Predictive Design Space Exploration Using Genetically Programmed Response Surfaces

Henry Cook
Department of Electrical Engineering and Computer Science
University of California, Berkeley
hcook@eecs.berkeley.edu

Kevin Skadron
Department of Computer Science
University of Virginia
skadron@cs.virginia.edu

ABSTRACT

Exponential increases in architectural design complexity threaten to make traditional processor design optimization techniques intractable. Genetically programmed response surfaces (GPRS) address this challenge by transforming the optimization process from a lengthy series of detailed simulations into the tractable formulation and rapid evaluation of a predictive model. We validate GPRS methodology on realistic processor design spaces and compare it to recently proposed techniques for predictive microarchitectural design space exploration.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques

General Terms

Design, Measurement, Performance

Keywords

genetic programming, predictive modeling, design space exploration

1. INTRODUCTION

Computer 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 [5, 18]. 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. The design space of a single core is already large; the multi-core or system-on-a-chip design spaces are nearly intractable because core architecture, core count, cache sizes, interconnect topology, energy efficiency, and thermal efficiency are *all* inter-related [18]. Design–specific analytical models of some components might be developed that are trivial to solve, but a model simple enough to derive *a priori* typically cannot account for emergent interactions between multiple variables.

DAC 2008, June 8–13, 2008, Anaheim, California, USA

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 problem is neither new nor unique to the computer architecture or design automation communities. 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, 21]. Given a sample of collected performance data, these techniques build approximation functions ("response surfaces") which predict the global performance of all candidate designs.

Techniques for creating globally accurate response surfaces range from simple linear regressions to highly non-linear artificial neural networks. We advocate the use of *genetic programming* [15] to create appropriate non-linear, polynomial approximation functions and fit them to 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 sample data points obtained from full–scale simulations [1].

Unlike a standard search algorithm, GP uses the evolutionary process to improve the *accuracy of a set of predictive functions*, rather than to heuristically improve the performance of a set of designs. Automatically creating a predictive approximation function becomes increasingly efficient compared to a direct heuristic search as design spaces grow. GPRS almost completely decouples the time it takes to optimize a design from the time required to run a cycle–accurate simulation. The simulations used can afford to be complex and realistic, and this increased complexity has a negligible impact on the time it takes to explore the entire design space and select optimal designs.

On a superscalar processor design space consisting of over 20K design configurations from [11], genetic programming creates response surfaces that globally predict IPC with less than 2% mean percentage error and less than 9% worst case error, based on sample sizes of less than 1% of the total design space. On a second superscalar processor design space containing almost billion design points from [17], we find that median percentage errors of the predictions range from 1.1% to 6.1% at sample sizes as low as 200 design points. This reduction in the number of required simulations by six orders of magnitude demonstrates the dramatic time-saving potential of automatically generated predictive models.

2. PREDICTIVE MODELING

GPRS and other predictive modeling techniques are fundamentally different from design optimization strategies based on heuristic searches, including genetic search algorithms. Predictive models are built using the simulated results of a small sample of designs picked in advance. Designers using heuristic searches do not know in advance what path the search will take through the design space, and so potentially expensive simulations must be run whenever the search algorithm needs to evaluate a design point.

GPRS avoids coupling the search time of the algorithm to the running time of the simulation. In other words, the GPRS process costs $O(simulationTime * sampleSize + evaluationTime * stepsToConverge)$ whereas a heuristic search costs $O(simulationTime * evaluationTime * stepsToConverge)$. For GPRS, *stepsToConverge* is relatively agnostic to design space dimensionality and *sampleSize* grows only linearly with design space dimensionality. In order to offer comparable scalability, the time to convergence of a heuristic search could not increase by more than a few evaluations per dimension added to the design space.

Figure 1 shows how many design evaluations a heuristic search could afford to make before a predictive search would be more time efficient for various simulation run times, assuming 10 possible bindings per dimension and two hours added to GPRS evaluation time per dimension (see Section 6). For truly large design spaces, it seems unlikely that a heuristic search could converge on a solution after so few design point evaluations, and therefore also unlikely that one would be more efficient than constructing a predictive model.

Another fundamental difference between predictive model generation and heuristic searching is the level of insight the end result provides into the nature of the design space. GPRS provides a global map of the entire design space in the form of an analytic function, making explicit to the architect which variables have a significant impact on the target performance measure and what interrelationships exist among them. A global view of performance behavior across the entire design space is a superset of the guidance an architect could expect from a heuristic search–based approach.

While we review some of the related work pertaining to automated design optimizations using heuristic searches, we focus our comparison studies on several recently proposed techniques for predictive exploration of microarchitectural design spaces in out–of–order execution (OOE) processors.

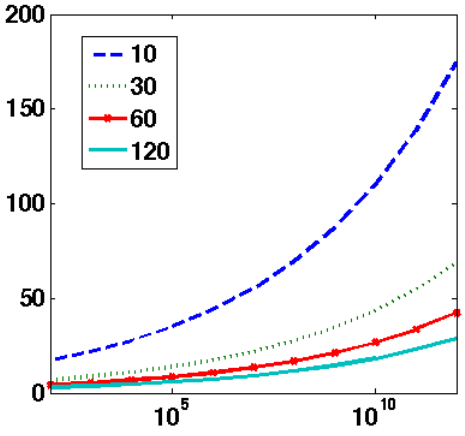

Figure 1: Number of evaluations heuristic search can afford before GPRS becomes more efficient, for various full–scale simulation times (in minutes). X–axis is number of design points, y-axis is number of evaluations.

3. OTHER 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 [21]. GPRS specifically has been used for optimizing the calcination of roman cement and predicting stress fractures in steel [1]. Genetic programming in general has been previously used in the computer architecture domain in the context of automatically synthesizing accurate branch predictors [8].

There have been many efforts to speed up the exploration process by reducing the number of instructions executed in simulation [6, 7, 14, 20]. 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.

Yi et al. use a Plackett-Burman fractional factorial design of experiments to prioritize design parameters for statistical sensitivity studies [22]. Joseph et al. derive linear performance models using stepwise regression [12]. These techniques are not meant to be used to make predictions about design performance, but rather to identify the interactions and significance of the various design parameters.

Eyerman et al. [10] and Karkhanis and Smith [13] propose CPI adders as accurate analytical models which capture superscalar processor designs. These models must be created by an architect with expert knowledge of the system, and it is unclear how they will address the complicated interactions of chip multiprocessor design spaces.

Lee and Brooks [17] perform regression modeling based on cubic splines in order to generate predictive approximation functions for superscalar processor performance and power. Their method is a statistical process rather than an automated algorithm, and requires an *a priori* understanding of variable relationships on the part of the architect [11].

An advantage of GPRS over 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 previous knowledge of the design space. However, this automation means that GPRS is much more computationally intensive than any statistical technique.

Ipek et al. [11] 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 [11]. Like GPRS, ANNs 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.

Kumar et al. attempt to search through a large het-

erogeneous chip multiprocessor design space using a hill–climbing search [16]. Hill–climbing selects a configuration 4.5% worse than the configuration selected by the exhaustive global search, while requiring 86% fewer full simulations. Eyerman et al. [9] perform a similar study in a embedded processor context and investigate the performance of a variety of search algorithms. During the search phase of the process they use statistical simulation [6] to speed up the evaluation of individual design points.

4. RESPONSE SURFACE CREATION

Due to space constraints we assume the reader is familiar with the basic processes of standard genetic search algorithms. Like all evolutionary algorithms, the biological concepts of natural selection and survival of the fittest are at the heart of genetic programming. The only difference is that rather than searching through the space of possible designs directly, our algorithm searches through the space of predictive functions which describe design performance.

The algorithm is structured such that accurate candidate response surfaces are more likely to 'survive' and be recombined, and so subsets of their highly fit structure spread across generations, leading to a rising average fitness and the eventual convergence of the algorithm on an accurate model. Readers seeking more information about genetic programming in general should refer to [15], and those seeking information about the mechanics of genetically programmed response surfaces should refer to [1].

4.1 Modeling Representation

Candidate response surfaces are polynomial equations represented as expression trees, where an individual node in a tree represents a user-defined operator, an encoded design parameter or a tuning parameter (Figure 2). Operators are defined by the user – 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 real–valued design parameters are encoded as floating point values, while boolean variables are converted into 0/1 values. The use of tuning parameters (Section 4.3) 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 [11].

4.2 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. We quantify the quality of the model by calculating the sum of absolute differences between the candidate surface's predictions and the collected performance data for all sample design points (Eq.1). Penalizing lengthy expression trees serves to keep the equations more compact and globally fit, and encourages the process to drop terms which do not improve the quality of predictions. In the fitness function (Eq.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

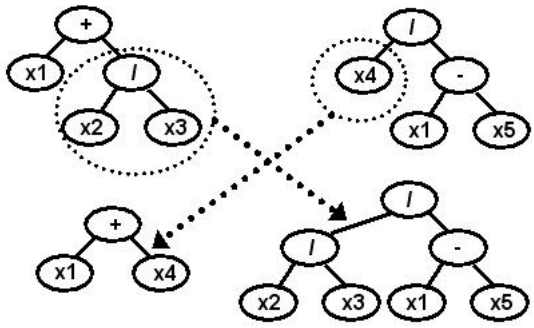

Figure 2: Expression trees undergoing crossover breeding

the function. 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.

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

$$\Phi(S_i) = Q(S_i) - c * ntp_i^2 \rightarrow max \quad (2)$$

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 highly fit candidates having a higher probability of being selected), and the selected candidates then swap randomly determined subtrees (Figure 2). 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).

4.3 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]. 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. 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 [19]. Levenberg–Marquardt is an iterative technique that can be thought of as a combination of steepest descent and the Gauss–Newton method, with the method of convergence changing depending on its proximity to a solution.

4.4 Design of Experiments

We 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]. Random sampling [17] does not guarantee an even sampling distribution, and sampling based on variance [11] does not allow us to collect the complete set of sample data prior to the response surface training process.

5. EXPERIMENTAL METHODOLOGY

We use simulated performance data previously collected by Ipek et al. [11] and Lee et al. [17] for their recent OOE processor 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 (with 2 or 3 parameter bindings each), resulting in over 20K unique design points. Parameters included issue width, frequency, branch predictor sizes, functional unit counts, ROB and register files sizes, and cache sizes. The Lee and Brooks design space has 23 design variables divided into 12 groups (with 2 to 10 parameter bindings each), resulting in approximately one billion design points. The parameter groups include pipeline depth, pipeline width, physical registers and reservation stations, cache sizes, functional unit latencies, and memory latencies

Both sets of performance data were collected from detailed simulations running over a subset of the SPEC CPU 2000 benchmarks. Ipek et al. made their benchmark selections based on metric similarity clustering suggestions provided by Phansalkar et al. [20], and used the MinneSPEC [14] reference inputs [11]. Lee and Brooks created their input traces by sampling the SPEC full reference input set [17].

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 ideal size of the sample set varies depending on the size of design space being studied. 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 simply extract these datapoints out of the previously collected sets of performance data, but in a brand new optimization study we would simulate each of them in full detail.

In selecting the parameters used to control the operation of the GP algorithm, we generally adhere to the 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 expression tree size penalty constant **c** to 0.000001, and use a 10% elite percentage and 10% kill percentage to aid in convergence.

For reasonable values, these algorithmic variables affect the efficiency of the algorithm and the speed at which it converges, and not its ability to arrive at an accurate solution. The size penalty constant is the most important to set appropriately because it can affect whether the models produced over or under fit the sample data. Cross validation can also be used to prevent overfitting when the proper value for this constant is unknown [11].

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 realize a toolchain for taking a set of a benchmark results, a simulator configuration script, and a list of design parameters with their bindings and automatically formulating DOEs and creating GPRSs based on them.

6. EXPERIMENTAL RESULTS

We find that the models generated by GPRS accurately and robustly depict the true behavior of many performance measures, even at extremely small sample sizes. GPRS identifies variables that have a significant impact on the performance measure by explicitly including them in an analytical equation, giving the architect insight into how they should be optimized and which subspaces should be explored.

Our first study makes use of data provided by Ipek et. al [11]. We generate DOEs for this twelve dimensional space and use them to train GPRSs that provide predictions for performance (IPC), branch prediction and L2 cache hit rate. We calculate percentage errors between the predicted values suggested by the response surface equation and the detailed simulation data. We calculate GPRS error across *all* data points collected by Ipek et al. — in this case, they conducted an exhaustive search of the entire processor design space — and find that the GPRS provides highly accurate predictions of design performance on a global scale.

With sample sizes as small as 0.1% of possible designs, predictions of branch predictor accuracy and L2 hit rates have $<$ 2% mean global error. GPRS predictions of IPC for all the non-sampled points achieve 2.8–4.9% mean error with standard deviations between 2.2–3.4%. When sampling increases to 0.5%, 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 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. Figure 3 shows the most and least accurately predicted benchmarks; other CDFs are reported in a technical report. The maximum percentage errors on any one prediction range up to only 15% for even the most poorly predicted benchmarks, with at least 80% of the points having a percentage error smaller than 5%. On the best–predicted

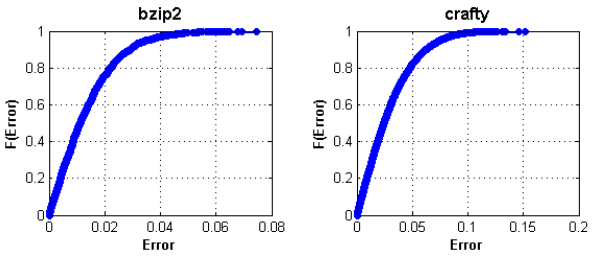

Figure 3: CDFs of error for IPC in the Ipek et al. design space at a 0.5% sample size. The Y-axis represents the percentage of design points with prediction errors < the X-axis value.

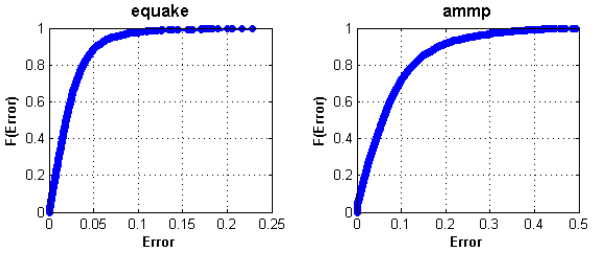

Figure 4: CDFs of error for BIPS in the Lee et al. design space at a 0.000002% sample size

benchmarks over 97% of the points have percentage errors smaller than 5% and the maximum error is only 7.3%.

Genetically programmed response surfaces are explicit approximation functions relating design parameters to performance measures, and we can make use of their explicit nature to gain insight into the target design space. The genetic programming process drops terms from the response surface equation which do not confer any additional predictive accuracy. Over time, parameters strongly correlated with the performance measure are retained, while insignificant variables are discarded. For this reason, an architect examining the response surface generated for a target benchmark can quickly tell which design decisions will actually have a significant impact on performance.

As expected, the GPRSs created for the Ipek et al. data identified different parameters as significant when asked to provide predictions for different performance measures. For the cache and branch prediction performance measures the cache sizes and branch predictor sizes were the only significant variables for all benchmarks. For IPC predictions, many parameters had a significant role in determining performance, and these parameters were different for surfaces trained on different benchmark applications. This makes intuitive sense — different architectural characteristics will have different performance impacts on memory-bound applications as compared to CPU-bound ones, for example.

The response surface makes it easy to predict exactly what the performance impact of a given design decision will be, even in the face of complicated multivariate interactions. None of the ratios or variable relationships need be specificied in advance — the genetic programming algorithm is wholly responsible for automatically identifying the relevant variables and their relationships.

Numerical optimization with off-the-shelf software can rapidly determine the optimal values for significant variables, and an architect could search through the remaining variables to find the precise optimal configuration. However, we find that even just assigning the remaining insignificant design variables to be random valid values will result in near optimal designs (see Table 1). Simply rapidly evaluating every possible point using the response surface model is also a tractable solution in some cases.

benchmark	(1) true	(2) pred.	(3) worst
applu	2.25	2.235	2.19
art	0.53	0.537	0.51
bzip2	1.48	1.433	1.42
crafty	1.76	1.744	1.62
equake	1.66	1.627	1.61
gcc	1.29	1.269	1.15
mcf	0.58	0.559	0.54
mesa	3.04	2.909	2.87
mgrid	1.73	1.723	1.41
swim	0.95	0.936	0.89
twolf	1.01	0.979	0.93
vortex	2.48	2.362	2.31

Table 1: Summary of GRPS results after training on Ipek et al.'s processor performance data with sample size of 0.5%. (1) True global optima in IPC as determined by exhaustive search. (2) Optimal IPC value predicted by GRPS. (3) Worst possible design's IPC after significant variables are analytically optimized.

Our second study made use of performance and power data collected from a superscalar processor design space by Lee and Brooks [17] to test their regression modeling technique for performance prediction. 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 our more robust sampling method. Figure 4 uses cumulative density functions of percentage error to illustrate the performance of GPRS on this design space.

At sample sizes as low as 200 out of over 1 billion design points per benchmark, we find that the median percentage error of the performance (BIPS) 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. For power (watts), the median percentage errors are at worst 6.2% with a variance of 5.7% and a maximum of 32.5%. The Lee and Brooks dataset does not include an exhaustive search of the design space, so we have evaluated GPRS error based on 1000 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.

Our success at making predictive models of the Lee and Brooks design space demonstrates that accurate predictions based on several orders of magnitude fewer simulations are certainly feasible when the design space is large enough. Simulations used to generate our model could afford to be 20 times more detailed than those used by Lee and Brooks, or 5 million times more detailed than those used in an exhaustive search, assuming we are willing to spend the same amount of time on each study.

We find that in practice the time required to create a

highly accurate GPRS is several hours, and for a population size of 500 running completely unparallelized every additional 100 sample points adds about two hours to convergence time. Blackbox neural networks can be created in only minutes, whereas regression analysis is not automated. However, due to our use of a formal DOE and the predictive interpolation provided by our response surface, sample size increases only linearly with each dimension added to the problem. At high dimensionality the time spent creating and evaluating a GPRS is significantly smaller than the time required to complete the larger set of full–scale simulations required for an exhaustive study, heuristic search, or predictive technique dependent on larger sample sizes.

7. CONCLUSIONS

Architects making use of genetically programmed response surfaces can anticipate vast reductions in simulation overhead and automated identification of important design parameters and subspaces, all while maintaining a high degree of confidence in the predictive results. GPRS provides architects with predictive polynomial equations that can be trivially solved to estimate what a costly cycle-accurate simulation would otherwise have produced, enabling an analysis of the performance impacts of proposed design changes without requiring additional detailed simulation data. Time saved by using GPRS allows the few detailed simulations required for training to be even more detailed, thus improving the overall efficiency and realism of the process.

Unlike the heuristic design space searches performed in [16] and [9], 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. Predictive techniques that are scalable and automated will become increasingly fundamental to producing effective computer architecture optimizations as the trend towards greater design complexity continues.

8. ACKNOWLEDGMENTS

This work was funded in part by a research grant from NSF, nos. CCR-0133634, CCF-0429765, and IIS-0612049, and formed the basis for Cook's undergraduate senior thesis in the Univ. of Virginia School of Engineering and Applied Science. We would like to thank John Lach, Paul Reynolds and the anonymous reviewers for their helpful comments, and the research groups of Sally McKee (Cornell) and David Brooks (Harvard) for access to their datasets.

9. REFERENCES

[1] L. Alvarez. *Design Optimization based on Genetic Programming*. PhD thesis, Univ. 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, 1977. In Russian.

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

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

[5] J. Davis, J. Laudon, and K. Olukotun. Maximizing cmp throughput with mediocre cores. In *PACT*, Oct. 2005.

[6] 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.

[7] L. Eeckhout, J. Sampson, and B. Calder. Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation. In *IISWC*, Oct. 2005.

[8] J. Emer and N. Gloy. A language for describing predictors and its application to automatic synthesis. In *ISCA*, Jun. 1997.

[9] S. Eyerman, L. Eeckhout, and K. D. Bosschere. Efficient design space exploration of high performance embedded out-of-order processors. In *DATE*, Mar. 2005.

[10] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith. A performance counter architecture for computing accurate cpi components. In *ASPLOS*, Oct. 2006.

[11] E. Ipek, S. McKee, B. de Supinski, M. Schulz, and R. Caruana. Efficiently exploring architectural design spaces via predictive modeling. In *ASPLOS*, Oct. 2006.

[12] P. Joseph, K. Vaswani, and M. J. Thazhuthaveetil. Construction and use of linear regression models or processor performance analysis. In *HPCA*, Feb. 2006.

[13] T. Karkhanis and J. E. Smith. A first-order superscalar processor model. In *ISCA*, 2004.

[14] A. KleinOsowski and D. Lilja. Minnespec: A new spec benchmark workload for simulation-based computer architecture research. *IEEE Comp. Arch. Letters*, 1(1), 2006.

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

[16] R. Kumar, D. M. Tullsen, and N. P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *PACT*, Sep. 2006.

[17] B. Lee and D. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ASPLOS*, Oct. 2006.

[18] Y. Li, B. C. Lee, D. Brooks, Z. Hu, and K. Skadron. Cmp design space exploration subject to physical constraints. In *HPCA*, Feb. 2006.

[19] M. Lourakis. levmar: Levenberg-marquardt nonlinear least squares algorithms in C/C++. http://www.ics.forth.gr/ lourakis/levmar/, 2004.

[20] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John. Measuring program similarity: Experiments with spec cpu benchmark suites. In *ISPASS*, Mar. 2005.

[21] G. Venter, R. Haftka, and J. Starnes. Construction of response surfaces for design optimization applications. In *6th Symp. on Mult. Anal. and Opt.*, 1996.

[22] J. Yi, D. Lilja, and D. Hawkins. A statistically rigorous approach for improving simulation methodology. In *HPCA*, Feb. 2003.