ACCEPTED: A. Beg, "Predicting Processor Performance with a Machine Learnt Model,"
in 50th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS/NEWCAS'07), Montreal, Canada, 2007, pp. 1098-1101.

Predicting Processor Performance with a Machine Learnt Model

Azam Beg
College of Information Technology
UAE University
Al-Ain, United Arab Emirates
abeg@uaeu.ac.ae

***Abstract*—Architectural simulators are traditionally used to study the design trade-offs for processor systems. The simulators are implemented in a high–level programming language or a hardware descriptive language, and are used to estimate the system performance prior to the hardware implementation. The simulations, however, may need to run for long periods of time for even a small set of design variations. In this paper, we propose a machine learnt (neural network/NN) model for estimating the execution performance of a superscalar processor. Multiple runs for the model are finished in less than a few milliseconds as compared to days or weeks required for simulation-based methods. The model is able to predict the execution throughput of a processor system with over 85% accuracy when tested with six SPEC2000 CPU integer benchmarks. The proposed model has possible applications in computer architecture research and teaching.**

I. Introduction & Previous Work

Hardware development is traditionally expedited using software models. The models are implemented in high level or hardware description languages. Relevant benchmark programs are then run on the models to get good approximations of actual hardware. Cycle-accurate simulators tend to be accurate but require weeks of simulation time with programs that run for a few billion cycles [1][2]. Noonburg and Shen [3] utilized the benchmark (program) traces to create a model for superscalar instruction level parallelism (ILP). They also made use of processor configurations to characterize the hardware. They combined the ILP and hardware parameters in their model. The prediction error with their models was as high as 22% for a few SPEC95 CPU benchmarks. Wallace and Bagherzadeh [1], and Hossain et al [2] presented models for conventional and trace cache behaviors, respectively. The analytical model by Hossain et al, due to its limited scope (only the caches, and not the full processor) had higher prediction accuracy, to be specific 7%. A processor system model needs to be inclusive of both the program and hardware behavior. The program behavior can be *static* or *dynamic*. The latter characterization can be done by capturing repeating patterns in a program [1]-[3]. Different parts of a program can be steady state or cyclical in nature [4]; this property of programs was used by Hamerly et al's simulation tool [5]. To speed up simulation, Wunderlich et al's approach statistically characterized the full-length benchmarks into smaller subsets. A recent processor model by Joseph et al [7] used detailed simulations to collect performance measures and then used radial basis functions to build a model as an alternative to simulations. Their model provided cycles per instruction (CPI) estimates with error ranges of 1.5%-12% for one of the SPEC2000 CPU benchmarks [8], and 1.5%-23% for another.

Artificial neural networks (NNs) are electronic equivalents of biological brains. The building blocks of NNs are simple processing entities called *neurons*. The neurons are interconnected to generate outputs in a parallel manner (as compared to the conventional sequential computers). Figure 1 shows a simple *feed-forward neural network* (FFNN) composed of three layers (input, hidden, and output) of neurons. The output of each layer only feeds the next layer and not any of the previous layers. Each neuron multiplies the inputs values with their respective weights. These values are then passed through an *activation function* (*sigmoid*, for example) to produce the final neuron output. The neuron weights are determined by the NN training process which involves presenting the NN with some known input examples (*training sets*). The weights are iteratively adjusted in a way that each set of inputs produces output(s) close to the example's pre-known output. An iteration of the weight-adjustment process is known as an *epoch*. Some known input-output sets (*validation sets*) are used for validating the NN prediction accuracy. The validation sets are not 'shown' to NN during the training process [9].

In this paper, we propose use of NNs for creating a prediction model for superscalar processor performance; the performance is measured in terms of *instructions completed per cycle* (IPC). The 'hardware' inputs to the model include key microarchitectural features such as fetch, decode, issue, and commit widths; number of integer and floating point

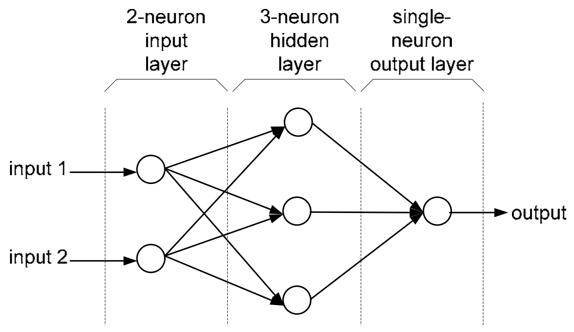

Figure 1. Structure of a simple feed-forward neural network that contains three layers of neurons, viz, input layer, hidden layer and an output layer. Output of one layer delivers an output only to the next layer and not to any previous layer, thus the name feed-forward neural network

ALUs; etc. The 'software' inputs can be representative of the dynamic nature of the programs; we selected a single such input, i.e., the average instructions per branch (easily acquired by running a single simulation of the given program). We utilized six different SPEC2000 CPU integer benchmark programs to acquire the data required for creating and validating the model.

II. Neural Network Data Acquisition

The NN model in our research emulates the behavior of a superscalar processor, i.e., SimpleScalar's *sim-outorder* architecture [10]. We used different configurations in *sim-outorder*'s 630 simulations (as listed in Table I) to collect the IPC data. The simulations for 6 different SPEC2000 CPU integer benchmarks (namely, *bzip2, crafty, eon, mcf, twolf,* and *vortex*) used *'test'* inputs [8]. We used simulator's fast-forwarding feature to reduce the effect of program initialization; first 10% of the program instructions were fast-forwarded. Maximum number of simulation cycles was set at 500 million instructions to finish the simulations in a reasonable amount of time [1][2].

We used a PERL script to create the command lines for 310 *sim-outorder* command lines. The script randomly selected from the parameter values listed in Table I. A sample command line for *crafty* benchmark is shown here:

```
sim-outorder -fastfwd 50000000 -max:inst
500000000 -cache:il1 il1:64:32:4:t
-cache:il2 il2:64:32:16:l -cache:dl1
dl1:4:8:4:f -cache:dl2 dl2:16:8:4:l
-cache:dl1lat 3 -cache:dl2lat 6
-cache:il1lat 2 -cache:il2lat 8 -tlb:itlb
itlb:256:16:2:t -tlb:dtlb none -tlb:lat 30
-mem:lat 16 2 -mem:width 16 -decode:width 4
-issue:width 8 -commit:width 4 -ruu:size 16
-lsq:size 8 -fetch:ifqsize 8 -fetch:speed 8
-fetch:mplat 6 -res:ialu 3 -res:imult 7
-res:fpalu 1 -res:fpmult 7 -bpred nottaken
crafty00.peak.ev6
```

The remaining (320) command lines varied a single parameter over its entire range, for example, the following two command lines for *bzip* benchmark use one and two decoders, respectively. All other parameters were left as *sim-outorder* defaults:

```
sim-outorder -fastfwd 50000000 -max:inst
500000000 -decode:width 1 bzip200.peak.ev6
input.random 2
```

```
sim-outorder -fastfwd 50000000 -max:inst
500000000 -decode:width 2 bzip200.peak.ev6
input.random 2
```

Sim-outorder simulation results were saved into text-based log files and were later parsed using another PERL script. The simulations were run on multiple x86-machines running *cygwin* (a UNIX emulator) under Windows-XP. Each of the 630 simulations lasted 2-2.5 hours.

III. Neural Network Structure and Training

Each input or output of the NN model corresponds to a single neuron. So IPC prediction needed only a single output neuron. There were 12 inputs to the NN, meaning there were 12 neurons in the input layer, i.e., 11 neurons were for hardware parameter representation, and one for software/ benchmark program. Each program was represented by its dynamic property called *average instructions per branch* (*sim-outorder* parameter *sim_IPB*); this parameter has been used in the past to represent programs [1][2]. The non-numerical value of 'branch prediction scheme' parameter was input to the NN as a symbol.

It is well known that a NN with a single hidden layer is able to model most non-linear systems, so we limited our experiments to NNs that comprised only one hidden layer of neurons.

Our experiments involved multiple NN configurations, mainly affecting the number of neurons in the network's hidden layer. As mentioned earlier, the number of inputs and outputs were fixed, so the corresponding neuron counts did not vary. We used MS-Windows-based Brain-Maker (version 3.75) [11] tool to model our *back-propagation* FFNNs. In Table I, we can notice the non-linearity of input parameter values. This wide variation in the inputs can adversely affect the learn-ability of a NN. So log_2 transform was applied to the non-linear parameters as a data *pre-processing* measure [11].

NN training was performed with the training set that comprised 90% of full data set. The other 10% data sets were used for validation. In order to investigate a somewhat wider range of NN topologies, the hidden layer size was varied between 1-45 neurons. Each experiment was repeated multiple times with random initial weights, in order to alleviate the possibility of local minima.

Percentages of correctly learnt data sets (whose accuracy fell within the desired ranges) are plotted in Figure 2. The accuracies start from their low values of 45.8% and go higher as neurons are added to the hidden layer. Maximum validation accuracy of 84.7% was attained with a relatively small hidden layer of only 7 neurons. For the same configuration, the training accuracy was 86.6%. A plausible reason for having such a small size of hidden layer produce the highest validation accuracy is: in the current models, all 13 microarchitectural features are being used as inputs; it is possible that not all of these inputs are contributing equitably to the outputs. Our observation of the neuron-weight matrices does not provide sufficient clues; some input-significance analysis may provide more insight into this phenomenon. Studying the degree of significance/contribution of each of the inputs to the model is a topic of our continued research.

TABLE I. HARDWARE (MICROARCHITECTURAL) PARAMETERS USED TO RUN SIMULATIONS WHICH ARE IN TURN USED TO ACQUIRE THE DATA FOR THE NN MODEL.

Description	Range
Load/store queue (instrs.)	2, 4, 8, 16, 32, 64, 128
Fetch queue width (instrs.)	2, 4, 8, 16, 32, 64, 128
Decode width (instrs.)	1, 2, 4, 8, 16, 32, 64
Issue width in a cycle	1, 2, 4, 8, 16, 32, 64
Commit width in a cycle	1, 2, 4, 8, 16, 32, 64
Register update unit (instrs.)	2, 4, 8, 16, 32, 64, 128
Ratio of CPU and bus speeds	2, 4, 8, 16, 32, 64, 128
Integer ALUs	1, 2, 3, 4, 5, 6, 7, 8
Integer multipliers	1, 2, 3, 4, 5, 6, 7, 8
Branch prediction scheme	Taken, Not-taken, Perfect
Branch misprediction penalty (cycles)	1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128

IV. PERFORMANCE PREDICTION WITH THE NEURAL NETWORK MODEL

The proposed NN model can be employed to investigate how different design parameters, for example, issue width, instruction decoder count, number of integer ALUs, etc., influence the processor performance. First, we present the example of varying branch misprediction penalty (i.e., the number of cycles wasted due to a mispredicted branch). Figure 3 shows the behavior of a processor in response to branch penalty, for 4 different benchmarks. The penalty ranges from 1 to 128. As expected, the longer it takes to recover from a branch misprediction, the lower the throughput (IPC) of the processor. We notice that the IPC curves for all the benchmarks are similar in shape. However, the dynamic nature of a program has a considerable effect on IPC's absolute values. *bzip2* exhibits a larger IPC over the complete range of branch penalties than the other 3 benchmarks (*crafty, eon*, and *mcf*). A detailed look into the benchmark characteristics may be needed to explain this difference in program-dependent performance metrics.

In Figure 4, we show the effect of varying issue width, i.e., the number of instructions issued per cycle. The issue width in this example varies from 1 to 32 for three different benchmarks, namely, *bzip, crafty* and *eon.* We notice that the overall processor throughput increases with larger issue width but flattens out around 8 decoders. This upper limit on decoder hardware can be attributed to the limited parallelism in the programs.

The point worth noting here is the speed with which the processor performance can be estimated by using our NN model. The two examples (presented above), if ran on a traditional cycle-accurate simulator would have required a few computer-days. Whereas, the NN model produces the same results in less than a few milliseconds. Our model's speed efficiency can be helpful for researchers or students alike. If used in academic environment, the models, in a very short span of time, would allow students to study a much

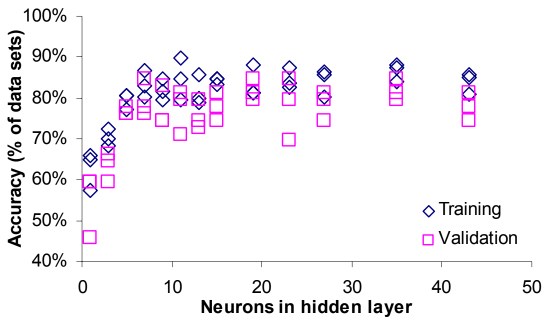

Figure 2. Training and validation accuracy (% of facts learnt correctly) as a function of hidden layer neuron count. (Brain-Maker training parameters: Training/testing tolerance = 0.15; learning rate adjustment type = heuristic. Initial neuron weights set randomly). Note that the final neuron weights and consequently the NN's post-training predictions, are dependent on the initial values selected by Brain-Maker (randomly).

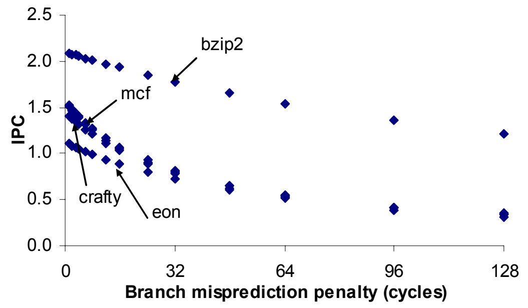

Figure 3. Sensitivity of IPC due to branch misprediction penalty. The more the cycles spent after a misprediction, the lower the processor throughput (IPC).

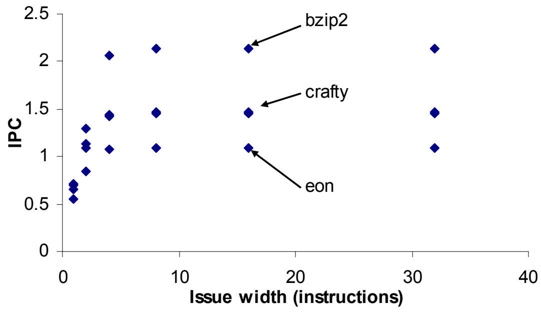

Figure 4. Sensitivity of IPC due to issue width. Issuing more than 8 instruction 8 in every cycle does not result in furthering the processor throughput (IPC).

wider range of processor system design space than if they were to use a detailed simulator.

V. CONCLUSIONS

Using the NN model proposed in this research, one can estimate the performance of a processor without requiring lengthy simulations. For prediction purposes, the model incorporates both the microarchitectural features of a processor and the program characteristics. The model can quickly estimate a processor system configuration to match the desired performance. This step can be followed by detailed cycle-accurate simulations. The model can also find application in the teaching of computer architecture courses. As a continuation of the current research, we are looking into: (1) the selection criteria for microarchitectural parameters based on their significance to the NN model, and (2) extending the choice of software parameters that would characterize the dynamic nature of the programs more accurately.

REFERENCES

[1] S. Wallace and N. Bagherzadeh, "Modeled and Measured Instruction Fetching Performance for Superscalar Microprocessors," IEEE Trans. on Parallel and Distrib. Syst., Vol. 9, No. 6, Jun. 1998, pp. 570-578.

[2] A. Hossain, D. J. Pease, J. S. Burns, and N. Parveen, "A Mathematical Model of Trace Cache," Proc. IEEE Inter. Conf. Appl. Specific Syst., Archit. (ASAP'02), San Jose, CA, USA, Jul. 2002.

[3] D. B. Noonburg and J. P. Shen, "Theoretical Modeling of Superscalar Processor Performance," Proc. Annual Inter. Symp. on Microarch. (MICRO-27), San Jose, CA, USA, Nov. 1994.

[4] S. Dhodapkar and J. E. Smith, "Comparing Program Phase Detection Techniques," In Proceedings of Int'l Symposium on Micro-architecture (MICRO-36), San Diego, CA, USA, Dec. 2003.

[5] G. Hamerly, E. Perelman, J. Lau, B. Calder, "Simpoint 3.0: Faster and More Flexible Program Analysis," Journal of Instruction Level Parallelism (JILP), (website) http://www.cse.ucsd.edu/~calder/papers/JILP-05-SimPoint3.pdf, 2005.

[6] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, J. C. Hoe, "SMARTS: Accelerating Micro-architecture Simulation via Rigorous Statistical Sampling," Proc. of Inter. Symp. Comp. Arch. (ISCA 2003), San Francisco, CA, USA, Jun. 2003.

[7] P. J. Joseph, K. Vaswani, M. J. Thazhuthaveetil, "A Predictive Performance Model for Superscalar Processors," Proc. Annual IEEE/ACM Inter. Symp. on Microarch. (MICRO'06), Orlando, FL, USA, Dec. 2006.

[8] SPEC2000 CPU benchmarks, (website) http://www.spec.org/cpu2000/.

[9] T. Mitchell, Machine Learning, McGraw Hill Co., Columbus, OH, USA, 1997.

[10] SimpleScalar 3.0d (simulation tool), (website) http://www.simplescalar.com.

[11] Brain-Maker User's Guide and Reference Manual, 7th ed. California Scientific Software Press, 1998.