Instruction-level Timing Error Prediction Model for 5-stage Pipelined ALPHA Processor

Yuanbo Fan, Chao Yan
Department of Electrical Engineering and Computer Science
Northwestern University
{yuanbo, chaoyan2012}@u.northwestern.edu

ABSTRACT

In this project, we build an instruction-level timing error prediction model for a 5-stage pipelined ALPHA processor which supports timing speculation. This model predicts if an instruction produces a timing error given the information of instruction sequence and data usage.

Keywords. Computer Architecture, Timing Speculation, Timing Error, Timing Error Prediction

1. INTRODUCTION

Timing error is defined as a violation in circuit-level timing constraints during program execution. In traditional processors, timing error may cause catostrophic system failure. Processors which support timing speculation are augmented with timing-error detection and correction techniques so that they able to recover from timing errors.

However, the recovery cost is very high. Also, based on some existing work, for a specific processor, timing error is a strong function of programs. Therefore, an effective timing-error prediction model can be built based on the information of how the programs are executed on the processor. Then, the compiler can apply this model to generate codes that have lower probability of producing timing errors by using instruction scheduling and instruction selection. Moreover, if the model is implemented in the hardware, it can predict timing error at run-time. With some padding techniques, it can avoid the full recovery cost due to timing errors.

In this project, we use machine learning algorithms to build an instruction-level timing error prediction model for a 5-stage pipelined ALPHA processor which supports timing speculation. This model predicts if an instruction has timing error given the information of instruction sequence and data usage.

2. LEARNING PROCESS

2.1 Dataset

Table 1: Description of test programs

Name	Descriptin
copy	copy memory contents of N elements
objsort	object sorting algorithm
parsort	parsort algorithm
fib	compute Nth fibonacci number
fib_rec	compute Nth fibonacci number recursively
evens	compute even numbers that are less than N
insertion	insertion sorting
parallel	compute first N multiple of M
sort	bubble sorting algorithm
saxpy	integer SAXPY

We use ten instruction-level test programs to build ten test sets. In each set, we select nine programs for training and the other one for testing. Then, the programs are executed on a 5-stage pipelined ALPHA processor. During the execution, we collect the information for each instruction including the opcode, two operand values and some control signals which indicate the data dependence and branch information. Also, a threshold delay is set, so we can label each instruction by if it produces a timing error.

```
/*
   TEST PROGRAM #3: compute first 16 fibonacci numbers
            with forwarding and stall conditions in the loop

   long output[16];

   void
   main(void)
   {
     long i, fib;

     output[0] = 1;
     output[1] = 2;
     for (i=2; i < 16; i++)
       output[i] = output[i-1] + output[i-2];
   }
*/

   data = 0x1000
   lda     $r3,data
   lda     $r4,data+8
   lda     $r5,data+16
   lda     $r9,2
   lda     $r1,1
   stq     $r1,0($r3)
   stq $r1,0($r4)
loop:   ldq     $r1,0($r3)
   ldq     $r2,0($r4)
   addq    $r2,$r1,$r2
   addq    $r3,0x8,$r3
   addq    $r4,0x8,$r4
   addq    $r9,0x1,$r9
   cmple   $r9,0xff,$r10
   stq     $r2,0($r5)
   addq    $r5,0x8,$r5
   bne     $r10,loop
   call_pal        0x555
```

Figure 1: An example of instruction-level programs

Each input instance in the training and testing set consists of several parts, including the current instruction and N-1 preceding instructions. N is defined as the instruction win-

Table 2: An example of collected instruction information

Opcode and Two Operands	Control Signal	Timing Error
ldq r3 0	0100	True
ldq r4 0	0001	False
addq r1 r2	0010	False
addq 0x8 r3	0010	True
addq 0x8 r4	0010	False
addq 0x1 r9	0010	True
cmple 0xff r9	0100	False
...	...	...

Table 3: An example of instances

Opcode	ra	rb	c0	c1	c2	Branch	Error
8	0	0	1	0	0	0	0
8	4095	0	1	0	0	0	0
8	4096	0	1	0	0	0	0
19	0	10	0	1	0	0	0
45	0	4096	1	3	2	0	0
41	0	4096	1	0	0	0	1
45	256	4096	1	0	1	0	0
16	4096	8	0	1	0	0	0
16	0	1	0	1	0	0	0
16	1	4095	5	0	0	0	0
61	44	4294967264	2	2	2	1	1
19	1	10	0	1	0	0	0
...	...	...	...	...	...	...	...

dow size which means the number of instructions which have effect on if the current instruction produces a timing error. Each instruction contains an opcode and two operands. For each operand value used in the instructions, we use both number- and bit- representation, because we think different representation may result in different models, specially while using Multilayer Perceptron algorithm. Table 3 shows an example of instances when the instruction window size is 1. Each opcode is encoded into a number, and operand values are in the format of number representation. There are three numbers (c0, c1 and c2) representing control signals and a binary number which indicates if it is a taken branch. The last number represents if there is a timing error.

2.2 Learning Algorithms

We use both J48 Decision Tree and Multilayer Perceptron classifier in the WEKA packages to build the model. The results and comparison will be shown in next section.

2.3 Software Packages

We use Synopsys tools to simulate the gate-level execution of programs and the WEKA to run machine learning algorithms on the dataset.

2.4 Validation

In each set, nine programs are used for training, while the other one is used for testing. Thus, the learned model can be validated by using 10-fold cross validation using the training set, which represents how the model performas on the programs which are similar to the training programs. Also, it can be validated using the testing set, which shows how the model predicts timing errors of programs which it has never seen during training.

3. RESULTS & ANALYSIS

In this section, we compare the models learned by both J48 Decision Tree and Multilayer Perceptron algorithms in terms of correct classification rate, time taken to build models, how different data representation affects the performance of the model, and the different validation results.

3.1 Instruction Window Size

We measure the correct classification rate across different instruction window size (1, 2 and 3). Instruciton window size of 2 gives the highest classification rate, as Figure 2 shows.

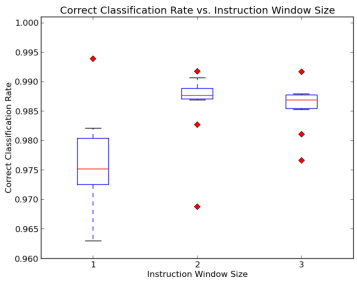

Figure 2: Classification Rate vs. Instruction Window Size

3.2 Time Consumption

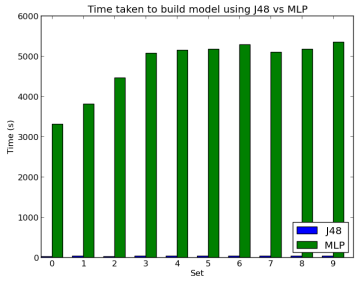

Figure 3: Time taken to build model using J48 and MLP

Figure 3 shows the time taken to build the model using J48 and MLP. As we can see, building model using MLP takes much longer time.

3.3 J48 vs. MLP

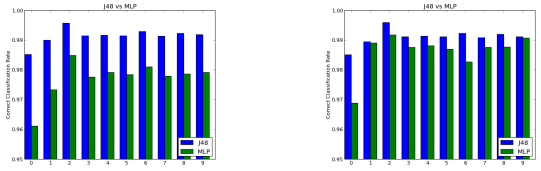

Figure 4: J48 vs. MLP with different input representation

When we use different data representation to build the model, the Decision Tree model is always better. However, you can see an improvement in correct classification rate of MLP model when using bit representation in Figure 4.

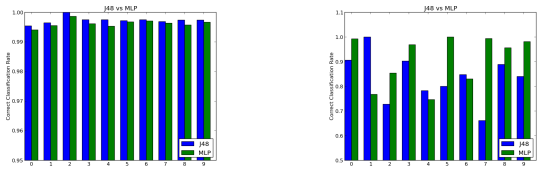

Figure 5: J48 and MLP with different validation

When we do 10-fold cross validation using training sets, both models show very high classification rates. However, once we validate the models using testing sets, the Decision Tree model has a significant decrease in correct classification rate, which the Multilayer Perceptron model still has relatively high rates, as shown in Figure 5. Therefore, even the Decision Tree model has better performance for programs which are similar to the training programs, the Multilayer Perceptron model is more robust to new programs.

3.4 Classification of Timing Errors

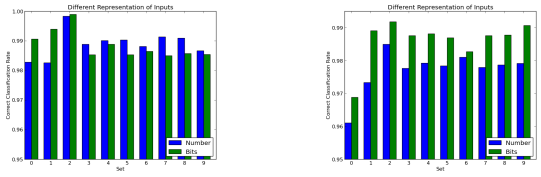

Figure 6: MLP with different error rates

We believe that the timing errors can be classified into two categories. One category is strongly linked to the operation , for example, addition, subtraction, load and store, while the other one is related to the data used in the operation.

Also, different data representation of inputs have a strong effect on the performance of Multilayer Perceptron model as proved above. The figures above show the different performance of Multilayer Perceptron model with different input representation under different error rates. The left one decribes the correct classification rate of Multilayer Perceptron model when the error rate is 10%, and the right one shows the classification rate when the error rate is 20%.

We can clearly see, when the error rate is 10%, the models with different input representations have similar performance. However, the error rate increases to 20%, the performance of Multilayer Perceptron model with bit representation outperforms, which indicates that the errors are caused by the data usage.

4. CONCLUSIONS

Both algorithms achieve good prediction results in the experiments.

Each algorithm has it own advantages and disadvantages.

- Building J48 Decision Tree model is much faster than building Multilayer Perceptron model, especially when the input dimension is high.
- J48 Decision Tree model performs better when predicting program which are similar to the training programs.
- Multilayer Perceptron model is more robust to the new programs.

The Decision Tree model is much easier to interpret. We can easily learn how the model is built from the inputs by studying the hierarchy of nodes in the output decision tree, while the Multilayer Perceptron model is like a blackbox.

Machine learning algorithms are also effective to learn interesting charateristics of timing errors.

5. REFERENCES

[1] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 7–18, 2003.

[2] D. Blaauw, S. Kalaiselvan, K. Lai, W.-H. Ma, S. Pant, C. Tokunaga, S. Das, and D. Bull. Razor ii: In situ error detection and correction for pvt and ser tolerance. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 400–622, 2008.

[3] J. Xin and R. Joseph. Exploiting locality to improve circuit-level timing speculation. *Computer Architecture Letters*, 8(2):40–43, 2009.

[4] J. Xin and R. Joseph. Identifying and predicting timing-critical instructions to boost timing speculation. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44 '11, pages 128–139, New York, NY, USA, 2011. ACM.

[5] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge. Opportunities and challenges for better than worst-case design. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 2–7, New York, NY, USA, 2005. ACM.