A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space

AZAM BEG,[1] and WALID IBRAHIM[1,2]

[1]UAE University, College of Information Technology, Al Ain, Abu Dhabi, UAE

[2] Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada

E-mail: {abeg, walidibr}@uaeu.ac.ae

Abstract

This paper presents a Web-based tool for predicting the performance of computer systems. The tool can be used for teaching how the hardware configurations (processor microarchitecture, memory hierarchy, etc.) and/or software (benchmark program) characteristics can affect the system throughput. Use of the proposed tool in computer architecture classes has demonstrated its effectiveness in improving the students' understanding of the related topics.

Keywords—Computer Architecture, Neural Networks, Cognitive Tools, Web-based Learning, e-Learning

1 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. Computer

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

architecture courses should also evolve to cover the new dual/multi-core architectures. Additionally, the lab component in computer architecture courses requires the design and execution of both hardware and software experiments. The labs should be designed with extra care to link them clearly to the theory.

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 bridge 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 the way they handle computer system simulations. The tools usually offer means for adding/removing hardware components, viewing simulation results, and conducting statistical analysis on the system performance. The nature of these tools varies widely in several dimensions [1]–[3]: 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 the performance of several computer architectures needs to be evaluated using detailed simulations of industry-standard benchmark programs, the computational and time resources can place a practical bound 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

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

parameters (e.g., number of cores, number of integer multipliers, number of floating point multipliers, fetch queue width, etc.) on the 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 huge number of compute cycles.

Neural networks (NNs) emulate the characteristics of biological neurons. NNs' 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* (Figure 1). Interconnections of neurons parallelize the operations that are usually performed sequentially by the traditional computers. The creation (*training*) of an NN-model involves repeated presentation of known input-output sets (*training examples*) to the network. With each cycle of training (*epoch*), the internal structure of the NN (specifically, the *neuron weights*) is adjusted in an attempt to bring the NN-output(s) closer to those of the training examples [4]-[19].

In this paper, we present a novel NN based software tool, named *PerfPred*, for predicting processor system performance. The tool can be used in a cognitive domain to improve the students understanding of how the *hardware* configurations (processor microarchitecture, memory hierarchy, etc.) and/or *software* (benchmark programs) characteristics affect the system throughput. Examples of hardware characteristics include: issue-width, ALU count, cache size, cache configuration, branch prediction scheme, etc. *Software* parameters depend on the program being executed. *PerfPred* uses the widely-used *instructions per cycle* (IPC) metric to measure the system throughput. The core of *PerfPred* is an NN prediction model which makes the tool several orders

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

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

In the next section, previous works related to educational tools in the field of computer architecture are briefly reviewed. The following section describes how an NN is used to build *PerfPred* tool. The tool's user interface is then described, followed by three use-cases. The paper's conclusions as well as the future extensions to the current work are presented in the last section.

2 PREVIOUS WORK

A variety of educational tools for use in computer architecture and organization courses, have been developed; the tools can not only ease the complexity of teaching but also improve its quality. The tools differ in several aspects which include complexity, simulation level, and user interface. Early tools were mainly text-based, while most of the current ones have graphical interfaces that provide visual representation of 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 educational resources for computer architecture is provided in [20]; five major websites dedicated to the topic are discussed. Throughout the orientation, the authors attempt to identify the gaps between the current Web-based resources and the resources needed by the instructors; the authors also try to find why these gaps exist.

Previous work in computer architecture tools can be divided into three main categories [21]. The first category includes simple tools that are mainly used in the introductory undergraduate courses. The second category includes more advanced tools largely used in advanced microarchitecture courses. These tools focus on sophisticated parallel architectures with multiple

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

CPUs. Finally, the third category comprises the tools dedicated to memory subsystems, and focuses 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 [22], use only simple addressing modes, limited instruction sets, and very simple memory models. While, other tools such as LC2 [23], SPIM [24], and SPIMSAL [25] tend to include more realistic sets of addressing modes, more complete instruction sets, more realistic memory hierarchies, and sometimes an interrupt mechanism.

The second category (more advanced tools) includes SimpleScalar [26], DLX, MipSim [27] and Mic-1 [28]. 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 [29], RSIM [30], WWT2 [31]) 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 [32], VirtualMemory [33], and SMPCache [34]. 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

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

specified. The tool can also be used to simulate multiple levels of cache as well as to classify the type of misses (compulsory, conflict and capacity).

VirtualMemory [33] is an online tool that allows simulation of virtual memory (main memory, hard disk, and page table), and shows statistical data during the simulation. SMPCache [34] is a trace-driven simulator for analyzing and teaching of cache memory systems on symmetric multiprocessors. Some of the parameters that the students can study are: program locality of reference; influence of number of processors, cache coherence protocols, schemes for bus arbitration, mapping techniques, 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, the major disadvantage of SMPCache is its slow simulation speed.

3 NN-BASED *PERFPRED* TOOL

As we mentioned earlier, an NN model constitutes the core of the *PerfPred* tool. The model uses several microarchitectural (hardware) parameters to predict the performance of different configurations of a processor system. The performance is measured in terms of the number of *instructions completed per cycle* (IPC). Each hardware parameter is represented by a single neuron in the NN's input layer. In addition, six other input neurons are used to represent six different SPEC2000 CPU integer benchmarks, namely, *bzip2, crafty, eon, mcf, twolf*, and *vortex* [35]. In our NN model, we opted to use single hidden layer of neurons, because one such layer is usually sufficient to represent most non-linear systems [4]. The model output (IPC) is produced by a single output neuron. We conducted numerous experiments – discussed shortly – to determine a suitable count of neurons in the hidden layer of the NN.

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

The proposed NN models a superscalar processor system, which in turn is based on SimpleScalar's *sim-outorder* architecture [26]. 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 10 minutes 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 [26], [35].

The data acquired from *sim-outorder* simulations needed to be transformed and scaled. As widely known, this step is required to insure that all the NN inputs equitably influence the training process [18], [19]. As an example, 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 of 0 to 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 [36]-[40]. For our NN model, we opted for 15% error allowance for the training and the validation/testing stages. Validation was done with input-output examples (10% of total dataset); this portion of the dataset was not shown to the NN during training.

Brain-Maker (version 3.75), an MS-Windows based software package [41], was used to create the NNs. This package uses *feed-forward* and *back-propagation* methodology of neural modeling. (NNs can also be easily created with a more widely available Neural Network ToolBox for Matlab. Sample code for a feed-forward NN is included in Appendix A). Brain-Maker parameters used during the training/validation process are as follow: training/testing tolerance = 0.15 (85% accuracy), learning rate adjustment type = heuristic, initial neuron weights set randomly.

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

In the proposed NN model, the numbers of input and output neurons were fixed. However, in an effort to find an optimally-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 initial neuron weights, 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 to test and validate 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.

We are extending the current research to study the contribution of each input parameter (listed in Table 1) to the NN model. This investigation may lead to a fewer number of neurons in the input and hidden layers.

3.1 *PerfPred* User Interface

A Web-based user interface has been designed for the *PerfPred* tool in order to improve its accessibility and availability. This interface (shown in Figure 3) allows students to easily select the input parameters for a given experiment. The students pick the desired parameter using one of the 'radio-buttons' (near the middle of the screen). The range for this parameter is entered in two text boxes (labeled 'From' and 'To') on the left of the radio button. All other parameters take single

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

values, as specified in the 'From' text box. After 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 student to save the predicted values and to export them into a different program, such as MS-Excel. Values from multiple runs can be combined into a single plot; three such examples are included in the next section.

4 *PERFPRED* USE CASES

In this section, a few examples are presented to highlight the insights students can gain by using *PerfPred* in a cognitive domain. Three benchmarks (*bzip2, crafty, and eon*) are used in these examples [35]. The first example deals with the fetch unit that brings in the instructions from the instruction cache on cache hits. The following example covers the next stage in the pipeline, where fetched instructions are sent to the decoding unit. The third example looks into the issue unit that sends the instructions to execution units as they become available.

In the first example, the 3 benchmarks are used to study the impact of instruction fetch queue (IFQ) on the processor throughput. During this experiment, the queue size was varied from 2 to 64. For all benchmarks, a general trend is observed that the IPC increases as the queue size increases as shown in Figure 4. The figure also shows that only an instruction fetch queue (IFQ) size of just 4 instructions is sufficient to get the maximum IPC. The *bzip* benchmark gets the most improvement going from IFQ of 2 instructions to an IFQ of 4 instructions, while the graphs of the *crafty* and *eon benchmarks* nearly overlap each other. Increased size of fetch unit may have limited use when instructions straddled across different cache block. The fetch unit looks at the set of instructions being issued and determines whether an instruction would cause structural or data hazard. If a hazard is likely to occur, the issue bandwidth reduces accordingly. In such cases, dynamic scheduling helps exploit wider instruction issues [24].

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

Figure 5 and Figure 6 show the effects of the decoder size and the issue width size on IPC respectively. Decoders are resident in the instruction dispatch stage. Just as the IPC plateaus at fetch width of 4, so does the decoder. Figure 5 shows that adding more than 4 decoders to the processor does not improve the processor efficiency. The curves of the *crafty* and *eon* benchmarks stay very close in this graph as well. Figure 6 shows that, initially, issuing more instructions in a cycle helps improve the processor throughput. However, the incremental gains due to additional issue hardware diminish, when the number of issued instructions goes beyond 8. A similar phenomenon was also observed when more decoders were added to the processor. In both cases (i.e., issue width and decode width), different benchmarks attain different benefits in terms of IPC. The upper limit on issue and decoder resources seems to be a function of the basic code block sizes (e.g., loop size) and the block frequency (loop recurrence). Another contributing factor could be the limited parallelism inherent in the programs themselves.

The effects of issue and commit stages on IPC exhibit similar trends as the earlier stages of the pipeline, i.e., fetch and decode units (see Figure 7). When the window size – the group of instructions from which one can execute – increases, more instructions can be issued. So the window size may be a limiting factor in the program runs. The compiler can help improve the parallelism in a program even when the window sizes are small. The integer programs – the only programs used in this paper – in general have lower parallelism than floating point programs [24].

Finally, the effect of branch misprediction penalty (in terms of cycles) on IPC is depicted in Figure 8. The graphs show how the penalty reduces the IPC when it increases from 1 cycle to 128. In order to achieve an appreciable amount of parallelism in programs, the executed instructions need to be widely separated. Higher accuracy in branch prediction schemes help improve the parallel execution of instructions. Speculative execution of instruction can also help the control

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

(branch) dependence [24].We also notice that the maximum IPC in all cases reaches only up to 2.07 for the three benchmark programs. Another point worth noting is that the system's IPC metric seems to depend heavily on the *dynamic* nature of the benchmark programs. This dynamic nature can be described as the instruction basic block distribution (i.e., their sizes and frequencies). A thorough investigation of the dynamic characteristics of the benchmark programs should reveal the reasons why IPCs for *crafty* and *eon* benchmarks are closer to each other but much lower than *bzip*.

Note that the data shown in Figures 4 to 8 would have required 84 runs of a simulator such as SimpleScalar. For a simulation run length of 500 million instructions, SimpleScalar takes an average of 13 minutes on our Linux-based x86 machine. So the total time for 84 simulations would be nearly 18 hours. In comparison, *PerfPred* provides performance estimations in just a few milliseconds. This significant improvement in speed has potential benefits for researchers and educators alike.

To test the effectiveness of *PerfPred* as a teaching tool, we used it in one of the undergraduate level computer architecture courses in one of the recent semesters. 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 introduced concepts. Continued use of the tool in the current and future semesters is expected to reinforce the tool's usefulness.

5 CONCLUSIONS

In this paper, we have presented *PerfPred*, an NN-based model for processor-system performance prediction; the tool was created using data acquired from a large number of

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

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 tool can be used for the study of microarchitectural design 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 such as *PerfPred* in computer architecture classes has demonstrated their effectiveness in improving the students' understanding of the related topics. It is expected that using *PerfPred* 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 programs' dynamic characteristics as input parameters, etc.

APPENDIX A

```
% Implementation of a three-layer feed-forward NN in MATLAB
neurcnt=[17, 16, 1];        % network size
                            % network instantiation:
net=newff(minmax,neurcnt,{'tansig','tansig','purelin'},'traingd');
net.trainParam.show = 50;  % training plot's x-axis divisions
net.trainParam.epochs = 1000;    % max epochs
net.trainParam.lr = 0.05;        % learning rate
net.trainParam.goal = 0.15;      % tolerance
net=train(net,p_trg,t_trg);      % network training
```

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

REFERENCES

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

[5] Beg, P.W.C. Prasad, and A. Beg, "Applicability of feed-forward and recurrent neural networks to Boolean function complexity modeling," *Expert Systems With Applications*, vol. 36, No. 1, 2008.

[6] Beg and Y. Chu, "Modeling of trace- and block-based caches," *J. Circ., Syst. & Comput.*, vol. 16, 2007.

[7] Assi, P.W.C. Prasad, A. Beg, and V.C. Prasad, "Complexity of XOR/XNOR Boolean functions: A Model using Binary Decision Diagrams and Back Propagation Neural Networks," *J. Comput. Sci. & Tech.,* vol. 7, No. 2, pp 141-147, 2007.

[8] Beg and Y. Chu, "Utilizing block size variability to enhance instruction fetch rate," *J. Comput. Sci. & Tech.,* vol. 7, No. 2, pp. 155-161, 2007.

[9] P. Chandana, A. Assi, and A. Beg, "Binary decision diagrams and neural networks*,"J. Supercomputing*, vol. 39, No. 3, pp. 301-320, 2007.

[10] A.K. Singh, A. Beg, and P.W.C. Prasad, "Modeling the path length delay projection," *Proc. Intl. Conf. Eng. & ICT (ICEI 2007)*, Nov. 2007.

[11] Beg and W. Ibrahim, "An online tool for teaching design trade-offs in computer architecture," *Proc. Intl. Conf. Eng. Educ.*, Sep. 2007.

[12] Beg, "Predicting processor performance with a machine learnt model," *Proc. IEEE Intl. Midwest Symp. Circuits & Syst., (MWSCAS/NEWCAS 2007)*, pp. 1098-1101, Aug. 5-8, 2007.

[13] P.W.C. Prasad and A. Beg, "A methodology for evaluation time approximation," *Proc. IEEE Intl. Midwest Symp. Circuits & Syst., (MWSCAS/NEWCAS 2007)*, pp. 776-778, Aug. 2007.

[14] Beg, P.W.C. Prasad, and S.M.N.A. Senanayake, "Learning Monte Carlo data for circuit path length," *Proc. Intl. Conf. Comput., Comm. & Control Tech., (CCCT 2007)*, Jul. 2007.

[15] Beg, P.W.C. Prasad, M. Arshad, and K. Hasnain, "Using recurrent neural networks for circuit complexity modeling", *Proc. IEEE INMIC Conf.*, pp. 194-197, Dec. 2006.

[16] P.W.C. Prasad, A.K. Singh, A. Beg, and A. Assi, "Modeling the XOR/XNOR Boolean functions' Complexity using Neural Networks," *Proc. IEEE Intl. Conf. on Electronics, Circ. & Syst., (ICECS 2006)*, pp. 1348-1351, Dec. 2006.

[17] Assi, P.W.C. Prasad, and A. Beg, "Boolean function complexity and neural networks," *Proc. WSEAS Intl. Conf. Neural Networks (NN'06)*, Jun. 2006.

[18] Beg and P.W.C. Prasad, "Data processing for effective modeling of circuit behavior," *Proc. WSEAS Intl. Conf. Evolutionary Comput. (EC'07)*, Vancouver, Canada, pp. 312-318, Jun. 2007.

[19] P.W.C. Prasad and A. Beg, "Investigating data preprocessing methods for circuit complexity models," *Expert Systems with Applications*, Vol. 38, No. 4, 2007.

[20] 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, pp. 126–131, May 2002.

[21] 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, pp.176-180, 2002.

[22] 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.

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

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

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

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

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

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

[33] N. Tran, and D. A. Menasce, George Mason University, *VirtualMemory*, [Online], Available: http://www.ecs.umass.edu/ece/koren/architecture/Vmemory/try.html

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

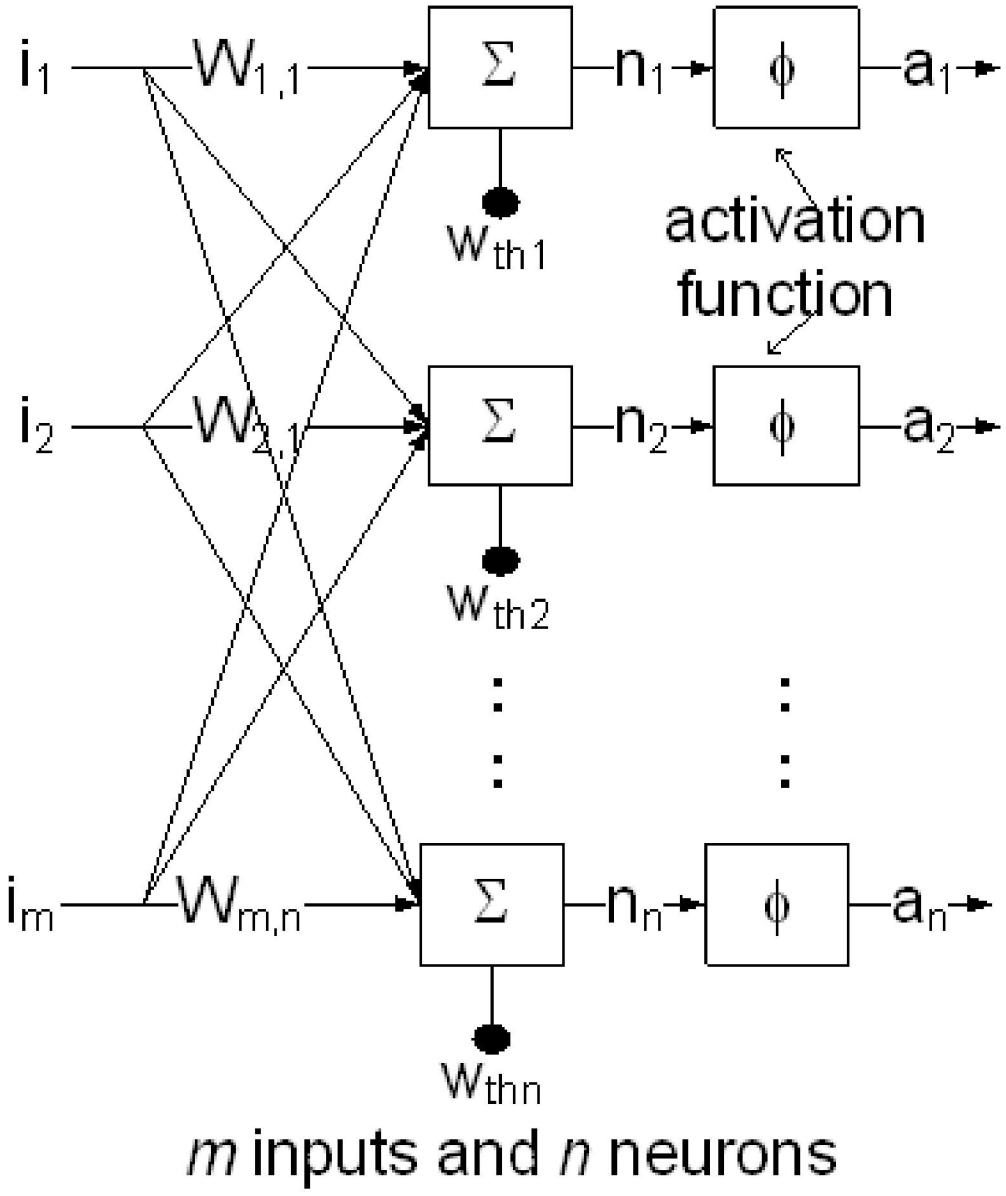
[41] "Brain-Maker User's Guide and Reference Manual," California Scientific Press, Nevada City, CA, 1998.

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2

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

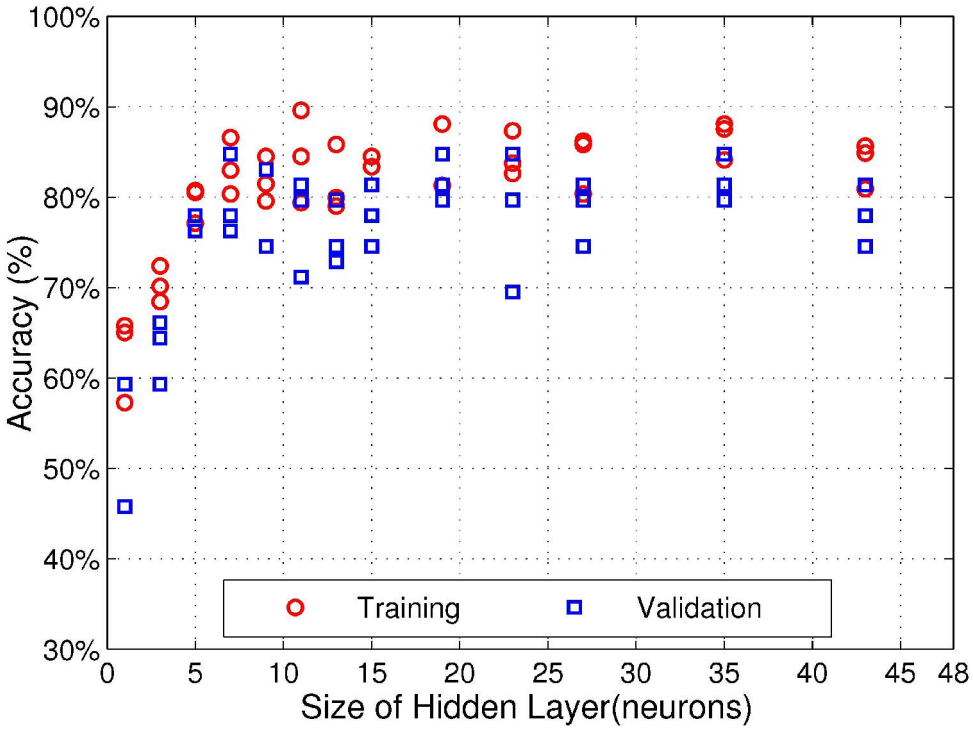
Parameter Type	**Input Neuron Description**	**Values**
Hardware	Load/store queue (instructions)	2, 4, 8, 16, 32, 64, 128
Hardware	Fetch queue width (instructions)	2, 4, 8, 16, 32, 64, 128
Hardware	Decode width (instructions)	1, 2, 4, 8, 16, 32, 64
Hardware	Issue width (instructions)	1, 2, 4, 8, 16, 32, 64
Hardware	Commit width (instructions)	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	T, NT, Perfect (a '*symbol*')
Hardware	Branch misprediction penalty	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

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2



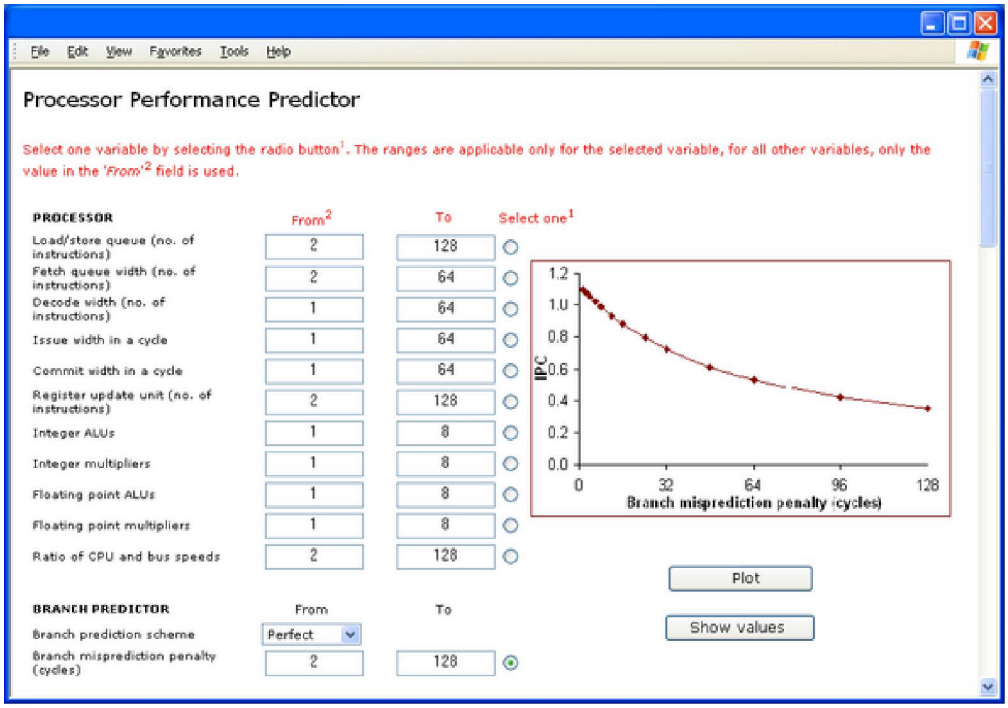
Neuron connections in one layer of a neural network
128x153mm (600 x 600 DPI)

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2



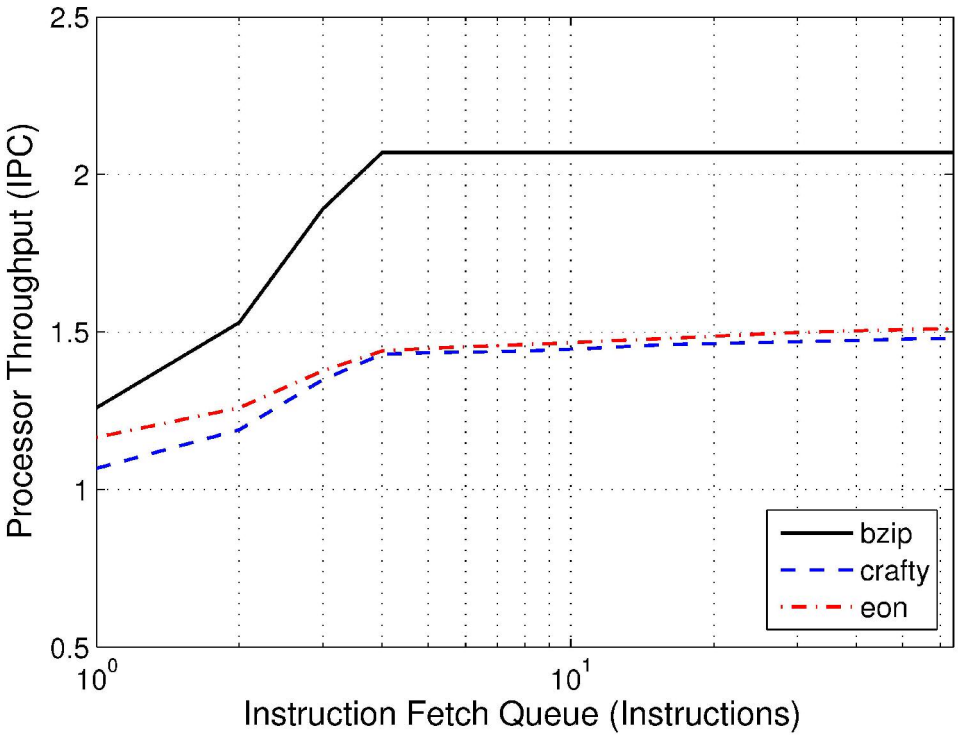
Training and validation accuracies as a function of the size of the single hidden layer in the neural network model
148x111mm (600 x 600 DPI)

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2



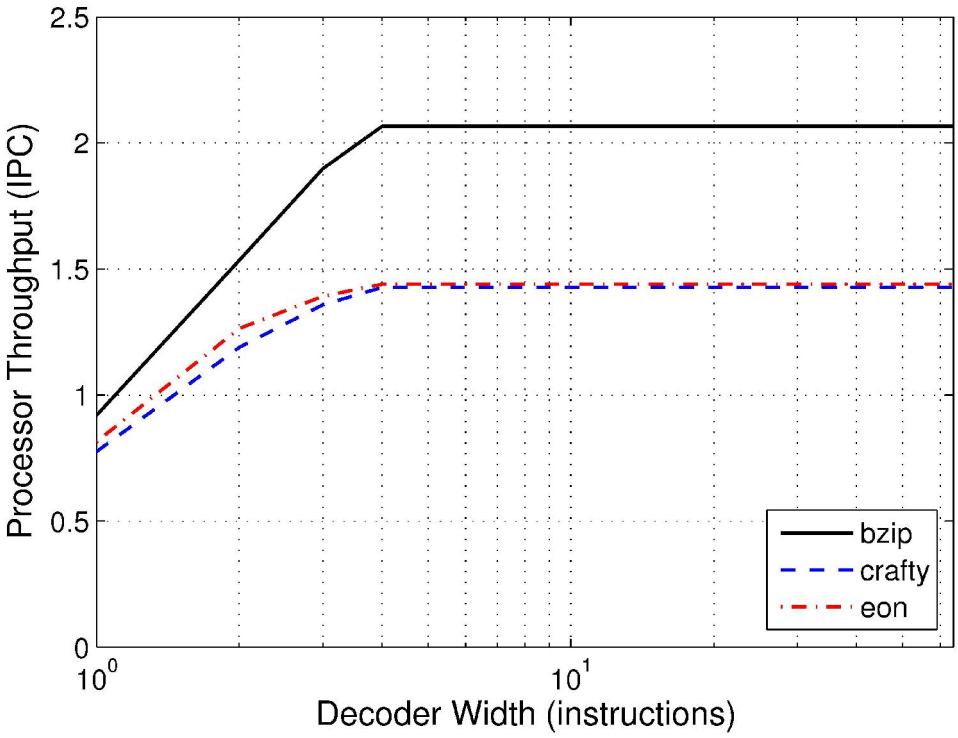
Web-based interface of PerfPred. In this example, branch misprediction penalty's effect on the system throughput is selected for modeling.
208x146mm (600 x 600 DPI)

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2



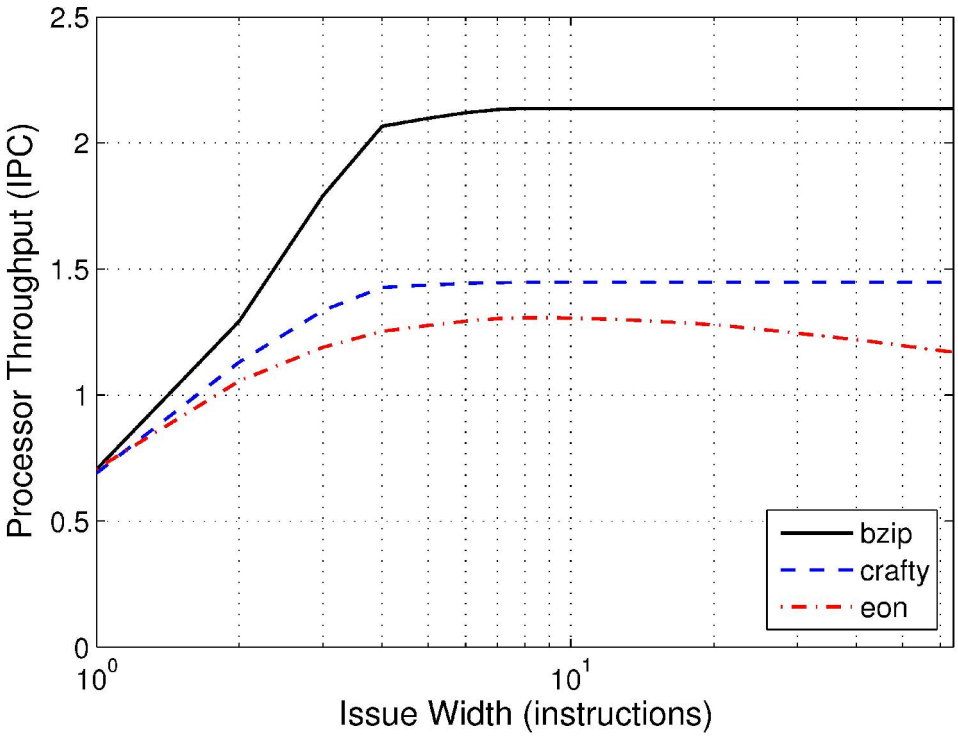
The impact of the instruction fetch queue on the processor throughput
148x111mm (600 x 600 DPI)

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2



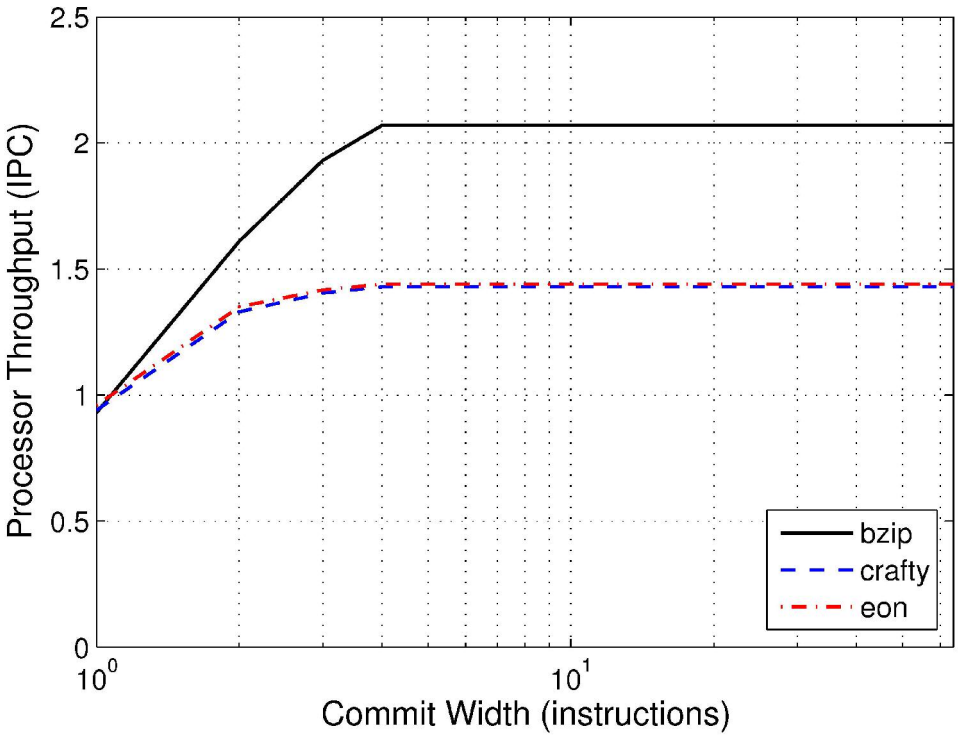
The impact of the decoder width on the processor throughput
148x111mm (600 x 600 DPI)

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2



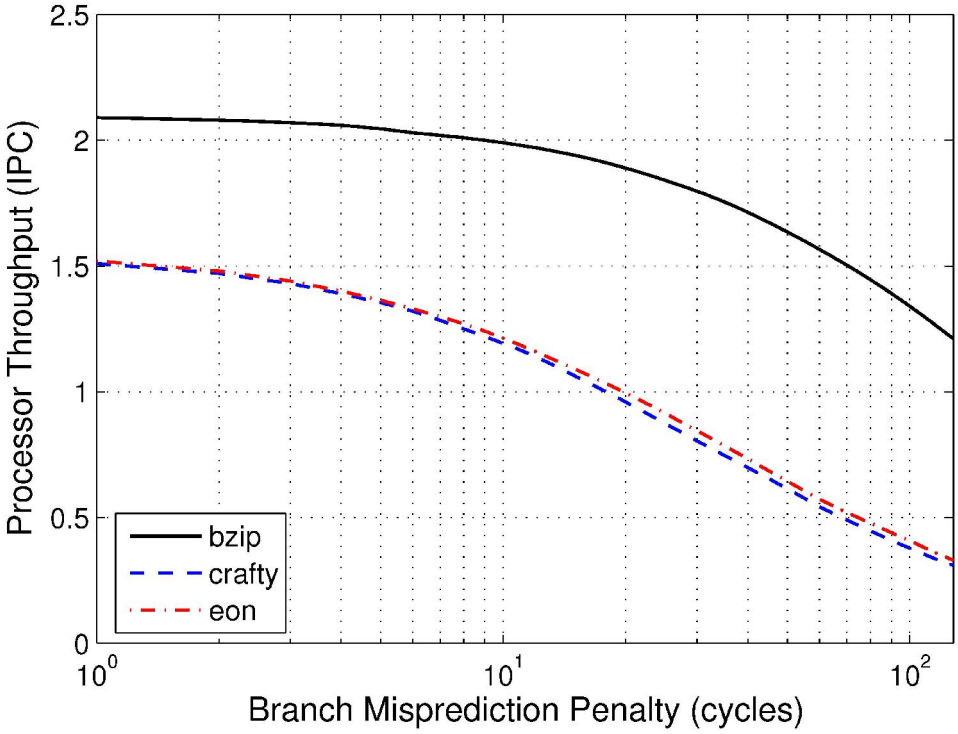
The impact of the issue width on the processor throughput
148x111mm (600 x 600 DPI)

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2



The impact of the commit width on the processor throughput
148x111mm (600 x 600 DPI)

A. Beg and W. Ibrahim, "PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space," Computer Applications In Engineering Education, Dec 2



The impact of the branch misprediction penalty on the processor throughput
148x111mm (600 x 600 DPI)