Evolutionary system for prediction and optimization of hardware architecture performance

P. A. Castillo[1] J.J. Merelo[1] M. Moreto[2] F. J. Cazorla[3] M. Valero[2,3] A.M. Mora[1] J.L.J. Laredo[1] S.A. McKee[4]

***Abstract*— The design of computer architectures is a very complex problem. The multiple parameters make the number of possible combinations extremely high. Many researchers have used simulation, although it is a slow solution since evaluating a single point of the search space can take hours. In this work we propose using evolutionary multilayer perceptron (MLP) to compute the performance of an architecture parameter settings. Instead of exploring the search space, simulating many configurations, our method randomly selects some architecture configurations; those are simulated to obtain their performance, and then an artificial neural network is trained to predict the remaining configurations performance. Results obtained show a high accuracy of the estimations using a simple method to select the configurations we have to simulate to optimize the MLP. In order to explore the search space, we have designed a genetic algorithm that uses the MLP as fitness function to find the niche where the best architecture configurations (those with higher performance) are located. Our models need only a small fraction of the design space, obtaining small errors and reducing required simulation by two orders of magnitude.**

I. Introduction

Designing a computer architecture needs a huge number of parameters to be calibrated. Each parameter can take different values which could impact in the architecture performance.

Usually, simulation techniques are used to evaluate different settings, searching for either the best combination of values or a promising niche within the search space. Although the improvement in simulators, search space size makes simulation times too high [1]. Even small search spaces can be impracticable when simulating [2], [3], [4]. That is why using a system that preditcs performance without actually running the simulator would save a lot of time in researching new hardware configurations, giving a range or a set of parameters that can then be simulated for an effective test of performance.

This paper extends Ipek's work [1], who proposed using artificial neural networks (ANN) for architecture performance (instructions per cicle, IPC) prediction. In order to optimize the ANN the training and validation patterns are sampled using *Active learning* [5].

In this paper we intend to simplify the sampling method of the parameter space, using random selection. We propose to focus the effort on the ANN optimization using GProp [6], [7], [8], [9], an evolutionary method for the design and optimization of neural networks.

[1]Department of Architecture and Computer Technology, ETSIIT, University of Granada. [2]Computer Architecture Department, HiPEAC European Network of Excellence, Technical University of Catalonia. [3]Barcelona Supercomputing Center, Technical University of Catalonia. [4]Cornell University. Contact email: `pedro@atc.ugr.es`

The experimentation process consists in randomly selecting a small fraction of the search space configurations. Those simulated points are used to train the MLP, which is used afterwards to predict the rest of architecture configuration performance. Figure 1 summarizes our modeling mechanism using random sampling to optimize an MLP and Figure 2 provides a different perspective on the steps in training versus using the model. Since running the set of parameters through the MLP is faster than making a simulation with the same parameters, once the best MLP is found, we use it as the fitness function of an evolutionary algorithm (EA) to search those configurations with higher IPC. Furthermore, once the configurations with best IPC are found, the designer can focus the study on that zone of the search space.

The rest of this paper is structured as follows: In section II related work is analysed. Section III describes the problem of exploring architectural design spaces. In section IV the GProp algorithm is introduced. Section V describes the experiments and presents the results obtained, followed by a brief conclusion in section VI.

II. Related Work

There are some recent works tackling the computer architecture design problem, mainly under two approaches: analytic and simulation methods.

Within the analytic approaches, Karkhanis and Smith [10] proposed a super-scalar microprocessor model which yields $87-95\%$ of accuracy in estimations. This model is further improved in a recent publication [11]. Yi et al. [12] studied parameter priority using fractional factorial design. By focusing on the most important parameters, the number of simulations required to explore a large design space can be reduced.

Other researchers (Chow and Ding [13] and Cai et al. [14]) proposed using principal components analysis to identify the most important parameters and their correlations for processor design. Eeckhout et al. [15] and Phansalkar et al. [16] used similar methods for workload and benchmark composition.

Muttreja et al. [17] developed high-level models to estimate performance and energy consumption. They simulated several embedded benchmarks with 1.3% error. Lee and Brooks [18] used regression for predicting performance and power consumption. However, their approach is not easy to apply and it requires some statistical knowledge.

The alternative to analytic methods is simulation [4]. Oskin et al. [19] developed a hybrid simulator to model instruction and data streams, Rapaca et al. [20] used another hybrid

978-1-4244-1823-7/08/$25.00©2008 IEEE

1. Identify important parameters
2. Perform a set of simulations for N random combinations of parameter settings
3. Normalize inputs and outputs. Encode nominal parameters, booleans as 0-1, and others as real values in the normalized [0-1] range. Collect the results in a data set
4. Divide data set
5. Use GProp to design an MLP to approximate patterns. During training, present each data point in the training set to the MLP. Obtain the fitness of each individual using the validation set
6. Once GProp has finished, predict any point in the parameter space by placing the parameters at the input layers of the MLP obtained

Fig. 1. Summary of steps in the modeling mechanism. Some configurations are randomly chosen and evaluated in the simulator. Then an MLP is trained to predict the configuration IPC. Finally, that MLP is used to approximate other configurations.

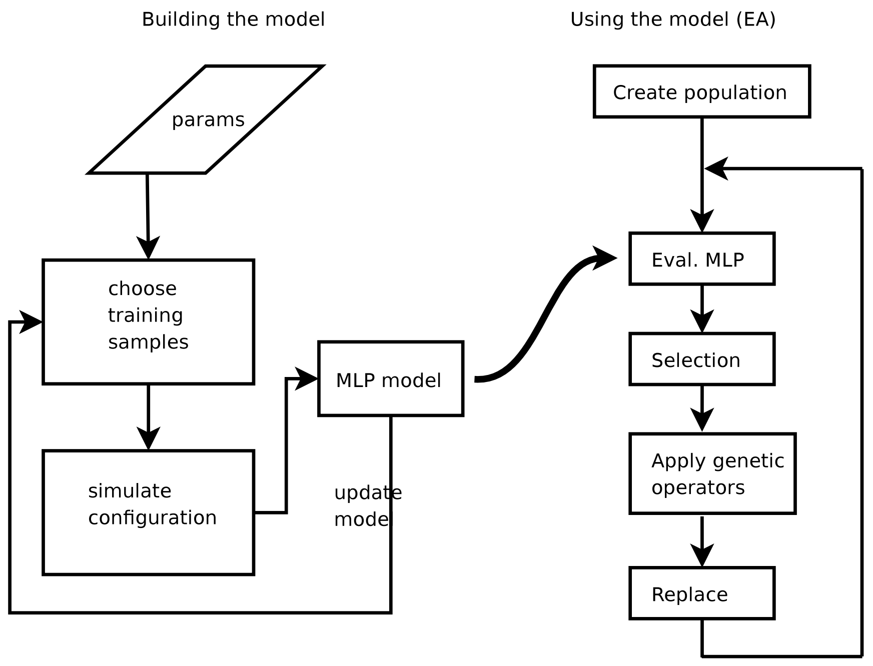

Fig. 2. Pictorial representation of modeling and explotation mechanisms. On the left main steps in modeling mechanism are shown. On the right, the scheme of the EA that searches for the best configurations is shown.

simulator and instructions code to infer information that is used to estimate statistics for other application code. Other authors, such as Wunderlich et al. [21] modeled minimal instruction stream to achieve results within desired confidence intervals. Haskins and Skadron [22] sampled application code to create a cache and branch predictor state.

Ipek et al. [1] developed accurate predictive design-space models simulating sampled points and using the results to train an ANN. Their methods yielded a high accuracy but the design space sampling method is rather complex. Lee et al. [23] also propose roughly the same technique: sampling a small part of the design space, and infer performance for the rest of the space using statistical techniques. They analyze several sampling techniques and inference methods, and test them on a superscalar processor design.

In this work, we intend to simplify the sampling method (using a random selection method that simulates less architecture configurations) and to improve performance approximation results using an evolutionary method for ANN design.

III. The Problem

Computer architects have to deal with several types of parameters that define a design: quantitative parameters (i.e. cache size), selections (i.e. cache associativity), numerical values (i.e. frequency) and logic values (i.e. core configuration). The encoding and the way these values are used to train and to exploit an ANN can influence the model accuracy.

In this work, we use the benchmark suite SPEC CPU 2000 [24] which is composed by a wide range of applications. Following prior work [1], we use bzip2, crafty, gcc, mcf, vortex, twolf, art, mgrid, applu, mesa, equake and swim. They cover a wide spectrum of the total set of benchmarking programs. We tackle the design of the memory system and CPU. These are defined by a set of parameters.

TABLE I

PARAMETER VALUES IN MEMORY SYSTEM STUDY.

Variable Parameters	Values
L1 DCache Size	8, 16, 32, 64 KB
L1 DCache Block Size	32, 64 B
L1 DCache Associativity	1, 2, 4, 8 Way
L1 Write Policy	WT, WB
L2 Cache Size	256, 512, 1024, 2048 KB
L2 Cache Block Size	64, 128 B
L2 Cache Associativity	1, 2, 4, 8, 16 Way
L2 Bus Width	8, 16, 32 B
Front Side Bus Frequency	0.533, 0.18, 1.4 GHz
Fixed Parameters	**Value**
Frequency	4 GHz
Fetch/Issue/Commit Width	4
LD/ST Units	2/2
ROB Size	128 Entries
Register File	96 Integer / 96 FP
LSQ Entries	48/48
SDRAM 100 ns	64 bit FSB
L1 ICache	32 KB / 2 Cycles
Branch Predictor	Tournament (21264)

TABLE II

PARAMETER VALUES IN THE PROCESSOR STUDY.

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

Table I shows parameters in the memory hierarchy study. Core frequency is 4GHz. The L2 bus runs at core frequency and the front-side bus is 64 bits. The cross product of all parameter values would in principle require 23040 simulations per benchmark.

Table II shows parameters in the microprocessor study. We use core frequencies of 2GHz and 4GHz, and calculate cache and SDRAM latencies and branch misprediction penalties based on these. We use 11- and 20-cycle minimum latencies for branch misprediction penalties in the 2GHz and 4GHz cases, respectively. For register files, we choose two of the four sizes in Table 2 based on ROB size (e.g., a 96 entry ROB makes little sense with 112 integer/fp registers). When choosing the number of functional units, we choose two sizes from Table II based on issue width. The number of load, store and branch units is the same as the number of floating point units. SDRAM latency is 100ns, and we simulate a 64-bit front-side bus at 800MHz. Taking into account these parameters and their values, the microprocessor study requires 20736 simulations per benchmark.

IV. The Method

We propose using GProp, an algorithm that evolves an MLP population. This method searches for the best network structure and initial weights, while minimizing the error rate. It makes use of the capabilities of two types of algorithms: the ability of EA [25], [26], [27] to find a solution close to the global optimum, and the ability of the quick-propagation algorithm [28] to tune it and to reach the nearest local minimum by means of local search from the solution found by the EA.

The complete description of the method and the results obtained using classification problems have been presented elsewhere [6], [7], [8], [9]. The designed method uses an elitist algorithm [29].

The representation ability of a neural network depends on the number of layers, on the number of neurons per layer and on the connectivity between layers. It was demonstrated that a network with two hidden layers can solve any pattern classification problem [30], [31], [32]. On the other hand, several authors have proved that any function approximation problem can be solved by using one hidden layer [33], [34], [35], [36].

An EA requires that each individual is encoded as a chromosome for it to be handled by the genetic operators of the EA. Some authors use binary or real encoding (representation of the networks in a binary or real number string) [37], [38], or indirect coding [39], [40], but G-Prop evolves the initial parameters of the network (initial weights and learning constants) using specific genetic operators. At the lowest level, an MLP is an object instantiated from the MLP C++ class. The data structure of this class is an array of vectors of neurons, where each neuron is a vector of weights. However, the EA does not use binary strings, but MLP objects and neurons.

We used Evolutionary Objects (EO) library [41], because of the facility that this library offers to evolve any object with a fitness function. It is a C++ toolbox which defines interfaces for many classes of algorithms used in evolutionary computation and, at the same time, provides some examples that use those interfaces. It is available at `http://geneura.ugr.es/~jmerelo/EO.html`.

The genetic operators act directly upon the ANN object (instead of performing hierarchical evolution at the neuron level [42]), but only *initial weights* and the *learning constant* are subject to evolution, not the weights obtained after training. In order to calculate the fitness, a clone of the MLP is created, and thus, the initial weights remain unchanged in the original MLP. Only when the training operator is used, changes are saved back into the individual genetic code that remains in the population.

When a genetic operator changes an MLP, it considers each hidden neuron (and its input and output weights) as a gene, so that if two MLPs are crossed, complete hidden layer neurons are interchanged (and weights to and from it are treated as one unit) [43], [44], [45].

Five variation operators are used to change MLPs: mutation, crossover, addition and elimination of hidden units, and quick-propagation training applied as operator.

The fitness function of an individual (MLP) is given by the mean squared error obtained on the validation process that follows training. In the case of two individuals showing an identical classification error, the one with the hidden layer containing the least number of neurons would be considered the best (the aim being small networks with a high generalization ability).

To present the data to the MLP, cardinal and continuous parameters are encoded as a real number in the [0,1] range, normalizing with minimax scaling via minimum and maximum values over the design space. For nominal parameters we allocate an input unit for each parameter setting, making the input corresponding to the desired setting 1 and those corresponding to other settings 0. Boolean parameters are represented as single inputs with 0/1 values. Target value (IPC) for model training is encoded in the same way as inputs. Normalized IPC predictions are scaled back to the actual range. Following the method presented in [1], when reporting error rates, we perform calculations based on values that are not normalized.

V. Experiments and Results

The following experiments have been carried out: We have searched and optimized an MLP to predict the IPC values for the Memory System and CPU problems. The MLP is trained on 1% of the total points (architecture configurations), and afterwards it predicts the IPC values for the whole design space. We choose this percentage as proposed in [1].

Then, the best configuration for each one of the benchmarking applications (either for Memory System and CPU problems) is found and the best MLP is used to predict the IPC for those architecture settings.

As a final experiment, as exploring the whole search space can be a costly problem, we propose to use an EA to find

TABLE III

MEAN SQUARED ERROR AND STANDARD DEVIATION FOR THE MEMORY SYSTEM (A) AND THE CPU (B) PROBLEMS. ONLY A 1% OF THE DESIGN SPACE HAS BEEN SIMULATED TO TRAIN THE MLPS. THE TABLE SHOWS THE RESULTS OBTAINED BY IPEK ET AL. [1] AND WITH THE GPROP METHOD.

Application	Ipek et al.	GProp
applu	3.11 ± 2.74	4.27 ± 1.08
art	6.63 ± 5.23	4.11 ± 0.45
bzip2	1.95 ± 1.84	1.62 ± 0.08
crafty	2.16 ± 2.10	2.96 ± 0.47
equake	2.32 ± 3.28	2.42 ± 0.35
gcc	3.69 ± 4.02	1.77 ± 0.16
mcf	4.61 ± 5.60	1.46 ± 0.10
mesa	2.85 ± 4.27	13.75 ± 4.22
mgrid	4.96 ± 6.12	4.34 ± 2.47
swim	0.66 ± 0.52	0.83 ± 0.11
twolf	4.13 ± 6.23	1.52 ± 0.22
vortex	5.53 ± 4.63	8.91 ± 0.59

(a) Memory system study

Application	Ipek et al.	GProp
applu	1.94 ± 1.45	4.83 ± 0.64
art	2.41 ± 1.91	1.09 ± 0.19
bzip2	1.30 ± 0.95	2.25 ± 0.23
crafty	2.65 ± 2.03	4.21 ± 0.50
equake	1.80 ± 1.39	3.03 ± 0.42
gcc	1.88 ± 1.48	2.39 ± 0.24
mcf	1.67 ± 1.38	1.05 ± 0.17
mesa	2.57 ± 1.96	8.38 ± 1.28
mgrid	1.39 ± 1.13	3.08 ± 0.58
swim	2.65 ± 2.05	1.72 ± 0.28
twolf	4.85 ± 4.76	1.32 ± 0.17
vortex	2.90 ± 2.17	6.01 ± 1.36

(b) CPU study

TABLE IV

BEST SIMULATED CONFIGURATION AND THE PREDICTION OBTAINED USING GPROP FOR THE MEMORY SYSTEM (A) AND THE CPU (B) PROBLEMS. FIRST COLUMN SHOW THE BENCHMARKING APPLICATIONS, THE SECOND ONE THE IPC OF THE BEST CONFIGURATION AFTER SIMULATING THE WHOLE SEARCH SPACE. THE THIRD COLUMN SHOWS THE PREDICTION OBTAINED USING GPROP FOR THAT CONFIGURATION (MEAN SQUARED ERROR AND STANDARD DEVIATION).

Application	IPC Best Simulated Configuration	IPC GProp Predicted Configuration
applu	1.79	1.74 ± 0.01
art	1.56	1.48 ± 0.01
bzip2	1.10	1.077 ± 0.002
crafty	1.33	1.29 ± 0.01
equake	1.17	1.15 ± 0.01
gcc	1.05	1.036 ± 0.003
mcf	0.47	0.444 ± 0.004
mesa	1.82	1.81 ± 0.01
mgrid	1.55	1.52 ± 0.02
swim	0.77	0.755 ± 0.002
twolf	0.90	0.889 ± 0.001
vortex	1.71	1.67 ± 0.01

(a) Memory system study

Application	IPC Best Simulated Configuration	IPC GProp Predicted Configuration
applu	2.25	2.15 ± 0.03
art	0.53	0.502 ± 0.001
bzip2	1.48	1.40 ± 0.03
crafty	1.76	1.65 ± 0.02
equake	1.66	1.56 ± 0.01
gcc	1.29	1.20 ± 0.01
mcf	0.58	0.54 ± 0.01
mesa	3.04	2.88 ± 0.08
mgrid	1.73	1.68 ± 0.02
swim	0.95	0.917 ± 0.004
twolf	1.01	0.97 ± 0.01
vortex	2.48	2.29 ± 0.07

(b) CPU study

the zone of the search space where the configurations with higher IPC are located.

We conducted our experiments on a bi-processor AMD AthlonXP with 1.66GHz and 1GB RAM. The evolutionary method and the later exploitation of the obtained MLPs consume about nine minutes, while the phase of approaching the whole design space takes less than a second.

Tables III (a) and (b) show the results obtained training an MLP using intelligent sampling [1] and those obtained using GProp with random sampling after 30 independent runs (mean squared error and standard deviation are reported).

Although GProp trains the MLP with a random 1% from the whole possible configurations, results are comparable and even better than those obtained using *Active Learning* for pattern sampling. Furthermore, GProp shows its robustnes with the low standard deviations reported versus those reported in [1] (Ipek column in the table).

Tables IV (a) and (b) show the best simulated configuration IPC and the prediction obtained using GProp for that configuration. The MLP yields a good prediction concerning the IPC value for the best setting (obtained by simulation). Furthermore, we observe from experimentation that MLP predicts the best settings within the same niche in the design space. In this experiment, Ipek et al. [1] only report the value

TABLE V

BEST SIMULATED CONFIGURATION AND THE OBTAINED USING THE EA FOR THE MEMORY SYSTEM (A) AND THE CPU (B) PROBLEMS. FIRST COLUMN SHOW THE BENCHMARKING APPLICATIONS, THE SECOND ONE THE CONFIGURATION PARAMETER VALUES AND THE THIRD COLUMN SHOWS THE CONFIGURATION OBTAINED USING THE EA.

Application	Best Simulated Configuration	EA Obtained Configuration
applu	16 32 1 WB 1024 128 16 16 0.533	64 64 1 WB 2048 128 16 16 1.4
art	16 64 1 WB 2048 128 16 32 0.533	64 32 1 WB 2048 64 16 32 1.4
bzip2	16 64 2 WB 1024 128 16 16 0.533	64 32 1 WB 2048 128 2 16 0.8
crafty	64 64 4 WB 2048 128 16 16 0.533	32 32 1 WB 2048 64 16 32 0.533
equake	16 32 2 WB 2048 128 16 16 1.4	64 64 8 WB 2048 128 16 32 1.4
gcc	16 32 1 WB 512 128 8 16 0.533	8 32 4 WT 512 128 4 32 0.533
mcf	64 64 2 WB 1024 128 16 16 1.4	64 64 1 WB 2048 128 16 32 1.4
mesa	16 32 1 WB 1024 128 16 16 0.533	8 32 1 WB 256 128 4 16 0.8
mgrid	16 64 1 WB 2048 128 16 16 0.533	64 64 1 WB 2048 128 16 8 0.533
swim	16 64 1 WB 2048 128 4 16 1.4	8 64 4 WT 1024 128 16 8 1.4
twolf	64 64 2 WB 256 128 4 16 0.533	16 32 8 WB 256 64 8 16 0.533
vortex	64 64 2 WB 1024 128 16 16 1.4	8 64 1 WB 1024 64 16 16 0.533

(a) Memory system study

Application	Best Simulated Configuration	EA Obtained Configuration
applu	8 2 32 1 2 8 160 112 64 8 32 1024	8 2 8 1 1 4 160 64 64 8 8 1024
art	8 2 32 1 1 8 160 112 64 32 32 1024	4 2 8 4 1 8 160 64 48 32 8 1024
bzip2	8 2 32 4 1 8 160 112 64 8 32 1024	8 2 8 1 1 4 96 112 32 8 8 1024
crafty	8 2 32 4 2 8 160 112 64 8 32 1024	8 2 8 1 2 4 160 64 32 8 32 1024
equake	8 2 32 4 1 8 160 112 64 32 32 1024	4 2 8 4 1 4 96 64 32 8 8 1024
gcc	8 2 32 4 2 8 160 112 64 32 32 1024	4 4 32 4 2 8 96 112 32 8 32 1024
mcf	6 2 32 2 1 8 160 112 64 8 32 1024	4 2 8 1 2 8 128 64 32 32 32 1024
mesa	8 2 32 2 1 8 160 112 64 32 8 1024	4 2 8 1 1 8 96 64 64 32 8 1024
mgrid	8 2 16 2 2 8 160 112 64 8 8 1024	4 2 8 1 1 4 96 64 32 8 8 1024
swim	8 2 16 4 2 8 160 112 64 8 8 1024	4 2 8 4 1 4 96 64 48 32 32 1024
twolf	8 2 32 4 2 4 160 112 64 8 32 256	4 2 8 1 1 4 160 64 64 8 8 256
vortex	8 2 32 4 2 8 160 112 64 32 32 1024	8 2 32 4 2 8 128 112 64 32 32 1024

(b) CPU study

for the Memory system problem in the bzip2 application. The best setting yields an IPC of 1.09, very close to the optimum and to the value obtained using GProp.

Finally, once the best MLP is found, we have used it to guide the search of an EA that tunes the parameter settings to find the niche in the design space where the best configurations are located. For each problem, the EA is quick at finding a good solution close to the IPC optimum for that problem. Table V shows the best simulated configuration parameter values and the obtained using the EA for the Memory System and the CPU problems.

VI. CONCLUSIONS AND FUTURE WORK

This work tackles the computer architecture design using the benchmark problems proposed in [1]. We have shown how an ANN can shape a wide search space from the knowledge of a small and random portion. Thus, the experiments use just a randomly chosen 1% of all the possible design settings; this implies that by randomly choosing 1% of possible parameter settings to simulate, we can obtain a good representation of the architecture performance function.

We have proposed using GProp, a method that evolves an MLP population to obtain a model that predicts the IPC value. The designed MLP predicts any architecture parameter configuration performance with a small error rate.

Furthermore, the proposed method uses a simple random pattern sampling mechanism for the training set. Results obtained are comparable to those presented by other authors, with a low standard deviation (algorithm robustness) as an improvement over them.

We have demonstrated that randomly selecting a small configurations set, it is possible to make accurate predictions. Moreover, our proposal is able to explore a wide search space far from the capabilities of current simulation methods.

We have developed an EA to automatically explore the search space to find the niche where the best configuration settings are located.

As future work, we could use a trained MLP to automatically suggest new points to simulate, so that the overall performance prediction error is minimized. Finally, we intend to tackle the design of real-world devices applying this technique, using the MLP for performance estimation instead of having to simulate the whole parameter space.

ACKNOWLEDGEMENTS

This work has been supported by the Spanish CICYT projects TIN2007-68083-C02-01, TIN2004-07739, TIN2007-60625 and grant AP-2005-3318, the Junta de Andalucia CICE project P06-TIC-02025 and the Granada University PIUGR 9/11/06 project.

REFERENCES

[1] E. Ipek, S. A. McKee, B. R. de Supinski, M. Schulz, and R. Caruana, “Efficiently Exploring Architectural Design Spaces via Predictive Modeling,” *In Proc. ASPLOS2006, pp. 195-206*, 2006.

[2] M. Martonosi and K. Skadron, “NSF computer performance evaluation workshop,” *http://www.princeton.edu/mrm/nsf_sim_final.pdf*, 2001.

[3] B. Jacob, “A case for studying DRAM issues at the system level,” *IEEE Micro, 23(4):44-56*, 2003.

[4] J. Davis, J. Laudon, and K. Olukotun, “Maximizing CMP throughput with mediocre cores,” *In Proc. PACT2005, pp. 51-62*, 2005.

[5] M. SaarTsechansky and F. Provost, “Active learning for class probability estimation and ranking,” *In Proc. IJCAI2001, pp. 911-920*, 2001.

[6] P. A. Castillo, J. Carpio, J. J. Merelo, V. Rivas, G. Romero, and A. Prieto, “Evolving Multilayer Perceptrons,” *Neural Processing Letters, vol. 12, no. 2, pp.115-127. October*, 2000.

[7] P. A. Castillo, J. J. Merelo, V. Rivas, G. Romero, and A. Prieto, “G-Prop: Global Optimization of Multilayer Perceptrons using GAs,” *Neurocomputing, Vol.35/1-4, pp.149-163*, 2000.

[8] P. Castillo, M. Arenas, J. J. Merelo, V. Rivas, and G. Romero, “Optimisation of Multilayer Perceptrons Using a Distributed Evolutionary Algorithm with SOAP,” *Lecture Notes in Computer Science, Vol.2439, pp.676-685, Springer-Verlag*, 2002.

[9] P. Castillo, J. Merelo, G. Romero, A. Prieto, and I. Rojas, “Statistical Analysis of the Parameters of a Neuro-Genetic Algorithm,” *IEEE Transactions on Neural Networks, vol.13, no.6, pp.1374-1394, ISSN:1045-9227, november*, 2002.

[10] T. Karkhanis and J. Smith, “A 1st-order superscalar processor model,” *In Proc. ISCA2004, pp.338-349*, 2004.

[11] T. S. Karkhanis and J. E. Smith, “Automated design of application specific superscalar processors: An analytical approach,” 2007, pp. 402–411.

[12] J. Yi, D. Lilja, and D. Hawkins, “A statistically-rigorous approach for improving simulation methodology,” *In Proc. HPCA2003, pp.281-291*, 2003.

[13] K. Chow and J. Ding, “Multivariate analysis of Pentium Pro processor,” *In Proc. of Intel Software Developers Conference Track 1, pp. 84-104, October 27-29, Portland, Oregon, USA*, 1997.

[14] G. Cai, K. Chow, T. Nakanishi, J. Hall, and M. Barany, “Multivariate prower/performance analysis for high performance mobile microprocessor design,” *In Power Driven Microarchitecture Workshop (ISCA 1998), Barcelona*, 1998.

[15] L. Eeckhout, R. Bell-Jr, B. Stougie, K. De1Bosschere, and L. John, “Control flow modeling in statistical simulation for accurate and efficient processor design studies,” *In Proc. ISCA2004, pp.350-360*, 2004.

[16] A.Phansalkar, A. Josi, L. Eeckhout, and L. John, “Measuring program similarity: Experiments with SPEC CPU benchmark suites,” *In Proc. ISPASS2005, pp.10-20*, 2005.

[17] A. Muttreja, A. Raghunathan, S. Ravi, and N. Jha, “Automated energy/performance macromodeling of embedded software,” *In Proc. DAC2004, pp.99-102*, 2004.

[18] B. Lee and D. Brooks, “Accurate and efficient regression modeling for microarchitectural performance and power prediction,” *In Proc. ASPLOS-2006, pp.185-194, ISBN:1-59593-451-0*, 2006.

[19] M. Oskin, F. Chong, and M. Farrens, “HLS: Combining statistical and symbolic simulation to guide microprocessor design,” *In Proc. ISCA2000, pp.71-82, ISBN: 1-58113-232-8*, 2000.

[20] V. Rapaka and D. Marculescu, “Pre-characterization free, efficient power/performance analysis of embedded and general purpose software applications,” *In Proc. DATE2003, pp.10504-10509*, 2003.

[21] R. Wunderlich, T. Wenish, B. Falsafi, and J. Hoe, “SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling,” *In Proc. ISCA2003, pp.84-95 isbn:0-7695-1945-8*, 2003.

[22] J. Haskins and K. Skadron, “Minimal subset evaluation: Rapid warm-up for simulated hardware state,” *In Proc. CDES2001, p.32, ISSN:1063-6404*, 2001.

[23] D. M. Lee, Benjamin C.; Brooks, “Spatial sampling and regression strategies,” *IEEE Micro*, vol. 27, no. 3, pp. 74–93, May-June 2007.

[24] SPEC, “Standard Performance Evaluation Corporation. SPEC CPU benchmark suite,” *http://specbench.org/osg/cpu2000*, 2000.

[25] D. Goldberg, “Zen and the art of genetic algorithms,” *In Procs. ICGA95, pp.80-85*, 1995.

[26] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs, Third, Extended Edition*. Springer-Verlag, 1996.

[27] A. Eiben and J. Smith, “Introduction to Evolutionary Computing,” *Springer, ISBN 3-540-40184-9*, 2003.

[28] S. Fahlman, “Faster-Learning Variations on Back-Propagation: An Empirical Study,” *Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann*, 1988.

[29] D. Whitley, *The GENITOR Algorithm and Selection Presure: Why rank-based allocation of reproductive trials is best*. in J.D. Schaffer (Ed.), Procc of The 3th Int. Conf. on Genetic Algorithms, Morgan Kauffmann, 116-121, 1989.

[30] R. Lippmann, “An introduction to computing with neural nets,” *IEEE ASSP Magazine, 3(4):4-22*, 1987.

[31] C. Bishop, *Neural Networks for Pattern Recognition*. Clarendon Press. Oxford University Press Inc., New York., 1996.

[32] R. Reed and R. M. II, *Neural Smithing*. Bradford. The MIT Press, Cambridge, Massachusetts, London, England., 1999.

[33] A. Kolmogorov, “On the representation of continuous functions of several variables by superpositions of continuous functions of one variable and addition,” *American Mathematical Society Translations 28:55-59*, 1963.

[34] A. Gallant and H. White, “There exists a neural network that does not make avoidable mistakes,” *In Proc. IJCNN88, vol 1, pp.657-644*, 1988.

[35] K. Funahashi, “On the approximate realization of continuous mappings by neural networks,” *Neural Networks 2(3):183-192*, 1989.

[36] G. Cybenko, “Approximation by superpositions of sigmoids,” *Mathematics of Control, Signals, and Systems 2:303-314*, 1989.

[37] I. de Falco, A. Iazzetta, P. Natale, and E. Tarantino, “Evolutionary Neural Networks for Nonlinear Dynamics Modeling,” *Lectures Notes in Computer Science, vol. 1498, 593-602*, 1998.

[38] P. Durr, C. Mattiussi, and D. Floreano, “Neuroevolution with Analog Genetic Encoding,” *Lecture Notes in Computer Science, vol.4193, pp.671-680. ISBN:978-3-540-38990-3*, 2006.

[39] A. Cangelosi, D. Parisi, and S. Nolfi, “Cell Division and Migration in a Genotype for Neural Networks,” *Network: Computation in Neural Systems, vol.5, pp.497-515*, 1994.

[40] F. Gruau, “Neural network synthesis using cellular encoding and the genetic algorithm,” *PhD thesis, Ecole Normale de Lyon, France*, 1994.

[41] J. J. Merelo, M. G. Arenas, J. Carpio, P. A. Castillo, V. M. Rivas, G. Romero, and M. Schoenauer, “Evolving objects,” *In M. Graña, editor, FEA2000 (Frontiers of Evolutionary Algorithms) proceedings. Proc. JCIS'2000. P.P.Wang(ed). Editorial Association for Intelligent Machinery. ISBN:0-9643456-9-2. Vol I, pp.1083-1086. Atlantic City, NJ, Feb. 27-March 3.*, 2000.

[42] D. Moriarty and R. Miikkulainen, “Hierarchical Evolution of Neural Networks,” *In Proc. ICEC98, pp.428-433*, 1998.

[43] D. Thierens, J. Suykens, J. Vandewalle, and B. D. Moor, “Genetic weight optimization of a feedforward neural network controller,” *In Proc. CANNGA93, pp. 658-663. Springer-Verlag*, 1993.

[44] J. J. Merelo, M. Patón, A. Caas, A. Prieto, and F. Morán, “Optimization of a competitive learning neural network by genetic algorithms,” *Lecture Notes in Computer Science, Vol. 686, pp. 185-192, Springer-Verlag*, 1993.

[45] ——, “Genetic optimization of a multilayer neural network for cluster classification tasks,” *Neural Network World, vol.3, pp.175-186*, 1993.