Architecture Performance Prediction Using Evolutionary Artificial Neural Networks

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

[1] Architecture and Computer Technology Department
University of Granada
{pedro,amorag,jmerelo,juanlu}@geneura.ugr.es
[2] Computer Architecture Department
Technical University of Catalonia
HiPEAC European Network of Excellence
{mateo,mmoreto}@ac.upc.edu
[3] Barcelona Supercomputing Center
{francisco.cazorla,mateo.valero}@bsc.es
[4] Cornell University
sam@csl.cornell.edu

Abstract. The design of computer architectures requires the setting of multiple parameters on which the final performance depends. The number of possible combinations make an extremely huge search space. A way of setting such parameters is simulating all the architecture configurations using benchmarks. However, simulation is a slow solution since evaluating a single point of the search space can take hours. In this work we propose using artificial neural networks to predict the configurations performance instead of simulating all them. A prior model proposed by Ypek et al. [1] uses multilayer perceptron (MLP) and statistical analysis of the search space to minimize the number of training samples needed. In this paper we use evolutionary MLP and a random sampling of the space, which reduces the need to compute the performance of parameter settings in advance. Results show a high accuracy of the estimations and a simplification in the method to select the configurations we have to simulate to optimize the MLP.

1 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 predicts performance without actually running the simulator would save a lot of time in researching

M. Giacobini et al. (Eds.): EvoWorkshops 2008, LNCS 4974, pp. 175–183, 2008.

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 Ypek's work [1], who proposed using artificial neural networks (ANN) for architecture performance (instructions per cycle, 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.

The experimentation process consists in randomly selecting 1% of the search space configurations. Those simulated points are used to train the MLP, and this is used afterwards to predict the rest of architecture configuration performance. Since the MLP is a fast method, a big amount of configurations can be evaluated in a shorter time. 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 2 related work is analysed. Section 3 describes the problem of exploring architectural design spaces. In section 4 the GProp algorithm is introduced. Section 5 describes the experiments and presents the results obtained, followed by a brief conclusion in section 6.

2 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 superscalar microprocessor model which yields $87 - 95\%$ of accuracy in estimations. Yi et al. [11] 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 [12] and Cai et al. [13]) proposed using principal components analysis to identify the most important parameters and their correlations for processor design. Eeckhout et al. [14] and Phansalkar et al. [15] used similar methods for workload and benchmark composition.

Muttreja et al. [16] developed high-level models to estimate performance and energy consumption. They simulated several embedded benchmarks with 1.3% error. Lee and Brooks [17] 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. [18] developed a hybrid simulator to model instruction and data streams, Rapaca et al. [19] used another hybrid simulator and instructions code to infer information that is used to estimate statistics for other application code. Other authors, such as Wunderlich et al. [20] modeled minimal instruction stream to achieve

results within desired confidence intervals. Haskins and Skadron [21] sampled application code to create a cache and branch predictor state.

Ypek 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.

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.

3 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 study the memory system and the CPU design problems. These are defined by a set of parameters (see [22] for details). We use the benchmark suite SPEC CPU 2000 [23] 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.

Table 1 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 requires 23040 simulations per benchmark.

Table 2 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 2 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.

4 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 evolutionary algorithms (EA) [24,25] to find a solution close to the

Table 1. 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)

global optimum, and the ability of the quick-propagation algorithm [26] 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 [27] algorithm.

In GProp, an individual is a data structure representing a complete MLP with two hidden layers, which implies the use of specific operators. Five variation operators are used to change MLPs: mutation, crossover, addition and elimination of hidden units, and quick-propagation training applied as operator.

The genetic operators act directly upon the ANN object, but only initial weights and the learning constant are subject to evolution, not the weights obtained after training. In order to compute fitness, a clone of the MLP is created, and thus, the initial weights remain unchanged in the original MLP.

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

Table 2. 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

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 like 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 not normalized values.

5 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 using the 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.

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 3 (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).

Table 3. 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 Ypek et al. [1] and with the GProp method.

Application	Ypek 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	Ypek 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

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 robustness with the low standard deviations reported versus those reported in [1] (Ypek column in the table).

Tables 4 (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, Ypek et al. [1] only report the value 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.

Table 4. 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

6 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 just use 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 propose 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 current simulation methods capabilities.

As future work, we plan the automatic exploitation of the promising settings that the MLP has discovered within the search space applying evolutionary techniques.

Acknowledgements

This work has been supported by the Spanish MICYT 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. Ipek, E., McKee, S.A., de Supinski, B.R., Schulz, M., Caruana, R.: Efficiently Exploring Architectural Design Spaces via Predictive Modeling. In: ASPLOS 2006, pp. 195–206 (2006)
2. Martonosi, M., Skadron, K.: NSF computer performance evaluation workshop (2001), http://www.princeton.edu/mrm/nsf_sim_final.pdf
3. Jacob, B.: A case for studying DRAM issues at the system level. IEEE Micro 23(4), 44–56 (2003)
4. Davis, J., Laudon, J., Olukotun, K.: Maximizing CMP throughput with mediocre cores. In: Proc. IEEE/ACM International Conference on Parallel Architectures and Compilation Techniques, pp. 51–62 (2005)
5. SaarTsechansky, M., Provost, F.: Active learning for class probability estimation and ranking. In: Proc. 17th International Joint Conference on Artificial Intelligence, pp. 911–920 (2001)
6. Castillo, P.A., Carpio, J., Merelo, J.J., Rivas, V., Romero, G., Prieto, A.: Evolving Multilayer Perceptrons. Neural Processing Letters 12(2), 115–127 (2000)
7. Castillo, P.A., Merelo, J.J., Rivas, V., Romero, G., Prieto, A.: G-Prop: Global Optimization of Multilayer Perceptrons using GAs. Neurocomputing 35(1-4), 149–163 (2000)
8. Castillo, P., Arenas, M., Merelo, J.J., Rivas, V., Romero, G.: Optimisation of Multilayer Perceptrons Using a Distributed Evolutionary Algorithm with SOAP. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 676–685. Springer, Heidelberg (2002)
9. Castillo, P., Merelo, J., Romero, G., Prieto, A., Rojas, I.: Statistical Analysis of the Parameters of a Neuro-Genetic Algorithm. IEEE Transactions on Neural Networks 13(6), 1374–1394 (2002)
10. Karkhanis, T., Smith, J.: A 1st-order superscalar processor model. In: Proc. 31st IEEE/ACM International Symposium on Computer Architecture, pp. 338–349 (2004)
11. Yi, J., Lilja, D., Hawkins, D.: A statistically-rigorous approach for improving simulation methodology. In: Proc. 9th IEEE Symposium on High Performance Computer Architecture, pp. 281–291 (2003)
12. Chow, K., Ding, J.: Multivariate analysis of Pentium Pro processor. In: Proceedings of Intel Software Developers Conference Track 1, Portland, Oregon, USA, October 27-29, 1997, pp. 84–104 (1997)
13. Cai, G., Chow, K., Nakanishi, T., Hall, J., Barany, M.: Multivariate prower/performance analysis for high performance mobile microprocessor design. In: Power Driven Microarchitecture Workshop (ISCA 1998), Barcelona (1998)

14. Eeckhout, L., Bell Jr, R., Stougie, B., De1Bosschere, K., John, L.: Control flow modeling in statistical simulation for accurate and efficient processor design studies. In: Proc. 31st IEEE/ACM International Symposium on Computer Architecture, pp. 350–336 (2004)
15. Phansalkar, A., Josi, A., Eeckhout, L., John, L.: Measuring program similarity: Experiments with SPEC CPU benchmark suites. In: Proc. IEEE International Symposium on Performance Analysis of Systems and Software, pp. 10–20 (2005)
16. Muttreja, A., Raghunathan, A., Ravi, S., Jha, N.: Automated energy/performance macromodeling of embedded software. In: Proc. 41st ACM/IEEE Design Automation Conference, pp. 99–102 (2004)
17. Lee, B., Brooks, D.: Accurate and efficient regression modeling for microarchitectural performance and power prediction. In: Proc. 12th ACM Symposium on Architectural Support for Programmming Languages and Operating Systems (ASPLOS-XII), San Jose, California, USA, pp. 185–194. ACM Press, New York (2006)
18. Oskin, M., Chong, F., Farrens, M.: HLS: Combining statistical and symbolic simulation to guide microprocessor design. In: Computer Architecture, 2000. Proc. 27th IEEE/ACM International Symposium on Computer Architecture (SIGARCH Comput. Archit. News), pp. 71–82. ACM Press, New York (2000)
19. Rapaka, V., Marculescu, D.: Pre-characterization free, efficient power/performance analysis of embedded and general purpose software applications. In: Proc. ACM/IEEE Design, Automation and Test in Europe Conference and Exposition, pp. 10504–10509 (2003)
20. Wunderlich, R., Wenish, T., Falsafi, B., Hoe, J.: SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In: Proc. 30th IEEE/ACM International Symposium on Computer Architecture (ISCA), San Diego, California, USA, June 9-11, 2003, vol. 8, pp. 84–95. IEEE Computer Society Press, Los Alamitos (2003)
21. Haskins, J., Skadron, K.: Minimal subset evaluation: Rapid warm-up for simulated hardware state. In: Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors, September 23-26, 2001, p. 32. IEEE Computer Society Press, Washington (2001)
22. Renau, J.: SESC (2007), `http://sesc.sourceforge.net/index.html`
23. SPEC: Standard Performance Evaluation Corporation. SPEC CPU benchmark suite (2000), `http://specbench.org/osg/cpu2000`
24. Goldberg, D.: Zen and the art of genetic algorithms. In: Procs. of the 6th International Conference on Genetic Algorithms, ICGA 1995, pp. 80–85 (1995)
25. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd Extended edn., Springer, Heidelberg (1996)
26. Fahlman, S.: Faster-Learning Variations on Back-Propagation: An Empirical Study. In: Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann, San Francisco (1988)
27. Whitley, D.: The GENITOR Algorithm and Selection Presure: Why rank-based allocation of reproductive trials is best. In: Schaffer, J.D. (ed.) Procc of The 3th Int. Conf. on Genetic Algorithms, pp. 116–121. Morgan Kaufmann, San Francisco (1989)