An Online Tool for Teaching Design Trade-offs in Computer Architecture

Azam Beg[1], Walid Ibrahim[2]
College of Information Technology, UAE University, Al-Ain, United Arab Emirates
[1]abeg@uaeu.ac.ae, [2]walidibr@ uaeu.ac.ae

***Abstract* - In order to make a connection between the theoretical and practical aspects of computer architecture/organization courses at undergraduate and graduate levels, many software tools have been used in the past. If a large design-space needs to be explored using detailed simulations of a few industry-standard benchmark programs, the computational and time resources can become an impediment, thus placing a practical limit on the number of experiments, a student can complete in one semester. In this paper, we propose a software tool for predicting processor system performance. The tool can be used for teaching how the hardware configurations (processor microarchitecture, memory hierarchy, etc.) and/or software (benchmark program) characteristics affect the system throughput (as represented by instructions completed per cycle). Use of the proposed tool in computer architecture classes has demonstrated its effectiveness in improving the students' understanding of the related topics.**

Index Terms – Computer architecture, Simulation tools, Neural method, Performance prediction model, Web-based education.

INTRODUCTION

Computer architecture and organization are considered ones of the most difficult courses both to teach and to learn. It is usually a challenge to teach a subject in an environment where the covered topics are advancing very rapidly. In such conditions, the instructors must be up to date with the state of the art. At the same time, they should continuously revise their lectures, tutorials, problem sets, lab exercises, and exams to match the new development in the covered topics. For example, today's optical storage material should be revised to cover the newly released HD DVD and Blu-Ray technologies in addition to the widely-used CD and DVD technologies. Additionally, the lab component in computer architecture courses requires the design and execution of both hardware and software experiments. This component should be designed with extra care to make a clear link between the theory and the practical labs.

Learning computer architecture is also challenging because of its high degree of complexity. It requires the understanding of several interrelated subjects that include system design, electronic circuits, digital logic, assembly-language programming, as well as application level programming, discrete math and performance analysis.

In order to fill the gap between the theoretical and practical aspects of computer architecture/organization courses at undergraduate and graduate levels, many software tools have been created and used in the past. These tools vary in how they handle digital system simulation. They usually offer means for adding/removing hardware components, viewing simulation results, and conducting statistical analysis of system performance. The nature of these tools varies widely in several dimensions [1]−[3] that include: simulation level (cycle-accurate, overall behavior), level of detail (functional blocks, RTL), scope (processor only, system level), as well as user-interface (graphical or command-line). Examples of some basic simulators are: Babbage's analytical engine, CASLE, CPU-SIM, EasyCPU, Little Man Computer, etc. The medium-complexity simulators include: SPIM, MIPSpim, THRSim11, etc. and some advanced simulators are: DLXSim, RSIM, OSim, SimpleScalar, etc.

If a large design-space needs to be explored using detailed simulations of a few industry-standard benchmark programs, the computational and time resources can place a practical hurdle on the number of experiments a student can complete in one semester. Consider the situation when the students are asked to study the effect of changing several parameters (e.g., number of integer multipliers, number of float point multipliers, fetch queue width, etc.) on the computer system performance. To achieve that, the students have to run several sets of simulations. In each set, they should change only a single parameter and measure the system performance while a benchmark program is executed. It is obvious that running these experiments using a detailed simulation-based tool (e.g., cycle-accurate simulator) will consume a large amount of computational and time resources.

In this paper, we propose a software tool, named *PerfPred*, for predicting processor system performance; the tool can be used for teaching how the *hardware* configurations (processor microarchitecture, memory hierarchy, etc.) and/or *software* (benchmark program) characteristics affect the system throughput. Examples of hardware parameters are: issue-width, ALU count, cache size, cache configuration, branch prediction scheme, etc. The *software* parameters are determined by the benchmark program selected for execution. *PerfPred* provides the output in terms of instructions per cycle (IPC) which is a widely-used metric for a processor system throughput. The core of *PerfPred* is a machine-learnt model (based on neural network (NN) methodology) that speeds the task of system performance prediction to less than a second; this is several

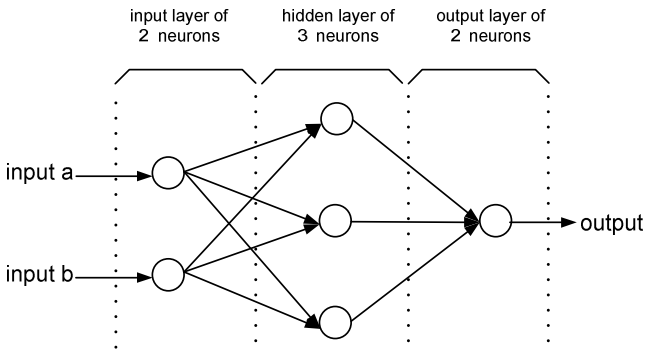

FIGURE 1
A SIMPLE NEURAL NETWORK CONTAINING THREE LAYERS: INPUT, HIDDEN, AND OUTPUT.

orders of magnitude faster than detailed simulation-based methods [4]. *PrefPred* has a user-friendly web-based interface, which removes the need for program installation on individual machines.

NN's ability to model the behavior of non-linear and high-dimensional systems has been used extensively in the past. NNs comprise of simple processing entities called neurons which emulate the characteristics of biological neurons (Figure 1). Interconnections of neurons parallelize the operations that are usually performed sequentially by the traditional computers .NN-model creation (*training*) involves repeated presentation of known input-output sets (*training examples*) to the network. With each cycle of training (*epoch*), NN's internal structure (specifically, the *neuron weights*) is adjusted in an attempt to bring the NN-output(s) closer to those of the training examples [5].

In the next section of this paper, we review the previous work related to teaching of computer architecture. Then we describe how a NN is used to build the nucleus of *PerfPred*. The user interface is also discussed, followed by two use cases of *PerfPred*. The last section of this paper presents the conclusions as well as future extensions to the current work.

PREVIOUS WORK

Due to the complexity of the computer architecture and organization courses, a variety of educational tools have been developed and used by several education institutions worldwide to ease this complexity and improve the quality of the teaching process. These tools differ greatly in their complexity, simulation level, and user interface. Early tools were mainly text-based, while most of the current ones have graphical interfaces that allow them to provide visual representation to the internal operation of a computer system. To help instructors select the right tool for teaching a specific computer architecture topic, an orientation to the current state-of-the-art in computer architecture education resources is provided in [6]. The orientation illustrates five major websites dedicated for computer architecture educational resources. Throughout the orientation, the authors attempt to identify the gaps between the current resources available on the Web and the resources needed by the instructors and try to find why this gap exists.

Previous work in computer architecture tools can be divided into three main categories [7]. The first category includes simple tools that are mainly used in the introductory undergraduate courses. The second category includes more advanced tools that are largely used in advanced microarchitecture courses. These tools focus more on sophisticated parallel architectures with multiple CPUs. Finally, the third category comprises the tools dedicated to memory subsystem category which focus on the interactions among the CPU, main memory, and cache memory.

Some of the tools in the first category, such as the Little Man Computer [8], use only simple addressing modes, limited instruction sets, and very simple memory models. While, other tools (belonging to the third category) such as LC2 [9], SPIM [10], and SPIMSAL [11] tend to include more realistic set of addressing modes, more complete instruction sets, more realistic memory hierarchies, and sometimes an interrupt mechanism.

The third category (more advanced tools) includes SimpleScalar [12], DLX, MipSim [13] and Mic-1 [14]. These tools are designed to allow the observation of machine language execution at the microcode level (e.g., data paths, control units). Advanced tools can be used to investigate the advantages and disadvantages (e.g., efficiency, complexity) of performance enhancing techniques such as pipelining, branch prediction and instruction-level parallelism. Some of these simulators are microprogrammable, allowing students to experiment with the design of instruction sets. The third category also includes tools for simulating multiprocessor architecture. These tools (e.g., GEMS [15], RSIM [16], WWT2 [17]) allow students to study the effect of different design parameters (e.g., memory sharing and locking schemes, instruction level parallelism, etc.) on the performance of complicated, multithreaded workloads such as databases and web servers.

The effect of the memory hierarchy (cache, main, and virtual) on the performance of computer systems is a mandatory topic in any undergraduate computer architecture/organization course. Therefore, researchers have paid great attention to the development of realistic memory simulation tools. Some of the well known tools include Dinero IV [18], VirtualMemory [19] and SPMCache [20]. Dinero IV is a cache simulator that supports various types of caches, i.e., direct mapped, set associative and fully associative. Block sizes, associativity and other parameters may also be specified. The tool can also be used to simulate multiple levels of cache as well as for classifying the type of misses (compulsory, conflict and capacity).

SMPCache is a trace-driven simulator for analysis and teaching of cache memory systems on symmetric multiprocessors. It has a graphic user-interface that allows the students to experiment different theoretical aspects of cache memories and multiprocessors. Some of the parameters that the students can study are: program locality; influence of number of processors, cache coherence protocols, schemes for bus arbitration, mapping, replacement policies, cache size (blocks in cache), number of cache sets (for set-associative caches), number of words by block (memory block size), and the word size. However, a disadvantage of SMPCache is its slow simulation speed. VirtualMemory presents a graphical interface allowing the

TABLE 1
PARAMETERS USED FOR *SIM-OUTORDER* SIMULATIONS AND FOR SUBSEQUENT *PERFPRED* MODEL CREATION AND TESTING

Parameter Type	Input Neuron Description	Range
Hardware	Load/store queue (instructions)	2, 4, 8, 16, 32, 64, 128
Hardware	Fetch queue width (instructions)	2, 4, 8, 16, 32, 64, 128
Hardware	Decode width (instructions)	1, 2, 4, 8, 16, 32, 64
Hardware	Issue width in a cycle	1, 2, 4, 8, 16, 32, 64
Hardware	Commit width in a cycle	1, 2, 4, 8, 16, 32, 64
Hardware	Register update unit (instructions)	2, 4, 8, 16, 32, 64, 128
Hardware	Ratio of CPU and bus speeds	2, 4, 8, 16, 32, 64, 128
Hardware	Integer ALUs	1, 2, 3, 4, 5, 6, 7, 8
Hardware	Integer multipliers	1, 2, 3, 4, 5, 6, 7, 8
Hardware	Branch prediction scheme	Taken, Not-taken, Perfect (represented as '*symbol*' in NN)
Hardware	Branch misprediction penalty (cycles)	1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128
Software	bzip2 (benchmark)	0, 1
Software	crafty (benchmark)	0, 1
Software	eon (benchmark)	0, 1
Software	mcf (benchmark)	0, 1
Software	twolf (benchmark)	0, 1
Software	vortex (benchmark)	0, 1

simulation of virtual memory (main memory, hard disk and page table) and exhibits statistical data during the simulation.

NN-BASED *PERFPRED* TOOL DEVELOPMENT

The NN-based *PerfPred* proposed in this paper includes different microarchitectural (hardware) parameters to predict the processor system performance which is measured in instructions completed per cycle (IPC). Each hardware parameter is represented by a single neuron in the input layer. In addition, six neurons represent six different SPEC2000 CPU integer benchmarks (i.e., *bzip2*, *crafty*, *eon*, *mcf*, *twolf*, and *vortex*) [21]. There is a single hidden layer in the model. One hidden layer is considered sufficient to represent most non-linear systems [5]. The model output (IPC) is produced by single output neuron. The general hierarchy of this model resembles the NN shown in Figure 1. Numerous experiments were carried out to determine a suitable count of neurons in the hidden layer of the NN (Details of these experiments are discussed shortly).

The NN presented here models a superscalar processor system, which in turn is based on SimpleScalar's *sim-outorder* architecture [12]. Table 1 lists different configurations of *sim-outorder* which were used to run more than 6000 simulations. The time for a single simulation on an x86-based Linux machine ranged from 0.5 to 2 hours. Six SPEC2000 CPU integer benchmarks with their respective 'test' inputs were used in these simulations. The simulations were fast-forwarded by 100 million instructions while the maximum number of instructions was limited to 500 million [12], [21].

The data acquired from *sim-outorder* simulations needed to be re-scaled and transformed. As widely known, this step is required to make sure that all of NN inputs equitably influence the training process. As an example, the log_2 transformation was used for the values: {2, 4, 8, 16, 32, 64, and 128}. After transformation, the input values were scaled to the range [0, 1].

The neural or analytical models can be several orders of magnitude faster than simulation models but a price may have to be paid in terms of accuracy. 20-39% error ranges were reported in many research works [22]-[26]. For the NN models, we opted for 15% error allowance for training and validating. Validation was done with the input-output examples (10% of total dataset); the later were not *shown* to the NN during training.

Brain-Maker (version 3.75), an MS-Windows based software package [27] was used to create the NN models. This package uses *feed-forward* and *back-propagation* methodology of neural modeling.

In our NN models, numbers of input and output neurons were fixed. However, in an effort to find an optimum-sized NN model, hidden layer sizes from 2 to 30 were experimented with. As expected, the more the number of hidden neurons, the more the model's training iteration count. Each NN-configuration was trained many times with randomly-set neuron weights at the training onset, to reduce the chances of running into *local minima*. As mentioned earlier, 90% of the complete data set was used for training purposes, while the other 10% was used for test/validate the NNs' predictive abilities.

The NN model training and validation statistics are graphically shown in Figure 2. We notice that with a count of just 8 neurons in the hidden layer, training accuracy (defined by the number of training sets predicted within the desired error allowance) of nearly 85% was attained. With 8 or more neurons in the hidden layer, the validation accuracy (defined by the number of validation sets predicted within the desired error allowance) remained quite close to the training accuracy, thus demonstrating the model's learning effectiveness. Further more, increasing the hidden layer size beyond 8 neurons did not lead to any significant improvement in prediction accuracy.

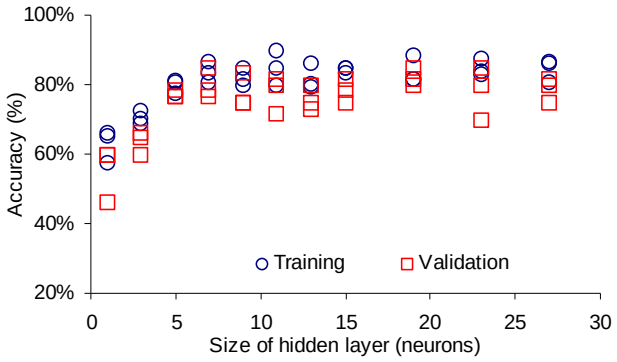

FIGURE 2
TRAINING AND VALIDATION ACCURACIES AS A FUNCTION OF THE SIZE OF THE SINGLE HIDDEN LAYER IN THE NN MODEL. BRAIN MAKER PARAMETERS: TRAINING/TESTING TOLERANCE = 0.15; LEARNING RATE ADJUSTMENT TYPE = HEURISTIC; INITIAL NEURON WEIGHTS SET RANDOMLY.

Notice that all hardware (and software) parameters (listed in Table 1) were used in building the currently present NN models. We are extending the current research to study the contribution of each input parameter to the model. This investigation may lead to a fewer number of neurons in the input and hidden layers.

PerfPred User Interface

A web-browser interface developed (in PHP) allows user to easily select the input parameters (see Figure 3). The users pick a given parameter using one of the 'radio-buttons' (near the middle of the screen). The range for this parameter is entered in two boxes (labeled 'To' and 'From') on the left of the radio button. All other parameters take a single value that is entered in the 'From' box. Once the input parameters are selected, the 'Plot' or 'Show values' button is pressed. Based on this selection, the predicted values are either plotted or listed in a *text* box. The text box allows the user to save the predicted values and export them into a different program such as MS-Excel. Values from multiple runs can be combined into a single plot; two such examples are shown in the next section.

PERFPRED USE CASES

In this section, we present two examples of the insight a user can gain by using *PerfPred* performance prediction tool. We chose 3 benchmarks (*bzip2*, *crafty*, and *eon*) to study the impact of varying the number of instructions issued per cycle (issue width) on the processor throughput. We varied the issue width from 2 to 64. For all benchmarks, we observe the general trend that IPC increases as more instructions are issued in a cycle (see Figure 4). However, the incremental gains made due to added issue hardware diminish when we go beyond 8 instructions. This limitation may come from the limited parallelism inherent in the programs themselves.

As a second example, Figure 5 shows the behavior of the processor system in response to branch misprediction, for 3 different benchmarks *mcf*, *twolf* and *vortex*. The number of cycles taken after a mispredicted branch ranges

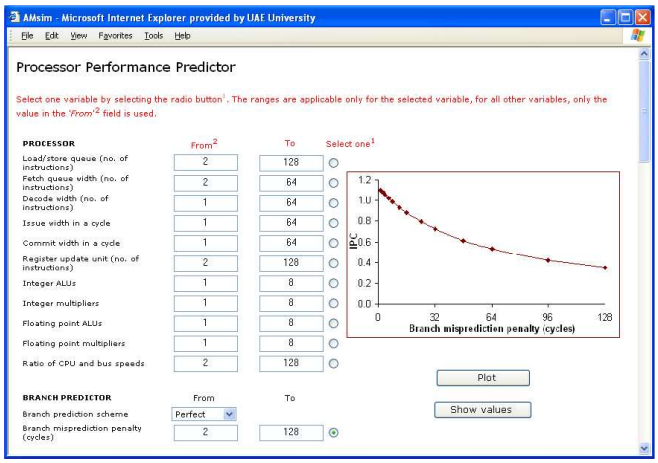

FIGURE 3
WEB-BASED INTERFACE OF *PERFPRED*. IN THIS EXAMPLE, 'MISS PENALTY OF BRANCH PREDICTOR' IS SELECTED TO STUDY ITS EFFECT ON THE SYSTEM THROUGHPUT.

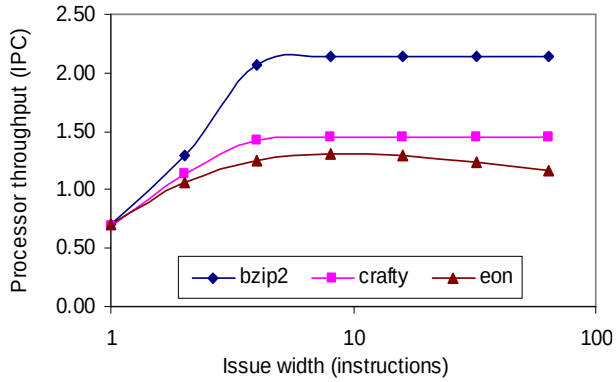

FIGURE 4
SENSITIVITY OF PROCESSOR THROUGHPUT (IPC) TO ISSUE WIDTH WHILE ALL OTHER PARAMETERS ARE FIXED. INITIALLY, ISSUING MORE INSTRUCTIONS IN A CYCLE HELPS IMPROVES THE PROCESSOR THROUGHPUT. HOWEVER, THIS INCREASE STOPS AS THE ISSUE WIDTH EXCEEDS 8 INSTRUCTIONS.

from 1 to 128. Expectedly, the longer it takes to recover from a branch misprediction, the lower the throughput (IPC) of the processor. The IPC curves for all the benchmarks are similar in shape. We notice that *vortex* shows less sensitivity to increased misprediction penalty than *mcf* and *twolf*. Characterization of basic blocks of these programs can shed more light on the reasons for the sensitivity.

Note that the above results could have been acquired by running 21 actual simulations for the first use case and 42 simulations for the second use case; both of these would have taken a day or more of computing time on one of today's PCs. In comparison, the same or more amount of information can be acquired by running the *PerfPred* tool in less than a second.

To test the effectiveness of *PerfPred* as a teaching tool, we used it in one of the undergraduate level computer architecture courses in the previous (Fall 2006) semester. The student perception of the effect of changing different microarchitectural features in a processor was evaluated before and after the tool introduction. Subsequent testing showed some improvement in understanding of the concepts. Continued use of the tool in the current and future semesters is expected to reinforce the tool's usefulness.

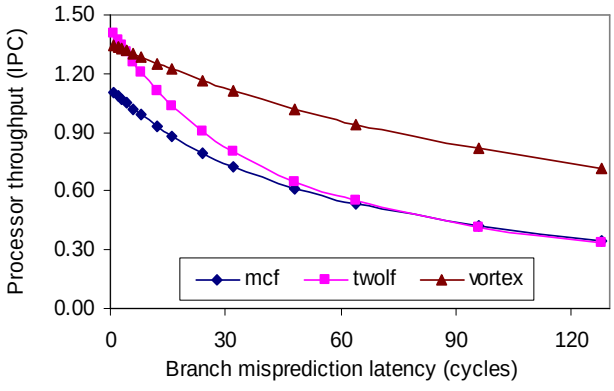

FIGURE 5
SENSITIVITY OF PROCESSOR THROUGHPUT TO ISSUE WIDTH WHILE ALL OTHER PARAMETERS ARE FIXED. AS EXPECTED, THE PROCESSOR COMPLETES FEWER INSTRUCTIONS PER CYCLE AS THE BRANCH MISPREDICTION LATENCY.

CONCLUSIONS

In this paper, we have presented an NN-based processor tool/model for performance prediction; the tool was created using the data acquired from a large number of simulations covering a wide area of superscalar processor design space. The resultant model provided fast and reasonably accurate estimates of the throughput of the processor. Prediction accuracies of 85% or better were observed, which are comparable to related NN models previously reported. The proposed performance prediction tool can be used for the study of microarchitectural trade-offs in a processor system design. The tool can also be used effectively in computer science courses related to compiler optimization.

In pedagogical settings, the students gain the theoretical knowledge of computer architecture/organization in the lectures; subsequent laboratory assignments using *PerfPred* reinforce their learning of factors affecting the computer system performance. Use of tools similar to *PrefPred* (e.g., SPMCache) in computer architecture classes has demonstrated its effectiveness in improving the students' understanding of the related topics. It is expected that using *PrefPred* will significantly help instructors close the gap between the theoretical and practical aspects of computer architecture/organization courses

Further enhancements to the proposed predictive model are a subject of our ongoing research. The improvements include: investigation of methods for improved prediction accuracy; dimension reduction of the model; inclusion of a program's dynamic characteristics as input parameters, etc.

ACKNOWLEDGMENT

This work was financially supported by the Research Affairs at the UAE University under a contract no. 02-02-9-11/06. The authors would like to thank Mr. Junaid Aziz for setting up the simulation environment and for developing *PerfPred* web interface.

REFERENCES

[1] J. Djordjevic, A. Milenkovic, and N. Grbanovic, "Flexible web-based educational system for teaching computer architecture and organization," *IEEE Trans. Educ.*, vol. 48, pp. 264–273, May 2005.

[2] E. Dirkx and J. Tiberghien, "An animated simulation environment for microprocessors," *Microprocessing and Microprogramming*, vol. 24, pp. 149–152, sep. 1988.

[3] B. Lees, "An interactive modeling system to assist the teaching of computer architecture," *Computers & Education*, vol. 8, pp. 419–426, 1984.

[4] A. Beg and Y. Chu, "Modeling of trace- and block-based caches," *J. Circuits, Syst. & Comput. (JCSC)*, vol. 16, 2007. In Press.

[5] T. Mitchell, "Machine learning," McGraw-Hill Co., Columbus, OH, USA, 1997.

[6] W. Yurcik and E. F. Gehringer, "A survey of web resources for teaching computer architecture," *Proc. Workshop on Comput. Archit. Educ. (WCAE)*, Anchorage, AK, USA, May 2002, pp. 126–131.

[7] G. S. Wolffe, W. Yurcik, H. Osborne, and M. A. Holliday, "Teaching computer organization with limited resources using simulators," *Proc. SIGCSE Technical Symp. on Comput. Sci. Educ.*, Covington, USA, 2002, pp.176-180.

[8] H. Osborne and W. Yurcik, "The educational range of visual simulations of the little man computer architecture paradigm," *Proc. 32nd ASEE/IEEE Frontiers in Educ. Conf. (FIE)*, Boston, MA, USA, Nov. 2002, pp. S4G-19–S4G-24.

[9] Y. Patt and S. Patel, "Introduction to computing systems," McGraw-Hill, 2001.

[10] D. Patterson and J. Hennessy, "Computer organization and design," 2nd edition, Morgan Kaufmann, 1998.

[11] J. Goodman and K. Miller "A programmer's view of computer architecture," Oxford U. Press, 1993.

[12] T. Austin, E. Larson and D. Ernst "SimpleScalar: an infrastructure for computer system modeling," *Comput.*, vol. 35, pp. 59–67, Feb. 2002.

[13] H. Grunbacher and H. Khosravipour, "WinDLX and MIPSim pipeline simulators for teaching computer architecture," *Proc. IEEE Symp. and Workshop on Eng.of Comput. Based Syst.*, Friedrichshafen, Germany, Mar. 1996, pp. 412–417.

[14] A. Tanenbaum, "Structured computer organization," 4th edition, Prentice Hall, 1999.

[15] M. M. K. Martian, et al, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Comput. Archit. News (CAN)*, vol. 33, pp. 92–99, Sept. 2005.

[16] C. J. Hughes , V. S. Pai , P. Ranganathan , and S. V. Adve, "RSIM: Simulating Shared-Memory Multiprocessors with ILP processors," *IEEE Comput.*, vol. 35, pp. 40–49, Feb. 2002.

[17] S. S. Mukherjee, et al. "Wisconsin wind tunnel II: A fast and portable architecture simulator," *Workshop on Perf. Analysis and its Impact on Design*, June 1997.

[18] M. D. Hill. University of Wisconsin. *Dinero IV*. [Online]. Available: http://www.cs.wisc.edu/~markhill/DineroIV/.

[19] N. Tran, and D. A. Menasce. George Mason University. *VirtualMemory*. [Online]. Available: http://cs.gmu.edu/cne/workbenches/vmsim/vm.html .

[20] M. A. Vega-Rodríguez, J. M. Sánchez-Pérez, and J. A. Gómez-Pulido, "An educational tool for testing caches on symmetric multiprocessors," *Microprocessors and Microsystems*, vol. 25, pp. 187–194, June 2001.

[21] Standard Performance Evaluation Corporation. SPEC2000 CPU benchmarks. [Online]. Available: http://www.spec.org/cpu2000/

[22] S. Wallace, and N. Bagherzadeh, "Modeled and measured instruction fetching performance for superscalar microprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, pp. 570–578, 1998.

[23] D. B. Noonburg, and J. P. Shen, "Theoretical modeling of superscalar processor performance," *Proc. 27th Intl. Symp. Microarch.*, San Jose, CA, USA, 1994, pp. 52–62.

[24] T. Wada, and S. Przybyski, "An analytical access time model for on-chip cache memories," *IEEE J. Solid State Circ.*, vol. 27, pp. 1147–1156, 1992.

[25] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "Simpoint 3.0: Faster and more flexible program analysis," *J. Instr. Level Parallelism (JILP)*, pp. 1–28, vol. 7, Sep. 2005.

[26] A. Agarwal, M. Horowitz, J. Hennessy, "An analytical cache model," *ACM Trans. Comput. Syst.*, vol. 7, pp. 184–215, 1989.

[27] "Brain-Maker User's Guide and Reference Manual," California Scientific Press, Nevada City, CA, 1998.