IEICE TRANS. INF. & SYST., VOL.E88–D, NO.5 MAY 2005

LETTER

Speculative Branch Folding for Pipelined Processors

Sang-Hyun PARK[†*a)], **Sungwook YU**[††], ***and*** **Jung-Wan CHO**[†††], ***Nonmembers***

SUMMARY This paper proposes an effective branch folding technique which combines branch instructions with predicted instructions. This technique can be implemented using an instruction queue, which buffers prefetched instructions. Most of the instructions in the instruction queue are forwarded to the execution unit in sequence. Branch instructions, however, are combined with predicted instructions in the instruction queue and these folded instructions are forwarded to the execution unit. Miss-prediction can be recovered by flushing folded instructions without processor state recovery and by restarting from the other path. Simulation and implementation results show that both performance and power consumption are significantly improved with little additional hardware cost.

key words: *branch folding, speculative, embedded processor, pipeline*

1. Introduction

Branching is one of the major sources of performance degradation in pipelined processors. It increases program execution cycles and thus makes the processors consume more power. Various techniques have been proposed to reduce branch penalty of pipelined processors [1]–[3]. These techniques have mainly focused on how to reduce performance degradation, without considering hardware cost or power consumption. For embedded applications, we need to consider hardware cost and power consumption as well as performance.

Branch folding is a technique to combine a branch instruction with a normal instruction, and forward the folded instruction instead of the branch instruction to execution pipeline. Because branch does not enter execution pipeline, effective execution time of the branch instruction is 0. While other techniques only consider branch penalties, the branch folding technique eliminates branch execution cycles as well as branch penalties.

In this paper, we propose an effective branch folding technique, speculative branch folding. While the original branch folding [2] combines branch instructions with the preceding instructions, the proposed technique combines branches with the predicted instructions of the branches. Speculative branch folding can be applied to all kinds of branches unlike another branch folding technique for embedded processors, which can be applied only to short backward branches [5]. The proposed branch folding technique is applied to an embedded processor with little hardware overhead and achieves both performance improvement and lower power consumption.

Manuscript received September 27, 2004.
Manuscript revised December 10, 2004.

[†]The author is with Department of Computer Science, Korea Advanced Institute of Science and Technology, Korea.
[††]The author is with the Faculty of Electrical and Electronics Engineering, Chung-Ang University, Korea.
[†††]The author is with the Faculty of Computer Science, Korea Advanced Institute of Science and Technology, Korea.
[*]Presently, with Samsung Electronics Co., Korea.
a) E-mail: sh94.park@samsung.com

DOI: 10.1093/ietisy/e88–d.5.1064

2. Related Works

Branch folding is originally proposed in CRISP microprocessor [2]. In this processor, a decoded instruction cache is inserted between fetch-and-decode unit and execution unit. Each cache entry contains next-PC, alternate next-PC, PC tag, and decoded instruction. When a branch instruction follows a normal instruction, next-PC field is filled with the predicted address of the branch and alternate next-PC field is filled with the address of the other path. Although this technique shows good performance when executing branch instructions, hardware cost becomes too large when applying to embedded processors. Decoded instruction cache contains 32 192-bit entries, total of 6144 bits, to implement CRISP microprocessor, which is about six-times larger than the total size of general purpose registers in an embedded processor core, CalmRISC^{TM}-32 [4].

In [5], a low cost branch folding technique is proposed for small tight loops. When a short backward branch (SBB) is decoded, the addresses of the branch and target instructions are stored in SBB address register and target address register, respectively, and the processor enters branch folding ACTIVE state. In ACTIVE state, the target address rather than SBB address is sent to memory if the address of the next instruction is the same as SBB address. This eliminates SBB instructions from the loop execution except for the first and the last iterations. Though adequate in embedded processors, this method can be applied only to small tight loops. Moreover, this method cannot be applied to outer branches of nested loop that is often used in embedded applications.

3. Speculative Branch Folding

Branch prediction makes it possible to fetch instructions from the predicted path before the branch is resolved. Speculative branch folding is a technique to combine a branch instruction with the first instruction of the predicted path. Instead of the branch instruction, only the speculatively folded instruction enters execution pipeline, which completely re-

moves branch execution cycles if the prediction is correct.

To implement speculative branch folding, we use a queue structure to store pre-fetched instructions and to combine the branches with the predicted instructions. As in [2], the pipeline structure of the proposed approach is divided into two parts, and then linked by this queue structure, as shown in Fig. 1. The pre-fetch unit fetches and partially decodes instructions to find branches. If a branch instruction is fetched, the pre-fetch unit stores the branch instruction in the instruction queue with some partially decoded information, branch flag and branch condition. Then, the pre-fetch unit fetches instructions from the predicted path.

The pre-fetch unit may operate in a pipelined manner to reduce cycle time. In this case, some fall-through instructions can be fetched from memory during prediction and they can be stored in the instruction queue regardless of predicted direction. Because, if predicted taken, the branch and predicted instructions cannot be located contiguously in the instruction queue, the index of the predicted instruction must be stored. When the branch instruction reaches the front of the instruction queue, the predicted instruction is restored using stored index and then combined with the branch condition. Instead of the branch instruction, the predicted instruction with the branch condition enters execution pipeline, which makes the same effect as the parallel execution of the branch instruction and the predicted instruction.

Figure 2 shows the sequence of this scenario. The pre-fetch unit is assumed to have 2-stage pipeline structure and fetch 2 instructions in one cycle. The instructions in gray boxes are the ones to be forwarded to the execution unit in each cycle. In cycle 2, the direction of the branch is predicted taken. Thus, the pre-fetch unit starts fetching target instructions, instr_6 and instr_7. Because instr_4 and instr_5 are fetched in cycle 2, the instruction queue contains these instructions in cycle 3. In cycle 4, the predicted instruction of the branch, instr_6, is folded with the branch instruction and this folded instruction will be forwarded to execution pipeline instead of the branch instruction.

Miss-prediction recovery is one of the essential problems in speculative branch folding. In the conventional single-issue, 5-stage pipeline structure (fetch, decode, execute, memory, write-back), a branch can be resolved in stage decode because the branch condition can be known no later than stage execute of the preceding instruction. When applied to this pipeline structure, miss-prediction recovery of speculative branch folding can be easily solved by flushing the folded instruction in stage decode because it does not change any state of the processor. Then, execution is restarted to follow the other path.

Miss-prediction penalty varies depending on branch direction if the pre-fetch unit is pipelined. As shown in Fig. 2, the miss-prediction penalty of the branch resolved as not-taken is only one cycle because some fall-through instructions already exist in the instruction queue. However, if the branch is predicted not-taken but resolved as taken, the processor must restart from the pre-fetch of the target instruction. In this case, the miss-prediction penalty will be one cycle bigger than the number of pipeline stages of the pre-fetch unit. Thus, always-taken prediction scheme is usually preferable. From our simulation results with 2-stage pipelined pre-fetch unit, always taken scheme shows better performance than more accurate other prediction schemes, as shown in Fig. 3. It should be noted that the CPI is less than 1 even though the processor is a single-issue machine. The reason is that the pre-fetch unit fetches 2 instructions in one cycle and the instruction queue eliminates branch instructions before entering the execution unit when the branch folding is successfully applied.

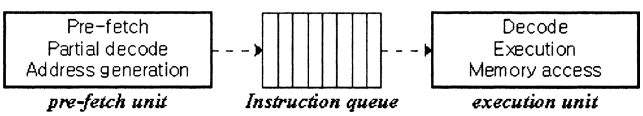

Fig. 1 Separated pipeline structure.

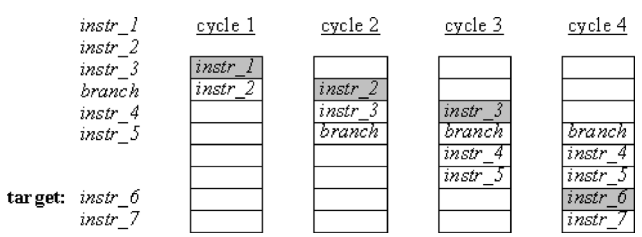

Fig. 2 Snapshot of instruction queue.

4. Experimental Result

Figure 4 compares speculative branch folding with base model, SBB-folding, and squashing. Base model is the conventional single-issue pipelined processor that has 2-cycle branch penalty, SBB-folding represents the method proposed in [5], and squashing is the delayed branch scheme with controlled squashing of the instruction in the delay

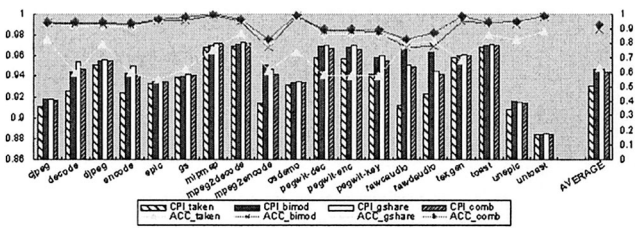

Fig. 3 CPI of branch folding vs. prediction accuracy.

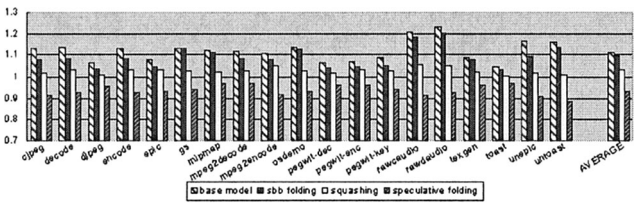

Fig. 4 CPI comparison.

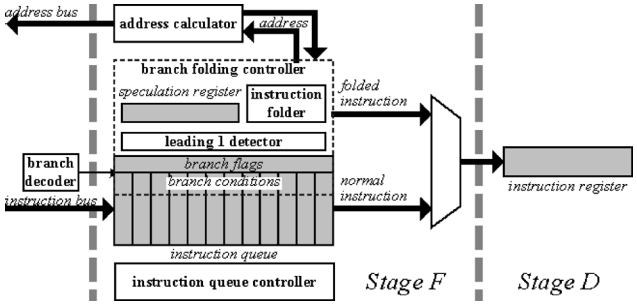

Fig. 5 Data path of fetch unit implementing branch folding.

Table 1 Implementation result.

	gate count	execution cycle	power (mW/MHz)	total energy (μJ)
original	332,605	7961	1.08868	8.666981
branch folding	334,742	6400	1.14474	7.326336

slot [12]. The proposed speculative branch folding technique assumes that the miss-prediction penalty of taken branch is 3 and not-taken branch is 1. Simplescalar [6] simulator and Mediabench test suit [7] are used in the evaluation. As can be seen in Fig. 4, the proposed speculative branch folding technique shows better performance than SBB-folding and delayed branch with squashing.

The proposed speculative branch folding technique has been implemented in an embedded processor, CalmDSP™ [8], by modifying the RTL code of the fetch unit of the integer core. Figure 5 shows the data path of the fetch unit that implements speculative branch folding technique in CalmDSP™. The dashed rectangular area represents the additional hardware. Each instruction queue entry has 4 additional bits to contain branch flag and branch condition. Speculation register contains 4-bit index of the target instruction in the instruction queue. In this implementation, the instruction queue contains only 320 bits, which leads to much lower cost than the decoded instruction cache in CRISP microprocessor [2]. Table 1 compares the original implementation results with the new implementation results that uses the branch folding technique. SYNOPSYS synthesis tool [9] has been used to obtain gate count and Verilog-XL [10] simulator has been used to count execution cycles. Power and total energy have been calculated by Samsung's in-house tool, Cubic-power [11]. As can be seen in Table 1, the proposed branch folding technique leads to both better performance and lower power consumption.

5. Conclusion

This paper proposes an effective branch folding technique. It has been shown to improve performance 15 percent compared to the technique proposed in [5]. The proposed technique has been applied to CalmDSP™and requires much less hardware cost than [2]. Total energy consumption has been reduced significantly because much fewer execution cycles are spent than an implementation without the proposed branch folding technique.

References

[1] J.K.F. Lee and A.J. Smith, "Branch prediction strategies and branch target buffer design," Computer, pp.6–22, Jan. 1984.
[2] D.R. Ditzel and H.R. McLellan, "Branch folding in the CRISP microprocessor: Reducing branch delay to zero," Proc. 14th Ann. Symp. Computer Architecture, pp.6–22, Jan. 1984.
[3] T.-Y. Yeh and Y. Pet, "Two-level adaptive training branch prediction," Proc. 24th Annual Symposium and Workshop Microarchitecture, IEEE CS Press, pp.51–61, Los Alamitos, Calif., 1991.
[4] S. Cho, S. Park, S. Kim, Y. Kim, S.-W. Jeong, B.-Y. Chung, H.-R. Roh, J.-H. Lee, H.-M. Yang, S.-H. Kwak, and M.-K. Lee, "CalmRISC™-32: A 32-bit low power MCU core," The 2nd IEEE Asia Pacific Conference on ASICs (AP-ASIC), pp.285–289, Cheju, Korea, Aug. 2000.
[5] L.H. Lee, J. Scott, B. Moyer, and J. Arends, "Low-cost branch folding for embedded applications with small tight loops," Proc. 32nd Int'l Symp. Microarchitecture, pp.103–111, 1999.
[6] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," Computer, pp.59–67, Feb. 2002.
[7] C. Lee, M. Potkonjak, and W.H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," Proc. 30th Int'l Symp. Microarchitecture, pp.330–335, 1997.
[8] Consortium of Semiconductor Advanced Research, System-IC 2010 Foundation Technology Project of High Performance Embedded MCU/CPU: final report, 2003.
[9] Design Compiler™ Reference Manual: Optimization and Timing Analysis, Synopsys, Nov. 2000.
[10] Verilog-XL Reference: Product Version 3.2, Cadence Design Systems, Jan. 2001.
[11] CubicWare User's Manual, Samsung Electronics, Oct. 2001.
[12] S. McFarling and J. Hennessy, "Reducing the cost of branches," Proc. 13th Ann. Symp. Computer Architecture, pp.396–403, 1986.