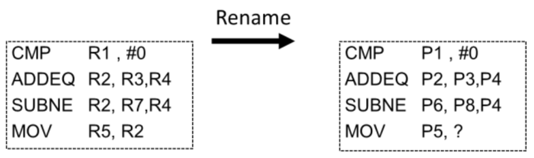
Using Condition Flag Prediction to Improve the Performance of Out-of-Order Processors

Tzu-Hsuan Hsu, Ching-Wen Lin and Chung-Ho Chen
Dept. of Electrical Engineering and Inst. of Computer & Communication Engineering
National Cheng Kung University, Tainan, Taiwan

Abstract—**If-conversion is a technique that reduces the misprediction penalties caused by conditional branches. However, executing If-converted code in out-of-order processors creates a naming problem which hinders the rename throughput. Predicting condition flag is an effective approach to resolve this problem. In this paper, we propose a scheme to predict the condition flag based on the ISA of ARM. By restoring two most recent unique condition flag values for each instruction dynamically in run time, and by using a condition flag selector when a condition flag-updating instruction reaches the renaming unit, we can predict the outcome of the condition flag-updating instruction. We show that such an approach is able to achieve the IPC performance increase of 6.62%.**

I. INTRODUCTION

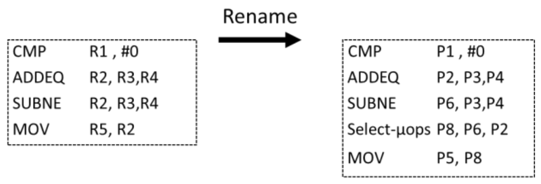
Branch prediction is used to remove the control dependency and expose the ILP. However, with a deeper pipelines, branch misprediction may result in severe performance degradation. For reduce the miss penalty, if-conversion [1] is used to eliminate conditional branches by exploiting conditional execution to transform the control dependence into data dependence. However, in Out-of-Order processors, the use of conditional execution has a register naming issue: there can be multiple register definitions on single destination register at rename time. If there are multiple updates of the same register of different conditions and if the condition flags have not been resolved, then it is unknown which physical register should be mapped on to the architectural register. Figure 1 illustrates the problem.



```
CMP     R1 , #0
ADDEQ   R2, R3,R4
SUBNE   R2, R7,R4
MOV     R5, R2
```

```
CMP     P1 , #0
ADDEQ   P2, P3,P4
SUBNE   P6, P8,P4
MOV     P5, ?
```

Figure 1. Multiple register definitions



```
CMP     R1 , #0
ADDEQ   R2, R3,R4
SUBNE   R2, R3,R4
MOV     R5, R2
```

```
CMP           P1 , #0
ADDEQ         P2, P3,P4
SUBNE         P6, P3,P4
Select-μops   P8, P6, P2
MOV           P5, P8
```

Figure 2. Insertion of select-μop

To deal with this problem, one approach could simply stall the renaming unit until the condition flag is resolved, however this would cause great performance degradation. Another simple approach is to change the semantic of conditional instruction to C-Style form such as: *register mapping = (condition) ? normal execution : previous register mapping*. This approach serializes the execution flow of conditional instruction. It may become the limitation for out-of-order processors to expose the ILP. Wang et.al [2] has proposed a Register Alias Table (RAT) to help finding the multiple register definitions at the rename time. If multiple definitions are found in RAT, a "select-μop" is generated and inserted into instruction stream dynamically. It is used to combine the different execution flow of conditional execution into a single one so that the new physical register for the select-μop becomes a unique mapping for the register of multiple definitions. Figure 2 shows the example of select-μop insertion. However, the use of RAT complicates the rename logic. In addition, this approach increases the register pressure, because each select-μop is allocated a new physical register. Furthermore, instructions with the false condition are not cancelled in pipeline at the rename time. So they are still consuming physical registers, issue window entries, and functional units. In IA64 ISA, predicting predicate is an effective approach [3-5] to solve multiple register definitions. The predicate predictor uses the PC of a compare instruction to predict the compare output. All predicates are known at the rename stage. So when the predicate of an instruction is predicted false, it can be removed from the pipeline before renaming. This approach not only resolves the multiple register definitions but also avoids the resource pressure caused by the removed instruction.

In this paper, we propose a condition flag prediction scheme based on the ISA of the popular ARM processors. The similarity between predicate prediction and condition flag prediction is that we predict the condition flag-updating instruction output. Once the conditional instruction reaches the rename stage, it always can be checked to determine if the condition is true or false.

The rest of the paper is organized as follows. Section II describes our proposed approach. Section III presents the experimental results. Finally, we conclude the paper in Section IV.

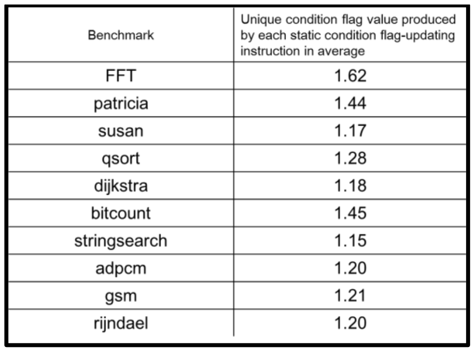
II. Condition Flag Prediction

For the consideration of hardware resource saving, predicting the predicates is a good solution for multiple register definitions based on IA64. In this section, we will describe how to predict the condition flag based on ARM ISA where each instruction can be conditional executed if intended.

A. *Flag value locality*

In ISA of ARM [9], the N, Z, C, and V bits are combined as the condition flag. The condition flag can be tested by conditional instructions to determine whether the instruction is to be executed or not. In our approach, the condition flag is considered as a 4-bit value. That is, the result of prediction is one of the 2^4 values. To capture the behavior of condition flag-updating instruction, we want to know that the amount of past history needs to be considered in making condition flag predictions. We recorded the condition flag value of each condition flag-updating instruction, and measured how many unique condition flag values are produced by each static condition flag-updating instruction in average. Table I shows the result. These values are for the MiBench [10] benchmark suite. From the table we can see that the number of unique flag values is lower than two in average. That is, for a large number of condition flag-updating instructions, the unique flag value updated by these instructions is two or fewer values. It would be worthwhile to exploit this behavior. If we store a maximum of two most recent unique flag value for each condition flag-updating instruction, and do a binary encoding of these two outcomes, we can use a condition flag selector to making a prediction.

TABLE I. Average Number of Unique Condition Flag



Benchmark	Unique condition flag value produced by each static condition flag-updating instruction in average
FFT	1.62
patricia	1.44
susan	1.17
qsort	1.28
dijkstra	1.18
bitcount	1.45
stringsearch	1.15
adpcm	1.20
gsm	1.21
rijndael	1.20

B. *Condition Flag Selector*

The control flow of a program is predicted by branch prediction before if-conversion is performed while the condition execution instruction is transformed from if-conversion and the value of a condition flag also determines the control flow. Therefore, we incorporate the branch prediction concept [11] into the condition flag selector.

C. *Condition Flag Predictor*

The condition flag predictor is mainly divided into two parts: Condition Flag Value Table (CFVT) and condition flag selector. Figure 3 shows the block diagram of a condition flag predictor. The CFVT has three fields : Tag, Flag Values, and Last used. The Flag Value field stores up to two most recent unique values of the condition code. These two values are associated with the binary encoding {0, 1}. When the condition flag-updating instructions keep producing one of these two values dynamically in run time, the flag value can be predicted by selecting one of the two outcomes from {0, 1}, and determine whether the following instruction is to be executed or not. When a third unique flag value is produced, it replaces the least recently used flag value from the flag values field. The Last Used field keeps recording which flag value was used.

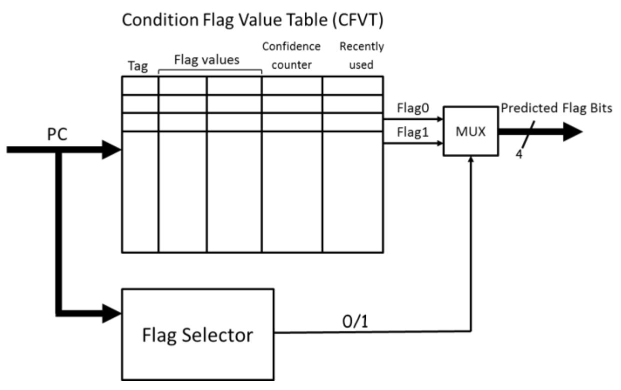


Figure 3. Block diagram of a condition flag predictor

For the condition flag selector, the mapping from branch prediction is as follows. Given the PC of a condition flag-updating instruction, the prediction is produced. The taken and not taken predicted by the branch predictor is mapped to the binary encoding {0, 1} of Flag Value field from CFVT.

The condition flag predictor works as follows. When making the prediction for an instruction, the corresponding CFVT entry is accessed, and its Tag field is tested to determine if the CFVT entry is a hit. If so, the condition flag selector generates either true or false result, then the flag value corresponding to that binary encoding is selected as the next prediction. If a miss occurs in the CFVT, then no prediction is made. In this case, when a following conditional instruction reaches the rename stage, it is stalled until the condition flag is resolved.

To reduce the penalty of miss prediction, we integrate a confidence threshold with the condition flag predictor to select which prediction is worthy to be used. Each entry of the CFVT is extended with a Confidence Counter field. It is a saturated counter that increases when a correct prediction occurs and zeroes when a misprediction occurs. A prediction is made if the counter value equals the confidence threshold.

III. Performance Evaluation

This section evaluates the performance of our condition flag prediction approach with various architectural design options.

A. *Experimental Setup*

All the experiments presented in this paper used a cycle-accurate SystemC simulation model [12] that runs ARMv6 ISA binaries. The main architectural parameters are shown in

Table II. The baseline architecture is identical to this, except that when the condition flag-updating instruction has not yet committed and the following conditional instruction has reached the rename unit, the execution is stalled until the condition flag is resolved. To implement the condition flag predictor we incorporate the gshare/local combined predictor into condition flag selector.

We have simulated ten benchmark programs from MiBench. All benchmarks have been compiled with GNU ARM cross-compiler.

TABLE II. PROCESSOR ARCHCHITECTURE PARAMETER VALUE USED

Fetch/Decode/Rename Width	2
Issue/Commit Width	4
Issue Window Size	32
Physical Registers	80
Function Units	2 ALU, 1 MUL, 1 L/S, 1 Branch, 1 Coprocessor unit
Branch Predictor	Multilevel : 1.Gshare 5-bit GHR, 1k entries count 2.Local 1k entries LHT, 1k entries count
D-Cache/I-Cache	64KB, 4-way, 3 Cycles delay
Condition Flag Predictor	2k entries condition flag table(size:8.75KB) Gshare/Local Conbined predictor : (size:4.25KB) 1.Gshare 5-bit GHR, 2k entries count 2.Local 2k entries LHT, 2k entries count

B. *Confidence Threshold*

Figure 4 shows the performance of condition flag predictor as the IPC speedups with threshold 0, 2, and 4 respectively. All speedup results are normalized to the baseline processor. Note that FFT with threshold 0, the speedup turns into a slowdown of -7.22%. Figure 5 shows the miss prediction rate of condition flag prediction. We can see that the miss prediction rate of FFT is up to 37%. The high miss prediction rate causes this performance penalty. For the FFT with threshold 2, the miss prediction rate is cut downs to 17% and the performance improves more than 9%. However, for the rest of benchmark programs, the decrease of miss prediction rate does not result in performance improvement. Figure 6 shows the distribution of no prediction, incorrect prediction and correct prediction. We can see that the increase of confidence threshold could result in losing opportunity for correct predictions and decrease of confidence threshold could result in great performance penalty because of miss prediction. Moreover, with higher threshold, the increase of no prediction has an impact on performance. This is because the conditional instructions are stalled at the rename unit until the corresponding condition flag is resolved. The wasted cycles decreases the register rename throughput. Figure 7 shows how the threshold increases that increase the number of stalled cycles caused by waiting condition flag to be resolved. Therefore, the confidence threshold does not affect equally in each benchmark program. Comparing these results of performance evaluation with difference confidence threshold, the condition flag predictor is able to achieve the IPC performance increase of 6.62% with threshold 2.

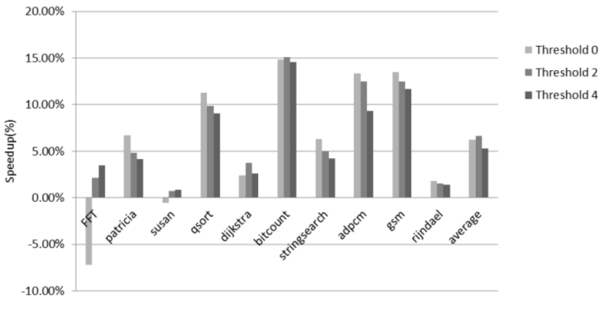


Figure 4. Performance impact of difference confidence threshold values

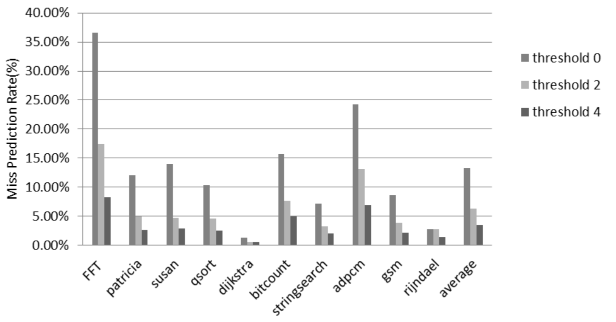


Figure 5. Miss prediction rate of condition flag predictor with difference confidence threshold values

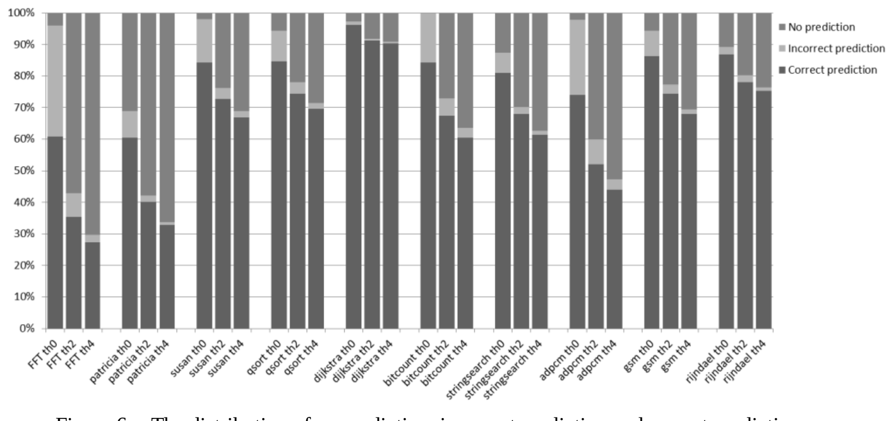


Figure 6. The distribution of no prediction, incorrect prediction and correct prediction

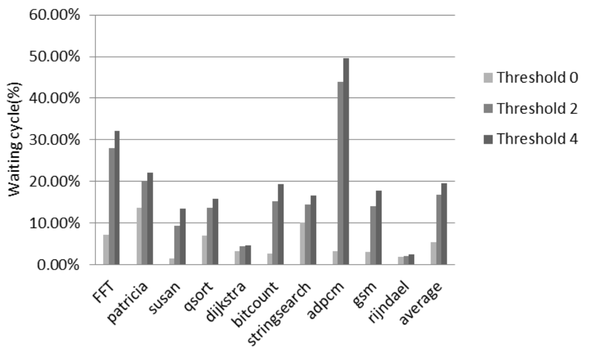


Figure 7. Percentage of stalled cycles caused by waiting condition flag is resolved

C. The Condition Flag Predictor For Conditional Branch

The condition flag determines the direction of conditional branch and flag predictor predicts the flag value. It is interesting to know whether branch predictor is actually needed since the condition flag predictor determines the direction of conditional branch indirectly. To examine this, we remove the branch predictor from the processor. Figure 8 shows the result of performance. Comparing with Figure 4, we can see that when the confidence threshold is 0 the average performance improvement is only slightly different. But for a processor without a branch predictor, when the threshold value is increased, the performance keeps decreasing constantly. This is because the waiting cycles caused by the conditional branch stall the instruction execution until the condition flag is resolved. This situation does not happen in the processor with a branch predictor. The branch predictor always determines the direction of a conditional branch even when the condition flag predictor makes no prediction or the condition flag is not resolved yet. For area consideration, we can remove the branch predictor and predict the condition flag with a lower threshold value. In this way, about only one percent of IPC is lost compared with the configuration that both predictors are used.

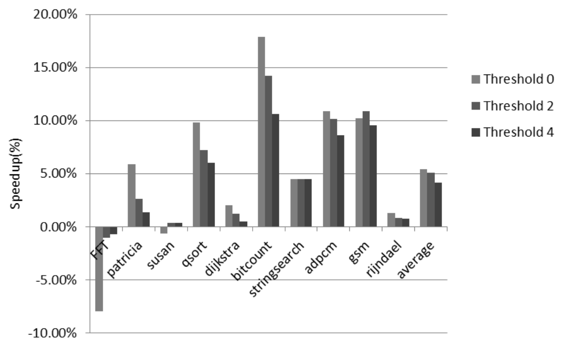


Figure 8. Performance impact of processor remove the branch predictor

IV. CONCLUSIONS

The execution of conditional instruction in out-of-order processors creates a multiple register definitions problem. Predicting condition flag is an effective approach to address this problem. In this paper, we propose a new scheme to predict the condition flag based on the ISA of ARM processors. To design the condition flag predictor, we use the concept of branch prediction to implement our flag selector and we combine a threshold value to the basic condition flag prediction mechanism. We have simulated benchmark programs in several threshold values. We find that the benchmark programs have different performance sensitivity to threshold adjustment. On average, our approach is able to achieve an IPC performance increase of 6.62%.

REFERENCES

[1] J. R. Allen, K. Kennedy, C. Porterfield, and Joe Warren, "Conversion of Control Dependence to Data Dependence," Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages Page 177 – 189, 1983.

[2] P. H. Wang, H. Wang, R. M. Kling, K. Ramakrishnan, and J. P. Shen, "Register Rename and Scheduling for Dynamic Execution of Predicated Code," High-Performance Computer Architecture(HPCA 7th), pp.15-25, Jan. 2001.

[3] W. Chuang and B. Calder, "Predicate Prediction for Efficient Out-of-Order Execution," Proceedings of the 17th annual international conference on Supercomputing, 2003.

[4] E. Quinones, J-M. Parcerisa, and A. Gonzalez, "Selective Predicate Prediction for Out-of-Order Processors," Proceedings of the 20th annual international conference on Supercomputing, 2006.

[5] E. Quinones, J-M. Parcerisa, and A. Gonzalez, "Improving Branch Prediction and Prediction and Predicated Execution in Out-of-Order Processors," Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture, 2007.

[6] K. W and M. Franklin, "Highly Accurate Data Value Prediction using Hybrid Predictors," in MICRO 30: Proceedings of the 30th annual IEEE/ACM International Symposium on Microarchitecture, pages 281-290, Washington, DC, USA, 1997.

[7] B. Calder, G. Reinman, and D. M. Tullsen, "Selective Value Prediction," in Proc. 26th Int. Symp. Comput. Arch, pp. 64-74, May 1999.

[8] Y. Sazeides and J. E. Smith, "The Predicability of Data Values," in MICRO 30: Proceedings of the 30th annual IEEE/ACM International Symposium on Microarchitecture, pages 248-258, Washington, DC, USA, 1997.

[9] ARM Corporation, "ARM Architecture Reference Manual."

[10] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," in Proc. IEEE 4th Annu. Workshop Workload Characterization, pp. 3-14, Dec. 2001.

[11] S. McFarling, "Combining Branch Predictors," Tech. Rep. Digital WRL, Jun. 1993.

[12] J.-W. Lin, "Design, Analysis, and Implementation of a Parameter-Based Out-of-Order Superscalar Microprocessor Conforming to ESL Methodology," Master Thesis, Dept. of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, Jul. 2008.