [19].

The rest of this paper is organized as follows: Section 2 explains recent related work, Section 3 describes the GP methodology in greater depth, Section 4 details our experimental methodology, Section 5 presents a illustrative case study in using GPRS, Section 6 shows how the technique performs on realistic data sets, Section 7 enumerates significant conclusions and outlines plans for future work.

2 Related Work

Response surface methodology is well established and has previously been used in many engineering fields to address design optimization problems where the functional computation costs of evaluating design fitness are high [23, 31]. GPRSs specifically have been used for optimizing the calcination of roman cement [29], and predicting stress fractures in steel [30]. Genetic programming in general has been previously used in the computer architecture domain in the context of automatically synthesizing accurate branch and jump predictors [11]. Alvarez [1] found that that the genetic programming process detects which design variables have a significant performance impact during the approximation model construction, and can sometimes extrapolate beyond the initially defined range of the variables. His results also indicate that the technique is robust in the face of noise in the sample performance data [28].

As Ipek et al. [14] have observed, the computer architecture community has tended to exclusively use with simulation-based methods and techniques to reduce the length of design space searches, such as sampling methods (e.g., [5, 13, 24]). Despite many calls for reduced reliance on simulation (e.g., [27]), only recently has the architecture community begun to more aggressively investigate analytical methods.

2.1 Reducing Simulation Inputs

Phansalkar et al. [22] and Eeckhout et al. [10] analyze the SPEC2k [7] benchmark suite and use Principle Components Analysis to select a representative subset of the most unique sections and applications in an attempt to reduce the total number of simulations required to characterize overall suite performance. Klein et al. offer MinneSPEC, a reduced input set for the SPEC2k CPU benchmark suite, as a workload suitable for rapid exploration of the design parameter space [16]. SimPoint [25] and SMARTS [32] also serve to automatically reduce instruction stream sizes and thereby decrease the time required for simulation analysis. Statistical simulation [9] generates small, synthetic traces based on observed application characteristics, allowing the actual application to be replaced in simulation with the more concise synthetic trace. All these techniques are fundamentally orthogonal to response surface methodology because they are concerned with decreasing per simulation overhead via reduced input sets, rather than decreasing the total number of simulations via output prediction.

2.2 Statistical and Analytical Models

Yi et al. use a Plackett-Burman fractional factorial design of experiments to prioritize design parameters for statistical sensitivity studies [33]. By fixing unimportant parameter values to reasonable constants, they reduce the size of the design space that must be searched. They propose sweeping the critical parameters with more extensive simulations to actually optimize the design; GPRS identifies significant variables and builds an approximation function of their behavior at the same time, thus providing a comparable level of insight while obviating the need for further simulations. Joseph et al. derive linear performance models using stepwise regression [15]. Their models are not meant to be used to make predictions about design performance, and like Yi et al.'s are intended only to identify the interactions and significance of the various design parameters.

Lee and Brooks [19] perform regression modeling based on cubic splines in order to generate predictive approximation functions for performance and power. Their design space is quite large, and consists of over 20 billion design points across 21 benchmarks. They achieve predictions of up to 96% mean accuracy while sampling only one in 5 million design points [19]. The method they present is a statistical process rather than an automated algorithm, and requires some *a priori* intuition about variable relationships on the part of the architect [14]. They employ uniform random sampling to select a set of designs for full–scale simulation.

Lee and Brooks made a subset of their superscalar performance simulation datasets available to us, enabling an evaluation of the relative accuracy of the GPRS technique. As reported in Section 6, GPRS achieves comparable or improved accuracy based on smaller sample sizes. An advantage of GPRS over statistical and analytical models is that the response surfaces are generated automatically, rather than having to be carefully constructed and tuned by the architect based on their knowledge of the design space. Conversely, while vastly better than an exhaustive search, GPRS is still much more computationally intensive than any statistical technique.

2.3 Heuristic Design Space Search

Kumar et al. attempt to search through a large heterogeneous chip multiprocessor design space heuristically [18]. They encounter over 2 billion configuration options even before factoring in permutations of applications, and their study contrasts exhaustive search results with those obtained by a simplistic hill-climbing search algorithm. Hill–climbing search selects a configuration 4.5% worse than the configuration selected by the exhaustive global search, while requiring 86% fewer full simulations [18].

Kumar et al.'s methodology is fundamentally different from GPRS. Their hill–climbing search requires a full-scale simulation whenever any single design point needs to be evaluated in order to further guide the direction of the search. Every step of any heuristic search requires yet another set of detailed simulations; genetic programming will create fit approximation functions far faster than the search algorithm could ever hope to navigate the design space. Creating an accurate approximation function and evaluating it numerically is far more efficient and scalable than searching through the design space heuristically.

2.4 Artificial Neural Networks

Ipek et al. [14] propose training artificial neural networks (ANNs) to create predictive global approximation models of several computer architecture design spaces. Their neural networks in general achieve 97–99% mean prediction accuracy after only training on 1–2% of the design space [14].

Neural networks are a useful tool because of their ability to automatically learn non–linear functions. However, they also represent a "blackbox" in that they give the designer no explicit insight into the relationships that the network develops between the input variables and the response function. Genetically programmed response surfaces, by contrast, create non–linear yet explicitly defined functions, and thereby allow the designer to see exactly what the algorithm has learned from the sample data.

Ipek et al. made the exhaustive, full-scale performance simulation datasets from their memory and superscalar processor studies available to us for use in our own analysis of GPRS. This allows us to make a fair, direct comparison based on realistic simulated processor performance data between GPRS and the established ANN technique.

3 Mechanics of Genetic Programming

This section assumes the reader is familiar with the basic processes of standard genetic algorithms. Genetic programming is an evolutionary technique that produces functions that characterize the behavior of a response variable in terms of a set of design parameters. Like all evolutionary algorithms, the biological concepts of natural selection and survival of the fittest are at the heart of genetic programming: the algorithm is structured such that accurate candidate response surfaces are more likely survive and reproduce, and so subsets of their highly fit structure spread across generations, leading to a rising average fitness and the eventual convergence of the algorithm. Readers seeking more information about genetic programming in general should refer to [17], and those seeking information about genetically programmed response surfaces should refer to [1].

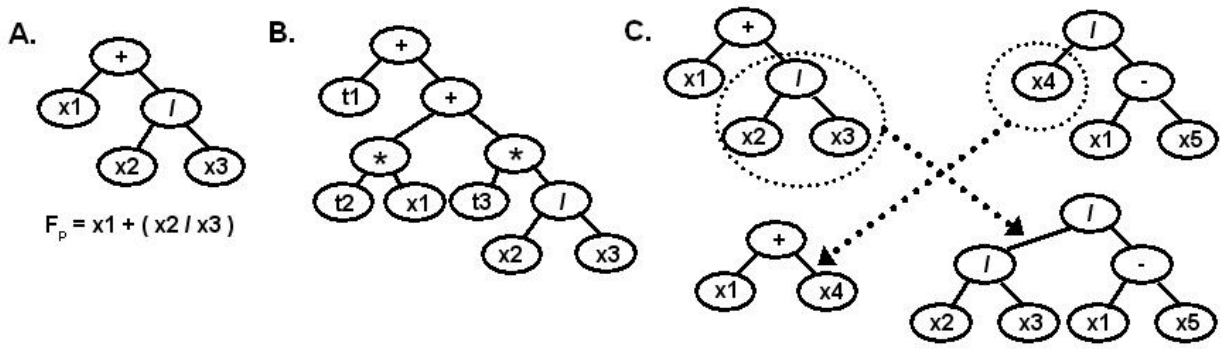

Figure 1: A. An expression tree B. Addition of tuning parameters C. Crossover breeding

The genetic programming process simply recombines and mutates a set of functions as it searches for one which fits well to the sample dataset. Response surface functions are encoded as binary expression trees, and the algorithm begins training with a population of randomly generated trees. Every generation, each candidate response surface is tuned to match the sample data as well as possible and is then evaluated. Candidates whose functions provide an accurate mapping between the design parameters and the performance metric are retained and recombined, while unfit surfaces are discarded. Eventually, the process converges on a small number of highly fit response surfaces.

3.1 Modeling Representation

Candidate response surfaces are represented as in-order expression trees, where an individual node in a tree represents a user-defined operator, an encoded design variable or a tuning parameter (Figure 1 A). Operators are defined by the architect – in this case we include simple arithmetic operators, square and square root operators and a logarithmic operator. If operators are undefined for certain inputs (e.g. divide by zero), a harsh fitness penalty is assigned to the expression tree containing them if such an exception is ever raised during fitness evaluation.

Cardinal and continuous design variables are encoded as floating point values, while boolean variables are converted into 0/1 values. These values are not normalized using minimax scaling because the use of tuning parameters (Section 3.3) serves to abstract the form of the approximation function from the data value ranges [1]. Nominal parameters (variables representing discrete choices with no inherent quantifiable properties) are represented with one-hot encoding [14].

3.2 Evolution

A candidate response surface's fitness is evaluated based on two metrics: the quality of the approximation of the experimental data by the candidate, and the size of its associated expression tree. Like Alvarez [1], we quantify the quality of the model by calculating the sum of squares of the difference between the candidate surface's predictions and the sample performance datapoints (Eq. 1). Penalizing lengthy expression trees serves to keep the equations more compact and globally fit, though obviously care must be taken not to promote oversimplification. In Equation 2 the constant **c** is used to control the size penalty; size is measured in terms of the number of tuning parameters (**ntp**) which have been introduced into the function. The genetic programming process attempts to minimize Equation 2. After a period where the fitness of of the best candidate fails to improve beyond a certain threshold, the GP algorithm is declared to have converged and is halted.

$$Q(S_i) = \frac{\sum_{p=1}^{P}(F_p - \tilde{F}_p)^2}{\sum_{p=1}^{P} {F_p}^2} \tag{1}$$

$$\Phi(S_i) = Q(S_i) + c * ntp_i^2 \rightarrow min \tag{2}$$

The evolutionary reproduction and mutation processes in GP are much like those of normal genetic algorithms. Candidates are selected for reproductive participation according to their fitness (with more fit candidates having a higher probability of being selected), and the selected candidates then swap randomly determined subtrees (Figure 1 C). With mutation, a randomly selected node in a randomly selected expression tree is mutated into a different node of the same type (i.e. operators mutate into other operators, design variables into other design variables).

A user–specified percentage of unfit individuals from the old generation are culled before ever getting a chance to reproduce — this is termed the kill percentage. Removing the most unfit individuals encourages further breeding between the more successful members of the population. A user–specified percentage of the most highly fit individuals are also added directly into the next generation unchanged — this is termed the elite percentage. These individuals may still breed and produce offspring for the new generation, but transferring them directly enables the best candidate solutions found so far to be preserved for multiple generations.

3.3 Tuning Parameters

A critical but completely automated step of the genetic programming process is the assignment of tuning parameters to a candidate expression tree. Tuning parameters abstract the structure of the expression from the specific data value ranges; using them results in simpler approximation functions [1]. Essentially, these added parameters serve to mold the generic structure provided by the expression tree so that it fits as closely as possible to the sample data. Tuning parameters are added by the algorithm to an expression tree deterministically based on the tree's topology, and their values are automatically adjusted to fit the candidate surface to the data (Figure 1 B). The effectiveness of the overall genetic programming technique hinges on the ability to detect fit candidates; poor tuning will result in theoretically fit expressions being inadvertently discarded by the technique.

For the tuning process itself, our implementation of the GP algorithm uses an open source implementation of the Levenberg–Marquardt algorithm for solving nonlinear least–squares problems. Levenberg–Marquardt is an iterative technique that can be thought of as a combination of steepest descent and the Gauss–Newton method [21], with the method of convergence changing depending on its proximity to a solution. Initial guesses at tuning parameter values are automatically supplied to the L–M algorithm by short runs of a generic genetic algorithm.

3.4 Design of Experiments

We choose to use a design of experiments (DOE) technique known as the Audze-Eglais Uniform Latin Hypercube design of experiments [2] to select the points included in the sample set of performance data used in the genetic programming of the response surfaces. Audze-Eglais selects sample points which are as evenly distributed as possible through the design space by formulating the problem as one of gravitational attraction and minimizing the "potential energy" of the system of points. We formulate our Audze-Eglais DOEs using the optimization technique described by Bates et al. [3].

A major benefit of using this particular DOE is that it gives us high confidence that even if very few points are included in the sample set used for training, they will still be evenly distributed across the space of all possible designs. Random sampling [19] is unable to guarantee such a fair representation, and sampling based on variance [14] does not allow us to collect the complete set of sample data prior to the response surface training process. Plackett-Burman DOEs like those used by Yi et al. [33] are actually overly sparse for design spaces of this size — the time spent completing the tens of extra full–scale simulations suggested

name	parameter	values
x0	fetch/issue/commit width	4,6,8 instructions
x1	frequency	2,4 GHz
x2	max branches	8,32
x3	branch predictor	1,2,4K entries
x4	branch target buffer	1,2K sets (2-way)
x5	ALUs/FPUs	2 choices per issue width
x6	ROB size	96,128,160
x7	Register file	2 choices per ROB size
x8	Ld/St queue	16/16,24/24,32/32
x9	L1 ICache	8,32Kbytes
x10	L1 DCache	8,32Kbytes
x11	L2 Cache	256,1024Kbytes

Table 1: Ipek et al. processor design space variables

name	set	cardinality
x0	depth	10
x1	width	3
x2	physical registers	10
x3	reservation stations	10
x4	I-L1 cache	5
x5	D-L1 cache	5
x6	L2 cache	5
x7	main memory	10
x8	control latency	2
x9	fixed-point latency	5
x10	floating point latency	5
x11	memory latency	5

Table 2: Lee and Books processor design space variable sets

by the Audze–Eglais design is certainly offset by the increased confidence we can have in the robustness of the resulting model.

4 Experimental Methodology

We use simulated performance data previously collected by Ipek et al. [14] and Lee et al. [19] for their recent predictive design space exploration studies. These datasets allow us to present a direct comparison of the accuracy of the three techniques, and also provide a validation of the effectiveness of GPRS at modeling realistic architectural design spaces. The Ipek et al. processor design space contains 12 design variables, resulting in over 20K unique design points. The Lee and Brooks design space has 23 design variables divided into 12 groups, with all variables assigned to the same group being varied together. The finer–grained parametrization of the design variables in the Lee and Brooks study results in approximately one billion design points. Both sets of performance data were collected from detailed simulations running over a subset of the SPEC2k CPU benchmarks [7]. Ipek et al. made their benchmark selections based on metric similarity clustering suggestions provided by Phansalkar et al. [22], and used the MinneSPEC [16] reference inputs [14]. Lee and Brooks created their input traces by sampling the SPEC full reference input set [19].

We use a tool based on [3] to automatically generate an Audze-Eglais Design of Experiments of a specified size for a user-supplied list of parameters and their associated valid bindings.. The size of the sample set varies depending on the size of design space being studied — a hard lower bound for most DOEs is that there must be at least as many sample points as there are dimensions. The output of our DOE formulator is a list of points within the global design space that will become the training set for the GPRS formation algorithm. For the purposes of this study, we then simply extract these datapoints out of the previously collected set of performance data, but in a brand new optimization study we would simulate each of them in full detail. The subset of collected performance data becomes the input to our GPRS builder application.

In selecting the parameters used to control the operation of the GP algorithm, we generally adhere to the

IPC	$1.322 + (1.834e-06 * ((-69835.5 * x1) - (((-1972.46 * x6) - ((-1129.92 * (x8 * x1)) - (0.359 * (x11 * ((-132.256 * x0) - (289.655 * x1))))))) + ((-1.318 * (x8 * (x6 * x0))) - ((-363.66 * (\frac{x1}{x5}) * (x11 * x0))) - (71303.8 * (\frac{x6}{x5})))))))$
Branch Prediction	$0.9616 + (0.000581 * (x3 * x4))$
L2 Miss Rate	$0.2824 + (-3.675e-06 * (((-1694.59 * x10) + (-420.003 * (\frac{x11}{x10}))) + (140.062 * x11)))$

Table 3: GPRS approximation functions generated for case study.

values suggested by Alvarez in his genetic programming optimization studies [1]. We use an initial population size of 500 candidate expression trees, limit the maximum number of generations to 1000 (the GP process almost always converged after fewer than 100 generations), set the size penalty constant **c** to 0.000001, and use a 10% elite percentage and 10% kill percentage to aid in convergence. Because a response surface is simply an analytic equation, off-the-shelf numeric solvers can be used to rapidly locate the predicted global optima and examine other features of the surface (we use Mathematica ®).

This short chain of applications is already almost fully automated, and represents an extremely powerful infrastructure for architects faced with difficult design optimization tasks. With this in mind, we are working to formalize a framework for taking a set of a benchmarks, a simulator configuration script, and a list of design parameters with their bindings and automatically formulating DOEs and creating GPRSs based on them, with no further user input required. Such a tool would ideally be publicly released with the final version of this manuscript.

5 A Simple Case Study

Using genetic programming to derive response surfaces is extraordinarily easy. The steps in the process are straightforward: determine which points will be included in the sample set, simulate those design points, run an automated tool on the collected data, and numerically optimize the resulting response surface. To emphasize the ease architects will have in creating genetically programmed response surfaces and in understanding their results, we present the following illustrative case study.

The superscalar processor design space described in Table 1 contains twelve design dimensions, each with several possible parameter values. As architects confronting the problem of finding the best configuration for such a superscalar processor design, we abhor the idea of fully simulating all 20K possible configurations. Creating an analytical regression model would require us to specify predictors for a response surface using domain-specific knowledge, and then carry out tests of statistical significance and fitness to tune the our regression model. An automated response surface builder requires less *a priori* architect knowledge and less architect effort to produce conclusive results.

The first step in generating a GPRS is to select a set of sample points from the global design space. The naive way to do this is pick points uniformly at random, but formulating a Design of Experiments is a more robust alternative. Plackett–Burman [33], Audze–Eglais [2], and Latin Hypercube are all proven methods for formulating DOEs. For the DOE-based results presented in this paper, we generate an Audze-Eglais DOE automatically using a tool based on [3].

After DOE formulation or random selection, we have a list of points in the design space. Each point on the list must be simulated to evaluate those performance measures that we are interested in optimizing or investigating. These simulations may take some time, but collecting this data is certainly less time consuming than exhaustively simulating millions or billions of possible designs. For our simple case study, we use measurements of IPC, branch prediction miss rate and L2 cache miss rate for the SPEC2k CPU benchmark application applu.

The performance data collected from the simulations are stored in a file that is fed to the automated GP-based response surface creation tool. Via evolution and selective breeding, the tool produces approximation equations for predicting performance based on the design variables listed in Table 1. The equations produced for our case study are reported in Table 3; all the equations had $< 3\%$ mean global error when tested against performance data from an exhaustive global search.

By examining the equations in Table 3 and variables in Table 1, we can see that the design variables that are useful in predicting the response of the performance measure change depending on which metric we use. Branch predictor size and the maximum number of branches are the variables that prove most useful

in predicting branch prediction accuracy, L2 cache size and L1 data cache size are the best predictors of L2 miss rate, and IPC is based on a more complicated combination of many variables' effects. None of the ratios or variable relationships were specified in advance — the genetic programming algorithm is wholly responsible for identifying the relevant variables and their relationships.

These conclusions are rather intuitive, but this basic case simply serves to further validate the technique. Such an instantly comprehensible response would not be possible using a trained artificial neural network. While an ANN would eventually allow us to draw the same conclusions, we would have to query and probe the network repeatedly and test the effect of changing different variable values in order to gain the same level of insight. The function can also be used as–is to instantly predict the performance impacts of making a change to a baseline design (e.g. a decrease in L2 cache size will decrease IPC by an analytically determinable amount).

However, our primary goal is not just to understand the relative impact of various design parameters on a performance measure, but also to actually optimize our target design. Given a tuned response surface embodied in an explicit approximation function, there are two ways this could be done. The brute force approach would be to rapidly evaluate every design point using the response surface equation and select one of the ones with the best predicted performance. A slightly more refined approach involves optimizing the variables included in the approximation function analytically or numerically, and then exploring the design points in the remaining subspace more thoroughly.

For our IPC case study, we numerically located the maximum of our approximation function. The following values are the parameter bindings which maximize IPC: {x0 = 8, x1 = 2, x6 = 160, x8 = 32/32, and x11 = 1024}. It would have been difficult to tell *a priori* that these specific variables were the most important design considerations. Furthermore, validating the predictive equation against the exhaustive simulation data reveals that as long as the design we choose incorporates these parameter bindings, the worst possible performance we could end up with is within 2.5% of the true global optima for IPC.

Using genetically programmed response surfaces to optimize designs merely requires architects to make use of a short sequence of automated processes. No matter how they choose to use the resulting approximation function, they save a huge amount of time that would have otherwise been spent running large numbers of detailed simulations. Our case study illustrates the ease of using the GPRS technique as a design optimization tool, as well as some of the benefits conferred by this technique to the architect in the form of insight into the target design space.

6 Experimental Results

We strove to address the following questions in our more general experimental studies:

1. *Global accuracy.* Do the equations generated by GPRS accurately and robustly depict the true behavior of the performance measures?

2. *Sample sizes.* How much of the design space must be simulated in full detail in order to produce accurate predictions?

3. *Identification of optima.* Does GP give the architect insight into which variables are important and which subspaces should be explored? How close to the true global optima are the points in the recommended subsets?

4. *Comparison with alternatives.* How well does GPRS perform as compared to other recently proposed techniques? What are the possibilities for integration?

Our first study makes use of data provided by Ipek et. al [14]. They conducted an exhaustive search of the design space delineated by Table 1 for a subset of the SPEC CPU 2000 benchmark suite [7]. We take this exhaustive, fully–simulated set of datapoints as our global population, and use an Audze-Eglais DOE to select the sample set of performance data used to train the GPRS.

We calculate percentage errors between the predicted values suggested by the response surface equation with the detailed simulation data. We calculate error across *all* collected data points — in this case, the entire design space — and find that the GPRSs provide highly accurate predictions of design performance on a global scale. This effectiveness is clearly illustrated by the learning curves in Figure 2. With sample

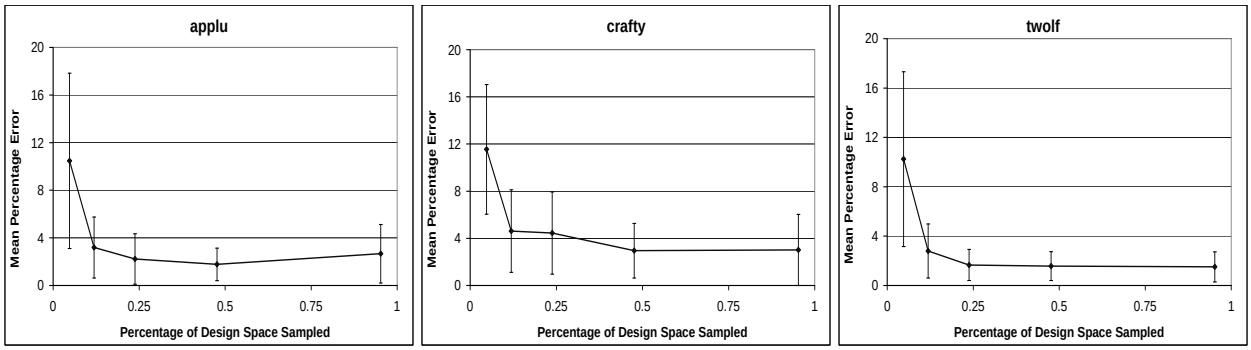

Figure 2: Mean GPRS error decreases as sample size grows. Error bars are 1 standard deviation.

sizes as small as 0.1%, the GPRS predictions of performance for all the non-sampled points achieve 2.8–4.9% mean error with standard deviations between 2.2–3.4%. When 0.5% of the design space is sampled, this result improves to 1.2–3.0% mean percentage error with standard deviations in the range of 1.1–2.3%. These results are generally comparable to the performance of the neural network–based technique, but with better accuracy achieved at the smaller sample sizes. Making successful predictions at very small sample sizes will become increasingly important as architectural design spaces continue to grow exponentially.

We are also concerned with the distribution of prediction error across the design space. Not only do we desire a low variance in error, but we also do not want to badly mispredict any individual points. To explore the worst of our predictions we use cumulative density functions of individual point error, in the fashion of both Ipek et al. [14] and Lee and Brooks [19]. Figure 3 shows that at a 0.5% sample size maximum percentage errors on any one point range up to only 15% for even the most poorly predicted benchmarks, with at least 80% of the points having a prediction percentage error smaller than 5%. On the best–predicted benchmarks over 94% of the points have percentage errors $<$ 4% and the maximum error is only 7%. Maximum error values for the ANN technique were not reported for the processor benchmarks.

Our second study made use of performance and power data that were collected from a superscalar processor design space by Lee and Brooks [19] to test their regression modeling technique for performance prediction. The design space used is described in Table 2, and they too simulated over a subset of the SPEC CPU 2000 benchmark suite. Their statistical analysis achieved median error rates as low as 4.1% for performance and 4.3% for power (with maximum errors of 33.1% and 45.2% respectively) when sampling fewer than 4000 design points out of over 1 billion per benchmark. Lee and Brooks used uniform random sampling when they selected the points used to formulate their regression models. For this study, we likewise use random sampling rather than a DOE — this allows us to evaluate the comparative effectiveness of GPRS independent of the more robust sampling method. Figure 4 uses cumulative density functions of percentage error to illustrate the performance of GPRS on this large design space.

At sample sizes as low as 200 out of 1 billion design points per benchmark, we find that the median percentage error of the predictions provided by the GRPSs range from 1.1% to 6.1%, with variances of 2.3% and 9.9% respectively. Maximum errors range between %23.1 and 44.8%; in general these values are outliers. Because the Lee and Brooks dataset does not represent an exhaustive search of the design space, we cannot perform the same global evaluation we are able to use on the Ipek et al. data. Instead, we evaluate GPRS error based on 300 datapoints not included in the sample set. Again we find that GPRS attains accuracy comparable to the alternative technique, but achieves it at smaller sample sizes. The reduction of six orders of magnitude in the total number of required simulations afforded by using GPRS as compared to an exhaustive search also speaks strongly to the merit of the technique.

It is interesting to note that our outlier error values are generally larger than those reported by Lee and Brooks, even though GPRS had lower median errors. Figure 5 graphs the absolute accuracy of the GPRS predictions as compared to a subset of the simulated performance data. The benchmark equake was the best predicted, while twolf was one of the worst. While there seems to be a slight bias towards underestimation in the the 0.4 to 0.7 IPC range for twolf, there is not strong evidence of a systematic bias generating outliers in these benchmarks.

Recall that genetically programmed response surfaces are explicit approximation functions relating design variables to performance measures, and we can make use of their explicit nature to gain insight into the

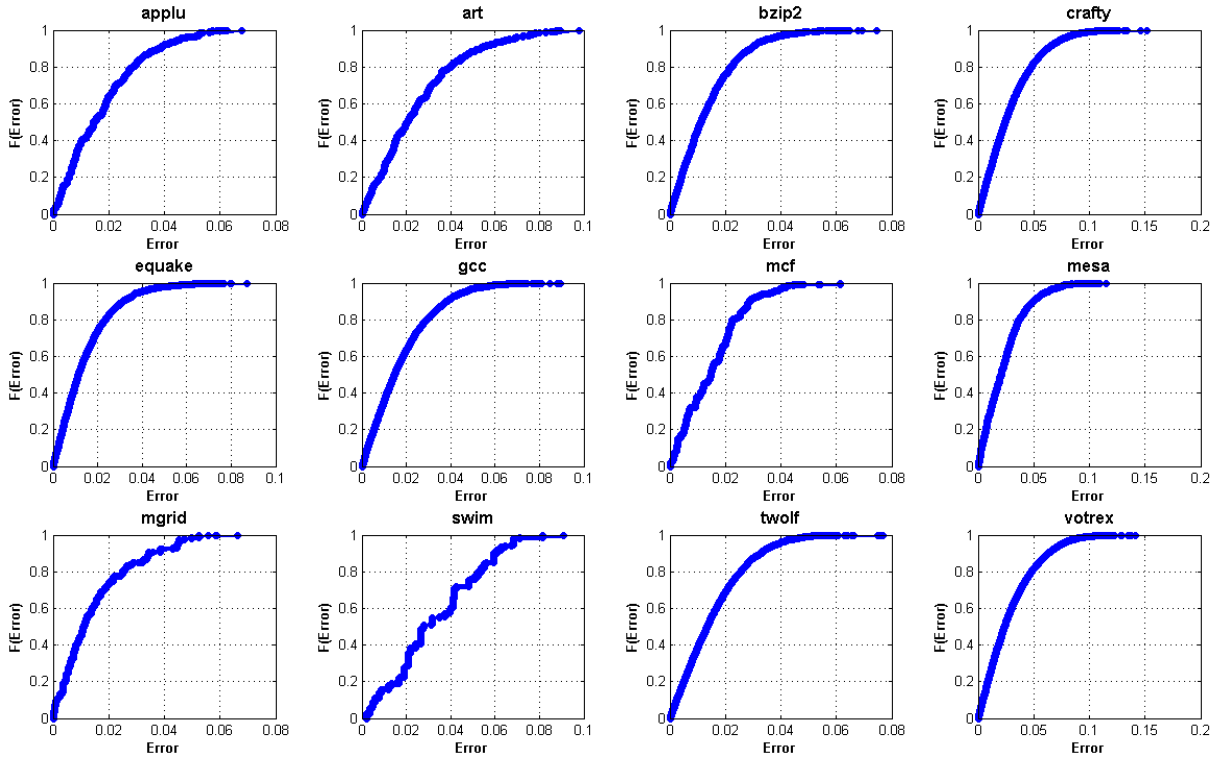

Figure 3: CDFs of GPRS error for Ipek et al. design space at a 0.5% sample size. The Y-axis represents the percentage of design points with prediction errors < the X-axis value.

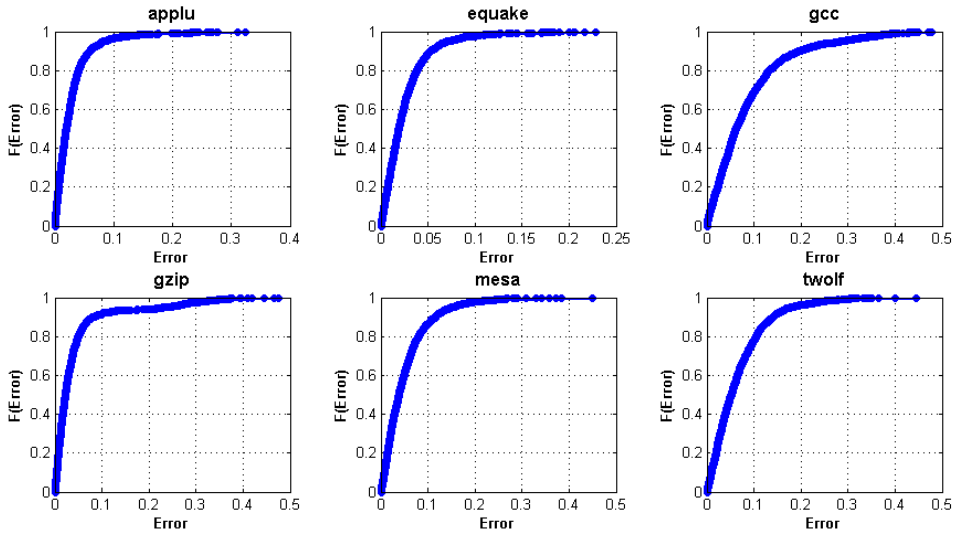

Figure 4: CDFs of GPRS error for Lee et al. design space at a 0.000002% sample size

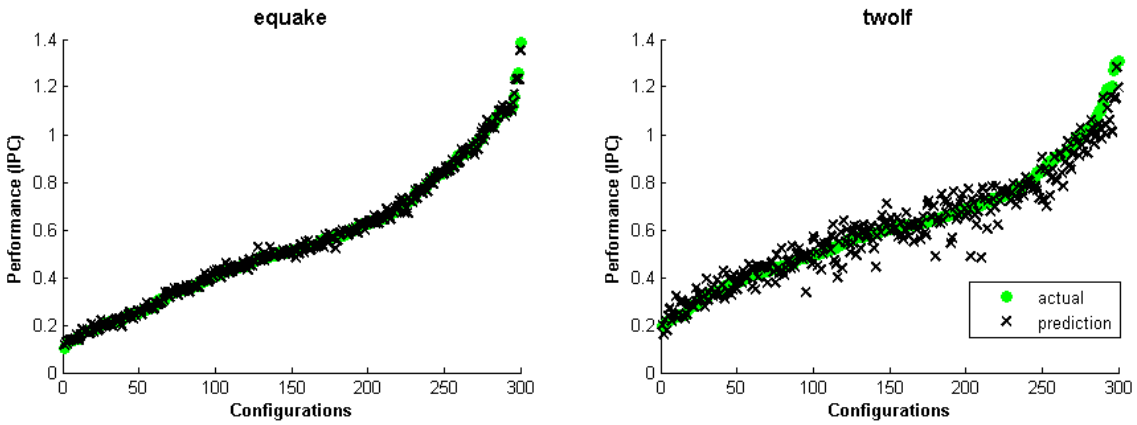

Figure 5: IPC values predicted by GPRS vs IPC values recorded by Lee et al.for 300 non-sampled configurations

benchmark	(1) design parameter bindings												(2) true	(3) pred.	(4) worst	(5) %
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11				
applu	8	2	•	•	•	•	160	•	64	•	•	1024	2.25	2.235	2.19	100
art	8	2	32	•	•	•	160	•	64	•	•	1024	0.53	0.537	0.51	100
bzip2	8	2	32	•	•	•	160	•	64	•	•	1024	1.48	1.433	1.42	100
crafty	8	2	•	4	2	8	•	112	64	•	32	•	1.76	1.744	1.62	100
equake	8	2	32	4	•	8	160	•	64	•	•	1024	1.66	1.627	1.61	100
gcc	8	2	32	4	2	8	•	•	64	32	32	•	1.29	1.269	1.15	100
mcf	•	2	32	•	•	•	160	•	64	•	•	1024	0.58	0.559	0.54	100
mesa	8	2	32	•	2	8	160	•	64	•	•	•	3.04	2.909	2.87	100
mgrid	8	2	•	•	•	8	•	•	64	•	•	1024	1.73	1.723	1.41	100
swim	•	2	•	•	•	•	160	•	64	•	•	•	0.95	0.936	0.89	100
twolf	8	2	32	4	2	8	•	•	64	•	32	256	1.01	0.979	0.93	75
vortex	8	2	•	4	2	8	160	•	64	•	•	1024	2.48	2.362	2.31	100

Table 4: Summary of GRPS results after training on Ipek et al.'s processor performance data with sample size of 0.5%. (1) Parameter bindings determined by numerically optimizing the GPRS function. • indicates that the parameter was dropped from the response surface function. (2) True global optima in IPC as determined by exhaustive search. (3) Optimal IPC value predicted by GRPS. (4) Worst value in subset delineated by parameter bindings. (5) Percent of true global optima contained in subset delineated by parameter bindings.

target design space. We found that the most accurate functions for the Ipek et al. processor design space included 5–9 of the 12 design variables. As can be seen in Table 4, different variables were identified as significant for different applications and performance measures. Intuitively, this makes sense considering that different architectural characteristics will have different performance impacts on memory-bound applications as compared to CPU-bound ones. For example, a benchmark which drops cache parameters from its response surface due to their irrelevant impact on performance is clearly not memory-bound. On the architecture side, it becomes quite clear when looking at the response surfaces generated for a target benchmark or set of benchmarks which design decisions will actually have a significant impact on performance. Furthermore, the response surface makes it easy to predict exactly what the performance impact of a given design decision will be.

Given a GPRS, a simple numerical optimization with off-the-shelf software rapidly determines the optimal values for the significant variables. We numerically calculate these parameter bindings for the response surfaces generated by the Ipek et al. processor performance data (Table 4). The remaining variables form a subset of the design space that contains the true global optima — in all but one of our tests on the Ipek et al. design space, 100% of the true global optima are contained within this subspace. An architect could search this subspace to find the precise optimal values, but even if they just assign the remaining variables to be random values within the initial sample range, the resulting design will still be near optimal. We find

that the performance of even the worst point within the subset space is always within 19% of the true global optima, and for most benchmarks was within 8%. The predicted optimal value for IPC was also very close to the value of the true optima in all cases.

Even a 0.1% sample size represents a reduction of three orders of magnitude in the number of simulations required per optimization. Our success with the Lee and Brooks dataset demonstrates that accurate predictions at even lower sample percentages are certainly feasible when the design space is large enough; we become limited only when the size of the sample becomes smaller than the number of design dimensions. Furthermore, due to our use of a formal DOE and the interpolation provided by our response surface, only a few more samples are needed per dimension added to the problem. This collapses the exponential growth of the design space back down to a linear increase in the size of the sample set.

We find that in practice the time required to create a highly accurate GPRS is two orders of magnitude larger than the few minutes required to train an artificial neural network [14]. There is some variance depending on how difficult it is for the surface to capture the design space, but generally for a population size of 500 running completely unparallelized we find that every additional 100 sample points adds about two hours to convergence time. However, the time spent creating and evaluating a GPRS is still significantly smaller than the time required to complete a large set of full–scale simulations, and the GPRS technique's increased effectiveness at smaller sample sizes may in some cases more than make up for the increased training time when compared to artificial neural network approaches on large design spaces. A further consideration is the embarrassingly parallel nature of most portions of the GPRS algorithm (e.g. the evaluation of candidates' fitness, offspring mutation, etc.), of which our current implementation does not take full advantage. This innate parallelism makes GPRS an appealing algorithm to run on state-of-the-art multicore architectures.

7 Conclusions and Future Work

Using genetically programmed response surfaces to optimize architecture designs only requires architects to make use of a short sequence of fully automated tools. In exchange, architects can anticipate vast reductions in simulation time, rapid identification of important design parameters and subspaces, and can maintain a high degree of confidence in the predictive results. Genetically programmed response surfaces predict the performance impacts of proposed design changes without requiring additional detailed simulation data, and they will scalably address future increases in design complexity.

Ultimately, genetic programming is a superior technique because it provides the architect with the explicit expression of an approximation function containing design variables proven to have a significant impact on the target performance measure. Having an explicit equation allows the architect to rapidly optimize these highly relevant design variables using a numerical method of their choice. The remaining variables, those not included in the approximation equation, make up a subset of the design space that is near optimal, and one which as far as we have seen will contain the global performance optima.

Exploring the subset design space made up of these trivial–impact design variables may allow the architect to fine–tune their design in order to achieve maximal performance. The architect can identify narrow regions of interest by exploring designs similar to the optima of the approximation function. However, the architect simultaneously can have a high level of confidence that the variables which were not expressly included in the approximation function are those which do not significantly impact performance in the first place.

Unlike the heuristic hill-climbing design space search performed in [18], the time our technique takes to locate optima is decoupled from the length of time it takes to run a full–scale simulation — simulations of the small number of design points in the sample set can all be done as a preprocessing step. The predictions generated by a GPRS are extremely accurate even when trained on small sample sets, which means that as design spaces grow exponentially the sample size only needs to grow slightly to keep accuracy high.

The GPRS method we have proposed has a high degree of potential for integration with other techniques. Insights gained into variable relationships could be useful in identifying predictors to create more accurate regression models [19]. As with ANNs [14], SimPoint's reduction in instruction stream sizes represents an orthogonal and complementary way to reduce total simulation time. The GP technique is highly parallelizable, which means that the process of generating approximation functions will become even more efficient as modern hardware further emphasizes parallelism. Improving the parallelization of the technique, integrating it with other predictive methods, and applying it to problems with multiple fitness criteria are all directions of future work. Our effort to formalize the tools used in this work into a robust framework for GPRS-based architecture optimization is ongoing, and we intend to release a public version with the final version of this

manuscript.

Genetically programmed response surfaces provide architects with predictive polynomial equations which can be trivially solved to accurately estimate what a long and costly cycle-accurate simulation would otherwise have produced. The time saved by using a GPRS allows for the few detailed simulations required for training to be even more detailed, thus improving the overall efficiency and realism of the process. Genetically programmed response surfaces produce fairly intuitive functions that clearly expose the relationship and relative importance of various configuration parameters. As design spaces continue to grow increasingly complicated and sizable, predictive techniques which are scalable and insightful will become fundamental to producing effective computer architecture optimizations.

8 Acknowledgments

This work was supported by NSF grant nos. CNS-0509245, CCF-0429765, and their associated REU supplements. This work grew out of a summer REU project and subsequent senior-thesis research [6]. We would like to thank Sally McKee, Engin Ipek, David Brooks, and Benjamin Lee for their willingness to provide their data (representing a staggering amount of computational effort) and answer questions. We would also like to thank Paul Reynolds, A. Benjamin Hocking, and David Tarjan for their helpful comments.

References

[1] L. Alvarez. *Design Optimization based on Genetic Programming.* PhD thesis, University of Bradford, 2000.

[2] P. Audze and V. Eglais. A new approach to the planning out of experiments. In *Problems of dynamics and strength*, volume 35, pages 104–197, 1977. In Russian.

[3] S. J. Bates, J. Sienz, and D. S. Langley. Formulation of the audzeeglais uniform latin hypercube design of experiments. *Advanced Engineering Software*, 34(8):493–506, Jun. 2003.

[4] G. Box and N. Draper. *Empirical modelbuilding and response surfaces.* John Wiley and Sons, 1987.

[5] T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In *Proceedings of the 1996 IEEE International Conference on Computer Design*, Oct. 1996.

[6] H. Cook. *Optimizing Chip Multiprocessor Designs Using Genetically Programmed Response Surfaces.* PhD thesis, University of Virginia, May 2007.

[7] Standard Performance Evaluation Corporation. Spec cpu benchmark suite. http://www.specbench.org/osg/cpu2000/, 2000.

[8] J. Davis, J. Laudon, and K. Olukotun. Maximizing cmp throughput with mediocre cores. In *Proceedings of the IEEE/ACM International Conference on Parallel Architectures and Compilation Techniques*, pages 51–62, Oct. 2005.

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

[10] L. Eeckhout, J. Sampson, and B. Calder. Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation. In *Proceedings of the First Annual IEEE International Symposium on Workload Characterization (IISWC)*, Oct. 2005.

[11] J. Emer and N. Gloy. In *Proceedings of the 24th Annual ACM/IEEE International Symposium on Computer Architecture*, Jun. 1997.

[12] S. Eyerman, L. Eeckhout, , and K. D. Bosschere. The shape of the processor design space and its implications for early stage explorations. In *Proceedings of the 7th WSEAS International Conference on Automatic Control, Modeling and Simulation*, Mar. 2005.

[13] J. W. Haskins, Jr. and K. Skadron. Memory reference reuse latency: Accelerated sampled microarchitecture simulation. In *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 195–203, Mar. 2003.

[14] E. Ipek, S.A. McKee, B.R. de Supinski, M. Schulz, and R. Caruana. Efficiently exploring architectural design spaces via predictive modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2006.

[15] P. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. Construction and use of linear regression models or processor performance analysis. In *Proceedings of the Twelfth IEEE International Symposium on High-Performance Computer Architecture*, Feb. 2006.

[16] A J KleinOsowski and David J. Lilja. Minnespec: A new spec benchmark workload for simulation-based computer architecture research. *IEEE Computer Architecture Letters*, 1(1), 2006.

[17] J. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, 1992.

[18] R. Kumar, D. M. Tullsen, and N. P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *Proceedings of the 2006 IEEE/ACM/IFIP International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2006.

[19] B. Lee and D. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the 12th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2006.

[20] Y. Li, B. C. Lee, D. Brooks, Z. Hu, and K. Skadron. Cmp design space exploration subject to physical constraints. In *Proceedings of the Twelfth IEEE International Symposium on High-Performance Computer Architecture*, Feb. 2006.

[21] K. Madsen, H.B. Nielsen, and O. Tingleff. *Methods for NonLinear Least Squares Problems*. IMM, Technical University of Denmark, 1999.

[22] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John. Measuring program similarity: Experiments with spec cpu benchmark suites. In *Proceedings of the 2005 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-05)*, pages 10–20, Mar. 2005.

[23] L. Schmit and B. Farshi. Some approximation concepts for structural synthesis. *AIAA Journal*, 12(5):692–699, 1974.

[24] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the 2001 IEEE/ACM/IFIP International Conference on Parallel Architectures and Compilation Techniques*, pages 3–14, Sept. 2001.

[25] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings 10th Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, pages 45–57, Oct. 2002.

[26] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark. Branch prediction, instruction-window size, and cache size: Performance tradeoffs and simulation techniques. *IEEE Transactions on Computers*, 48(11):1260–81, Nov. 1999.

[27] K. Skadron, M. Martonosi, D. August, M. D. Hill, D. J. Lilja, and V. S. Pai. Challenges and recommendations for computer architecture evaluation. *IEEE Computer*, 36(8):30–36, Aug. 2003.

[28] V. Toropov and L. Alvarez. Application of genetic programming and multipoint approximations to optimization problems with random noise. In *Annual GAMM Meeting*, 1998.

[29] V. Toropov and L. Alvarez. Application of genetic programming and response surface methodology to optimization and inverse problems. In *Proceedings of the International Symposium on Inverse Problems in Engineering Mechanics (ISIP)*, 1998.

[30] V. Toropov, L. Alvarez, and H. Ravaii. Recognition of damage in steel structures using genetic programming methodology. In *Proceedings of the Second International Conference on Identification in Engineering Systems*, pages 382–391, 1999.

[31] G. Venter, R.T. Haftka, and J.H. Starnes. Construction of response surfaces for design optimization applications. In *Proceedings of the 6th Symposium on Multidisciplinary Analysis and Optimization*, 1996. AIAA paper 964040CP.

[32] R. Wunderlich, T. Wenish, B. Falsafi, and J. Hoe. Smarts: Accelerating microarchitectural simulation via rigorous statistical sampling. In *Proceedings of the 30th IEEE/ACM International Symposium on Computer Architecture*, pages 84–95, Jun. 2003.

[33] J. Yi, D. Lilja, and D. Hawkins. A statistically rigorous approach for improving simulation methodology. In *Proceeding of the International Symposium on High Performance Computer Architecture (HPCA)*, pages 281–291, Feb. 2003.