Incorporating Program Characteristics into a Processor Model

Azam Beg

Abstract – **Architectural simulators used for microprocessor design study and optimization can require large amount computational time and/or resources. In such cases, models can be a fast alternative to lengthy simulations, and can help reach a designer near-optimal system configuration. However, The non-linear characteristics of a processor system make the modeling task quite challenging. The models not only need to incorporate the micro-architectural parameters but also the dynamic behavior of programs. This paper presents a hybrid (hardware/software), non-linear model for processors. The model provides accurate predictions of processor throughput for a wide range of design space.**

Index Terms – **Processor throughput, instructions per cycle (IPC), processor model, micro-architecture simulation, basic blocks.**

I. Introduction

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. A processor system model needs to be inclusive of both the program and hardware behavior. The program behavior can be *dynamic* or *static*. The dynamic characterization can be done by capturing repeating patterns in a program [1]-[3]. Cycle-accurate simulators tend to be accurate but require weeks of simulation time with programs running 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 combined the ILP and hardware parameters in their model. The prediction error with their models was as high as 22% for some of the SPEC CPU95 benchmarks[4]. Wallace and Bagherzadeh [1], and Hossain et al. [2] presented models for conventional and trace caches, 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 – 7% to be specific. Different parts of a program can be steady state or cyclical in nature [5]; this property of programs was exploited in Hamerly et al.'s simulation tool 0. To speed up simulation, Wunderlich et al. [8] statistically characterized the full-length benchmarks into smaller subsets. Joseph et al. [9] collected performance measures from detailed simulations and then used radial basis functions (RBFs) 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 SPEC CPU2000 benchmarks [10], and 1.5%-23% for another. Lee et al.'s simulator [11] identified basic blocks that repeated often; it then used block behavior within a combination of fast (*sim-cache*) and slow (*sim-outorder*) to speed up the simulations by a factor of 3.3. A similar method of capturing the dynamic nature of a program is to detect its recurring patterns (called *program phases*). This approach requires one to pick the appropriate granularity for phase detection and the time for capturing the phases for unique characterization of a program [12]-[15]. Beg [16], and Beg and Ibrahim [17] presented machine-learnt models for predictor processor system performance. Their models used a wider range of hardware (processor and memory) parameters than Joseph et al.'s [9] predictive RBF. In [17], the authors also proposed that the models be used as a tool for computer architecture pedagogy. These models characterized the complete program trace with a single variable – a somewhat limited representation of a program's dynamic behavior.

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 fashion (as compared to the conventional sequential computers). A simple *feed-forward neural network* (FFNN) composed of layers of neurons: input, hidden, and output. The outputs of each layer only feed the next layer and not any of the previous layers. The neurons multiply their inputs values with their respective *weights*, before passing them through an *activation function* (such as *sigmoid*) to produce the final neuron output. The neuron weights are determined by training the NNs with some known input examples (*training sets*). The weights are iteratively adjusted in such a way that each set of inputs produces output(s) close to the example's pre-known output(s). An iteration of the weight-tuning 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 training [18].

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

Azam Beg is with the College of Information Technology, United Arab Emirates (UAE) University, P O Box 17551, Al-Maqam Campus, Building 22, Al-Ain, UAE (phone: +971-3-713-5563; fax: +971-3-767-2018; email: abeg@uaeu.ac.ae).

etc. The 'software' inputs represent the dynamic nature of the programs – sets of basic block frequencies (instead of single input, the average block size [16][17]). Inclusion of multiple software parameters helps the user investigate more accurately how the dynamic nature of a program affects the processor performance.

II. Data Acquisition & Experimental Setup

The NN model in our research emulates the behavior of a superscalar processor, i.e., SimpleScalar's *sim-outorder* architecture [19]. We used different configurations in *sim-outorder*'s 630 simulations (as listed in Table 1) to collect the IPC data. The simulations for 6 different SPEC CPU2000 integer benchmarks (namely, *bzip2*, *crafty*, *eon*, *mcf*, *twolf*, and *vortex*) used *'test'* inputs [8]. To reduce the effect of program initialization, we *fast-forwarded* the simulations by 50 million cycles [19]. We limited each run to 500 million instructions in order to complete all simulations in a reasonable amount of time [1][2]. 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.

First, we created 310 *sim-outorder* command lines by randomly selecting the parameter values listed in Table 1. One such 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 other 320 command lines were created by varying at a time, a single parameter over its entire range, while all other parameters were kept at their default values. For example, the following command lines for *bzip* benchmark use decoder widths of 1, 2, 4, and 8:

```
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 -fastfwd 50000000 -max:inst
500000000 -decode:width 4 bzip200.peak.ev6
input.random 2

sim-outorder -fastfwd 50000000 -max:inst
500000000 -decode:width 8 bzip200.peak.ev6
input.random 2
```

Besides containing hardware-related data, *sim-outorder*'s text-based log files also included basic block data required for

Table 1. Hardware (microarchitectural) and software parameters used in NN models

No.	Input Parameter Type	Description	Range/Values
1	Hardware	Load/store queue (instrs.)	2, 4, 8, 16, 32, 64, 128
2	Hardware	Fetch queue width (instrs.)	2, 4, 8, 16, 32, 64, 128
3	Hardware	Decode width (instrs.)	1, 2, 4, 8, 16, 32, 64
4	Hardware	Issue width (instrs.)	1, 2, 4, 8, 16, 32, 64
5	Hardware	Commit width (instrs.)	1, 2, 4, 8, 16, 32, 64
6	Hardware	Register update unit (instrs.)	2, 4, 8, 16, 32, 64, 128
7	Hardware	Ratio of CPU and bus speeds	2, 4, 8, 16, 32, 64, 128
8	Hardware	Integer ALUs	1, 2, 3, 4, 5, 6, 7, 8
9	Hardware	Integer multipliers	1, 2, 3, 4, 5, 6, 7, 8
10	Hardware	Branch prediction scheme	'Taken,' 'Not-taken,' 'Perfect' (*symbols*)
11	Hardware	Branch misprediction penalty (cycles)	1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128
12	Software	No. of blocks containing a single instruction	1-100 (percentage)
13	Software	No. of blocks containing 2 instructions	1-100 (percentage)
14	Software	No. of blocks containing 3 instructions	1-100 (percentage)
15	Software	No. of blocks containing 4 instructions	1-100 (percentage)
16	Software	No. of blocks containing 5 or more instructions	1-100 (percentage)

program characterization. We experimented with different groups (histograms) of basic block occurrences, but found the groups shown in Fig. 1 to be the most appropriate for NN model development.

III. Neural Network Structure and Training

Fig. 2 shows a simple NN with software and hardware inputs. Each of the 16 inputs (not shown in the figure) of the NN model corresponds to a single neuron; 11 neurons are for hardware representation, and 5 for program/software. The non-numerical 'branch prediction scheme' (hardware) parameter is input to the NN as a *symbol*. A single 'continuous' neuron in the output layer is sufficient for NN's IPC predictions.

It is well-known fact that a NN with a single hidden layer is able to model most non-linear systems, so we limited our experiments to one-hidden layer NNs. Additionally, in Table 1, we can notice the non-linearity of input parameter values. This logarithmic nature of the inputs can adversely affect the learn-ability of a NN, so we applied log_2 transformed such inputs, as a data *pre-processing* step. We used MS-Windows-based Brain-Maker (version 3.75) [20] tool to model our *back-propagation* FFNNs. NN training was performed with the training set that comprised of 90% of full data set. The remaining 10% data sets were used for validation purposes. For every NN configuration, we conducted 3 or more training sessions completely independent of each other, in order to find the best performing NN, and to avoid the possibility of local minima.

As we can see in Fig. 3, the validation accuracy stayed in sync with the training accuracy thus demonstrating the health of the NN. Maximum validation accuracy of 85% was achieved with a hidden layer as small as 7 neurons. Although the same accuracy was attained by larger hidden layers of 17 and 23 neurons, we chose the smallest NN with the use cases discussed shortly. A small number of hidden neurons producing the high validation accuracy may mean that not all 16 hardware and software inputs are contributing equitably to the outputs. Although the examination of the neuron-weight matrices does not provide sufficient clues, the input-significance analysis may provide more insight into this phenomenon – a topic of our continued research.

IV. Performance Prediction with the Neural Network Model

The proposed NN model can be used to investigate how different hardware and software parameters influence the processor performance. Firstly, Fig. 4 shows the effect of varying decoder width, i.e., the number of instructions decoded per cycle. The decoder width in this example varies from 1 to 64 for three different benchmarks. All three benchmarks flatten out after 4-wide decoder unit. Secondly, Fig. 5 shows the IPC behavior in response to commit unit variations. The unit shows maximum throughput with 8-instruction width. Further runs using the model, one can investigate which hardware elements (ALU, issue width, etc.) or the software characteristics (parallelism in a program, branch frequencies, etc.) are the limiting factors in processor performance.

To illustrate how the dynamic nature of a program – represented by basic block distribution – affects the execution performance, we created two artificial program traces, T1 and T2 (Fig. 6). T1 has small percentages of 1 and 2-instruction blocks and much larger percentage of larger (5 or more instructions) blocks. Such a trace may come from programs in which loop unrolling has been done. T2's distribution is just the opposite of T1, i.e., there are more blocks of smaller sizes (1 or 2 instructions) and the smaller number of blocks are of larger sizes (5 or more instructions). When the block frequencies of T1 and T2 are input to the NN model, we observe that T1 has 33% lower IPC than T2; this may be due to the small cache (number of sets and line size) used in the experiment; smaller caches can cause thrashing and thus lower the execution throughput. On the other hand, T2's smaller blocks may have better chances of fitting in cache lines making the fetch stage efficient and thus improving the overall execution throughput.

A point worth noting: the examples presented above, if ran on a traditional detailed, cycle-accurate simulator would have required several computer-days. Whereas, the NN model produces the same results almost instantly. The model's speed efficiency can be helpful for researchers and students alike.

V. Conclusions

Using the NN model proposed in this research, one can estimate the performance of a processor without requiring lengthy simulations. The model incorporates both the microarchitectural features of a processor and the program characteristics. So a designer can quickly make educated guesses about the micro-architectural parameters for optimum hardware performance, while a compiler designer can speedily observe the effects of code characteristics on the overall execution throughput. The model can also find application in the teaching of computer architecture and compiler design. 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, (2) extending the choice of software parameters that would characterize the dynamic nature of the programs more accurately, and (3) creating a multi-processor prediction model.

References

[1] S. Wallace and N. Bagherzadeh, "Modeled and Measured Instruction Fetching Performance for Superscalar Microprocessors," IEEE Trans. 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. Inter. Symp. Microarch. (MICRO-27), San Jose, CA, USA, Nov. 1994.
[4] SPEC CPU95 benchmarks, (website) http://www.spec.org/cpu95/.
[5] S. Dhodapkar and J. E. Smith, "Comparing Program Phase Detection Techniques," Proc. Inter. Symp. Microarch. (MICRO-36), San Diego, CA, USA, Dec. 2003.

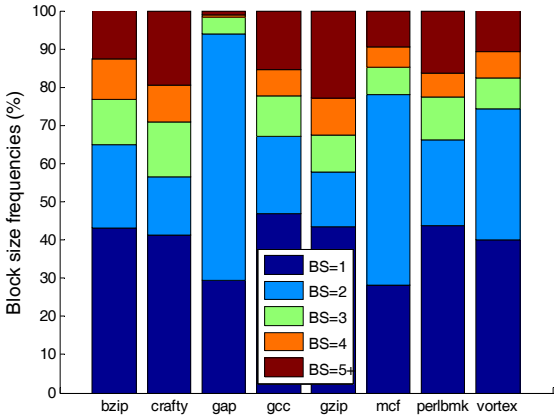

Fig. 1. Basic block sizes (BS) in SPEC CPU2000 integer benchmarks. 'BS=1' represents the percentage of blocks of size 1, 'BS=2' represents blocks of size 2, and so on. And 'BS=5+' represents the blocks of size 5 or more.

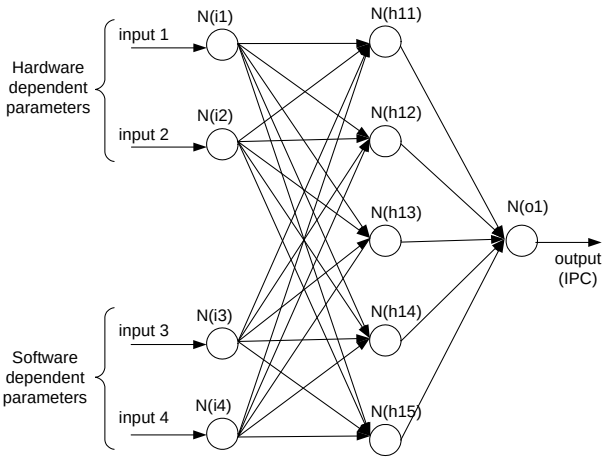

Fig. 2. Structure of a simple feed-forward neural network that contains three layers of neurons, viz, input layer, hidden layer and an output layer.

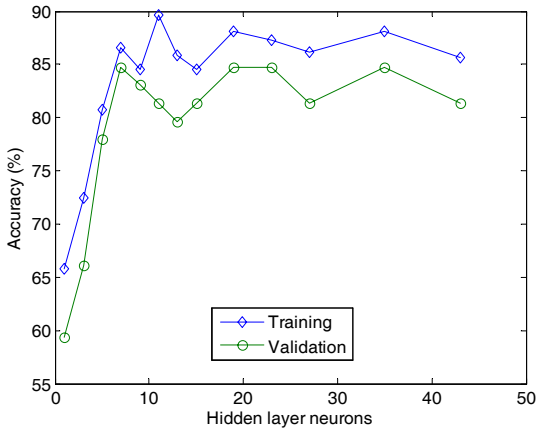

Fig. 3. Accuracy as a function of the hidden layer size. Best validation accuracy was observed for 7, 17, and 23 hidden neurons. We chose the smallest of the three NNs for running purposes.

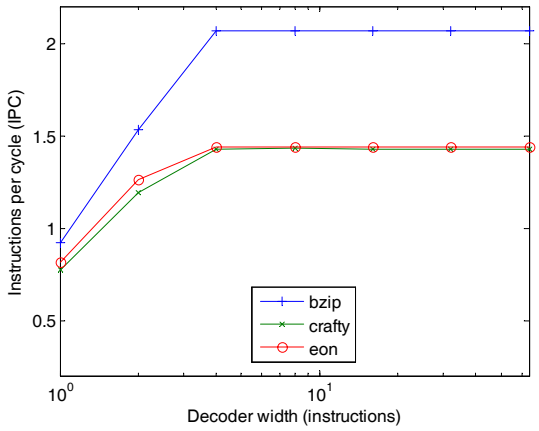

Fig. 4. Sensitivity of IPC to decoder width, while all other parameters are set to sim-outorder defaults. As little as 4 bytes are able to produce the best IPC.

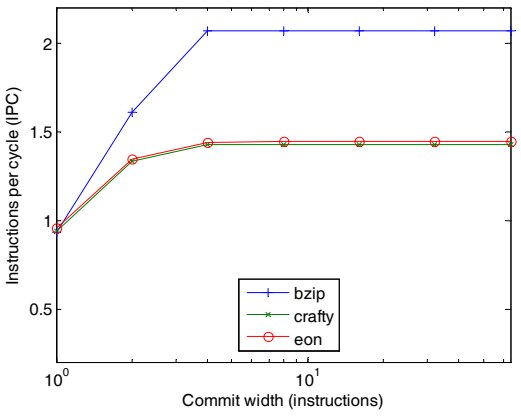

Fig. 5. Sensitivity of IPC to commit width, while all other parameters are set to *sim-outorder* defaults. Only a 4 instruction-wide commit unit is sufficient under the given conditions.

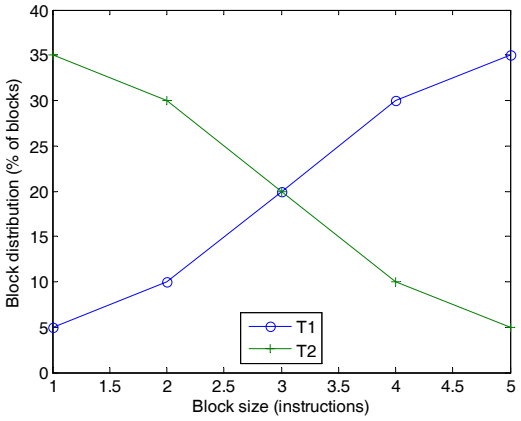

Fig. 6. Traces with different block distributions exhibit different IPCs, while all hardware parameters are set to *sim-outorder* defaults. IPC for T1 is 1.15 and IPC for T2 is 1.30.

ISBN: 978-988-98671-0-2

[6] G. Hamerly, E. Perelman, J. Lau, B. Calder, “Simpoint 3.0: Faster and More Flexible Program Analysis,” J. Instr. Level Parallelism (JILP), (website) http://www.cse.ucsd.edu/~calder/papers/JILP-05-
[7] SimPoint3.pdf, 2005.
[8] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, “SMARTS: Accelerating Micro-architecture Simulation via Rigorous Statistical Sampling," Proc. Inter. Symp. Comp. Arch. (ISCA 2003), San Francisco, CA, USA, Jun. 2003.
[9] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "A Predictive Performance Model for Superscalar Processors," Proc. IEEE/ACM Inter. Symp. Microarch. (MICRO'06), Orlando, FL, USA, Dec. 2006.
[10] SPEC CPU2000 benchmarks, (website) http://www.spec.org/cpu2000/.
[11] W. Lee, K. Patel, and M. Pedram, "B2Sim: A Fast Micro-Architecture Simulator Based on Basic Block Characterization", Proc. IEEE/ACM/IFIP Conf. Hardware/Software Codesign and Syst. Synthesis (CODES+ISSS'06), Seoul, S. Korea, Oct. 2006, pp. 199-204.
[12] T. Sherwood and B. Calder, “Time Varying Behavior of Programs,” UCSD Technical Reports, (website) http://www-cse.ucsd.edu/~calder/papers/UCSD-CS99-630.pdf, Aug. 1999.
[13] T. Sherwood, E. Perelman, and B. Calder, “Basic Block Distribution Analysis to Find Periodic Behaviors and Simulation Points in Applications,” Proc. Inter. Conf. Parallel Microarch. and Compilation Techniques (PACT-2001), Barcelona, Spain, Sep. 2001, pp. 3-14.
[14] T. Sherwood, S. Sair, and B. Calder., “Phase Tracking and Prediction,” Proc. Inter. Symp. Comp. Arch. (ISCA 2003), San Francisco, CA, USA, Jun. 2003, pp. 336-347.
[15] A. S. Dhodapkar and J. E. Smith, “Comparing Program Phase Detection Techniques,” Proc. Inter. Symp. Micro-arch. (MICRO-36), San Diego, CA, USA, Dec. 2003. pp. 217-227.
[16] A. Beg, “Predicting Processor Performance with a Machine Learnt Model,” Proc. 50th IEEE Inter. Midwest Symp. Circuits and Syst., (MWSCAS/NEWCAS'07), Montreal, Canada, Aug. 5- 8, 2007, pp. 1098-1101.
[17] A. Beg and W. Ibrahim, “PerfPred: A Web-Based Tool for Exploring Computer Architecture Design Space,” Comp. Applications In Eng. Educ., 2008 (In Press).
[18] T. Mitchell, Machine Learning, McGraw Hill Co., Columbus, OH, USA, 1997.
[19] SimpleScalar LLC, (website) http://www.simplescalar.com.
[20] Brain-Maker User’s Guide and Reference Manual, 7th ed. California Scientific Software Press, 1998.