Dynamic Exploitation of Redundancy in Programs Using Value Prediction and Instruction Reuse

Gokul Nath Babu Manoharan and Jayaprakash S. Narayanan
Department of Computer Science and Engineering, College Of Engineering Guindy,
Anna University, Chennai, India
{gokulnathbabu, jayan82}@yahoo.com

Abstract

Value Prediction (VP) and Instruction Reuse (IR) are the two techniques used to capture the redundancy in programs. This paper proposes a method to combine both the IR and VP in order to reuse or predict the result of an instruction. Confidence values are associated with the predicted values to minimize the mispredictions. When an instruction mispredicts, only the consumers of the mispredicted instruction are re-executed rather than flushing the instructions following the mispredicted instruction. A new stage called confirm stage is added after the writeback stage of a 6-deep pipeline to enable selective reexecution. Only those instructions executing with the predicted operands go through the confirm stage so that the instructions not using the predicted operands are unaffected by this additional stage. Further branch misprediction penalty is not increased because of this new stage since branches do not use predicted operands. Finally, simulation results show an average increase in speedup of 10% on a 4-wide superscalar processor.

1. Introduction and Related Work

Modern processors remove most artificial constraints on execution throughput. Out-of-order processors remove artificial dependencies imposed by instruction ordering; register renaming removes false dependencies; and aggressive branch prediction schemes greatly reduce serialization of instruction execution due to branches. Therefore, the bottleneck for many workloads on current processors is the true dependencies in the code. They force the instructions to be serialized reducing the degree of ILP. Data Speculation is a technique that collapses this dependency chains by predicting the result of the instruction so that the dependent instructions are speculatively executed.

The amount of redundancy present in the program i.e. many instructions perform the same computation and produce the same result again and again, forms the basis of data speculation. Many studies [1, 2, 3, 4] show that more than 75% of the dynamic instructions produce the same result as before. VP [2, 5] and IR [6] are the two techniques that have been proposed to exploit this redundancy. Both the techniques try to improve the performance by removing the data dependencies.

Both VP and IR use different approaches in exploiting the redundancy in the program. VP predicts the results of instructions (or, alternatively, the inputs to other instructions) based on the previously seen results, performs computation using the predicted values, and confirms the speculation at a later point. The critical path is shortened since the instructions that would normally be executed sequentially could be executed (speculatively) in parallel. On the other hand, IR recognizes that a certain computation chain has been performed before and therefore need not be performed again, i.e., it "splices out" a chain of computation from the critical path. The study in [7] shows the difference between IR and VP and how they interact with the processor pipeline. It also shows that 84% to 97% redundancy in the program can be captured.

In this paper both IR and VP are used to capture the maximum amount of redundancy possible. The study in [8] shows a mixed predictor using both IR and VP. Using a single Mixed Buffer for both IR and VP in [8] leads to conflict misses. Confidence values are not associated with the values predicted in [8]. The details regarding the selective reexecution of instructions that have consumed mispredicted operands are not clearly specified in [8]. In this paper we have solved the above said problems and a new stage called confirm stage is added that aids in reducing the misprediction penalty by selective re-execution of mispredicted instructions and the instructions truly dependent on it.

1.1 Overview

This paper is organized as follows: section 2 describes the value prediction architecture used, section 3 describes the instruction reuse architecture used, section 4 describes the processor pipeline using both VP and IR, section 5 shows the simulation results and section 6 summarizes and concludes the work.

2. Value Prediction Architecture

This section describes the Value Prediction Architecture used in this paper.

2.1 Value Prediction Table

Several Architectures have been proposed for value prediction including Last Value Prediction, Stride Prediction, Context-Based Prediction and Hybrid approaches. In this paper we use a hybrid of stride and context-based predictor as proposed in [9]. Confidence values are associated with the predicted values as proposed in [10].

2.1.1 Stride

A stride predictor keeps track of not only the last value brought in by an instruction, but also the difference between that value and the previous value.

This difference is called the stride. The predictor speculates that the new value seen by the instruction will be the sum of the last value seen and the stride. In this paper two-delta stride predictor is used which replaces the predicted stride with the new stride only if the new stride has been seen twice in a row. A warmup counter is associated with each entry so that initial mispredictions are avoided and the stride predictor starts predicting only after the warmup counter reaches a threshold.

2.1.2 Context

A context predictor bases its prediction on the last several values seen. In this paper the last 4 values produced by an instruction are used. A table called the Value History Table (VHT) contains the last 4 values produced for instructions in the table. Another cache, called the Value Prediction Table (VPT), contains the actual values to be predicted. An instruction's PC is used to index into the VHT, which holds the past history of the instruction. The 4 history values in this entry are combined and folded using an xor hash into a single index into the VPT [11]. This entry in the VPT contains the value to be predicted. Confidence counters are associated with each entry in VPT to guide prediction. Warmup counter is also used to avoid initial mispredictions.

2.1.3 Hybrid

In this paper hybrid predictor as proposed in [9] is used. It consists of a stride predictor and a context predictor, which are as described above. The stride information is added in the VHT entry. Confidence counters guide the prediction. There are separate confidence counters for stride and context predictors. On correct prediction the confidence counter is incremented by the increment bonus and on misprediction it is decremented by the misprediction penalty. If both the predictors hit then the prediction that has higher confidence is used. If the confidences are equal then the preference is given to stride predictor as in [10]. Further the values are predicted only if the confidence is above the prediction threshold. Replacement counter is associated with each entry in the VHT. This counter aids in preventing replacement of highly predictable instruction due to conflict miss. The number of entries in the VHT is 4K and that in the VPT is 8K. The misprediction penalty is 4, increment bonus is 2 and the prediction threshold is 6. The saturating limit in the counters is 15.

3. Instruction Reuse Architecture

This section describes the instruction reuse architecture used in this paper.

3.1 Reuse Buffer

In [6] various schemes for dynamic instruction reuse has been specified that include S_v, S_n, $S_{n+d.}$ In this paper we use S_v that tracks the operand values of the instruction. In the reuse buffer operand values and the corresponding result are stored. When an instruction is decoded, its current operand values are compared with those stored in the RB. If they are the same, then the result stored in the RB is reused. This comparison of available operands with the operands stored in the reuse buffer is the reuse test. The reuse buffer is indexed using the instruction's PC. The reuse buffer is 4-way associative so that 4 different results of the instruction can be stored.

Replacement counter is associated with each entry, which is incremented by increment bonus on hit i.e. the referred instruction can be reused and is decremented by misprediction penalty on miss i.e. the entry for the instruction is present, but can't be reused because the current operands are not present. When a new instruction conflicts with the instruction already present in the buffer, it is not immediately replaced. But the replacement counter of the instruction present in the buffer is decremented. If the value in the replacement buffer is less than the replacement threshold then the new instruction replaces the old instruction. Thus the instruction that can be highly reused is not replaced. The reuse buffer used in this paper contains 1K entries.

Next section describes how the reuse buffer and the hybrid predictor interact in the pipeline.

4. Processor Pipeline using both Value Prediction and Instruction Reuse

The processor pipeline using both hybrid predictor and instruction reuse looks as shown in Figure 1.

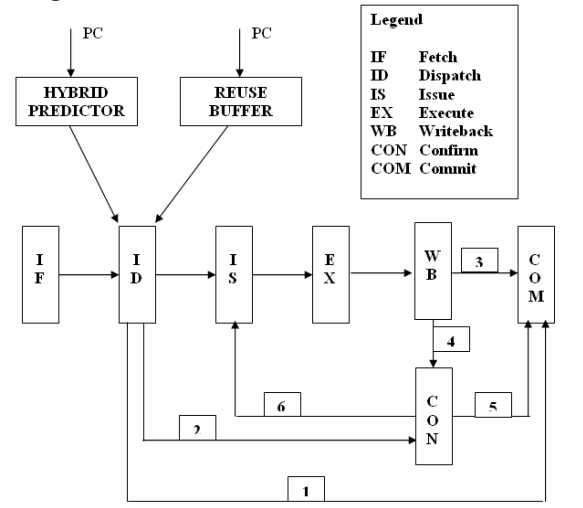

Figure 1

4.1 Unmodified Pipeline

The pipeline assumed in this paper is described in this subsection. In the fetch stage the instructions are fetched and put into the fetch queue. In the dispatch stage the instruction is decoded and an entry is allocated for that instruction in the reorder

buffer. The entry in the reorder buffer functions as the Reservation Station (RS). RS takes care of fetching the operands and issuing the instruction to the corresponding functional unit and committing the instructions in order. The RS has the status information denoting at which stage of the pipeline the instruction is currently in.

The following subsections describe how this pipeline is modified to enable the use of both VP and IR.

4.2 Modified Pipeline

4.2.1 Fetch Stage

In the fetch stage of the processor, the value prediction table and the reuse buffer is accessed for the range of PCs being fetched. The table lookups could potentially take multiple cycles, and needs to complete by the time the instruction enters the dispatch stage. Efficient techniques, like those proposed in [12], are needed to handle multiple value predictions, but modeling this is beyond the scope of this paper.

4.2.2 Dispatch Stage

In the dispatch stage, an entry in the reorder buffer is allocated for in order commit of the instructions. The instruction is decoded, and register renaming is done, where the instruction is allocated a physical register. The allocated physical register is normally used to hold the result value for an instruction but it is also used to temporarily hold the predicted value for an instruction. To the register status information, which normally contains the information of the producer to that register, an additional bit is also associated which is set when the register holds the predicted result. Confidence value for the predicted result is also added to the register status information. A prediction flag is also added to the RS to indicate whether the instruction has predicted or not. Three bits are added to RS for each operand being used. One bit is set when the instruction is issued with the predicted operand; the second is set when that predicted operand is mispredicted; the third is set when the actual operand becomes ready. These bits are known as is_operand_predicted bit, is_operand_mispredicted bit and is_operand_ready bit respectively.

Branch instructions are not executed with predicted operands to avoid spurious branch mispredictions. In the reuse buffer the target of the branch is stored. When a branch instruction access results in a hit in the reuse buffer and passes the reuse test, branch mispredictions are detected in the dispatch stage itself. The effective address calculation for loads and stores do not use predicted operands, while the result of load is predicted.

4.2.2.1 Instruction Flow on Reuse Buffer Hit

If there is a hit in the reuse buffer then the reuse test is performed, provided the operands are available as predicted operands or actual operands. If the test is successful and the operands are actual operands then the instruction directly moves to the commit stage writing the result to the register as labeled (1) in Figure 1.

If the reuse test is successful with predicted operands, the hit confidence is calculated. Hit confidence is the minimum of the confidence values of the predicted operands. If VP returns a predicted value then the predicted value's confidence is compared with the hit confidence. If the hit confidence is greater then the prediction flag in the destination register is set, the result from the reuse buffer is written in the register, the prediction confidence in the register is made equal to the hit confidence and the instruction moves to the confirm stage setting the prediction flag in the RS as shown by (2) in Figure 1. If the hit confidence is less then the instruction decides to use the predicted values from the VP, rather than the value from the reuse buffer obtained using the predicted operands. So the predicted result from the VP is written into the register along with the prediction confidence, the predicted flag in the register as well as in the RS is set. The instruction then moves to the issue stage and waits for the actual operands. This enables in avoiding the use of predicted values with low confidence.

4.2.2.2 Instruction Flow on Reuse Buffer Miss

If there is a miss in the reuse buffer then actions are similar to the earlier case in which hit confidence is less than the prediction confidence. If no prediction is available from the VP then the instruction waits in the issue stage for the operands to be available, either in predicted or in actual form.

4.2.3 Issue Stage

In the issue stage, the instruction waits for the operands to be available. If the predicted flag in the RS is set then the instruction issues only after the actual operands are available. If the prediction flag is not set then the instruction can issue with the predicted operands. This makes sure that the instruction whose result is predicted executes only with the actual operands and the instruction that has not predicted its result can issue with the predicted operands. If the instruction that has predicted the result is allowed to use the predicted operands then multiple reexecution of the instructions using the predicted result of this instruction is possible. This also makes sure that an instruction is executed at most twice in case of misprediction.

4.2.4 Write Back Stage

After execution, the instruction enters the writeback stage. If the predicted flag in the RS is set, the result obtained is the correct result of the instruction, as it would have issued only with the actual operands. This correct result is compared with the predicted result. If both are equal then the register status for the destination is cleared and the result is

published in the Common Data Bus (CDB). The dependent instructions waiting on the result of this instruction can be issued. If the dependent instruction has been issued with the predicted result then the corresponding operand's is_operand_mispredicted bit is not set and the is_operand_ready bit is set.

If there is a misprediction then the correct value is published in the CDB. The register status of the destination is also cleared. The instruction waiting on the result of this instruction can be issued. If the dependent instruction has been already issued with the predicted operand, then the corresponding operand's is_operand_mispredicted bit is set and the is_operand_ready bit is also set.

If both is_operands_predicted and prediction flag of the RS are not set then the instruction has used the actual operands, hence the register status is cleared. The instruction moves to the commit stage as shown by (3) in Figure 1 for all the above cases.

Else the instruction has used the predicted operands then the predicted flag in the RS is set; the predicted value is written into the register along with the prediction confidence, which is now the minimum of the confidence values of the operands and the prediction flag in the register is also set. The instruction then moves to the confirm stage as shown by (4) in Figure 1. The instructions dependent on this instruction can use the predicted result and can be issued.

The reuse buffer and the value predictor are also updated in this stage.

4.2.5 Confirm Stage

When instruction is in the confirm stage, RS checks whether all its operands are ready. If all the operands are ready and no operand has been mispredicted, which can be found from the is_operand_mispredicted bit, the instruction moves to the commit stage as shown by (5) in Figure 1. The register status information is cleared and the instructions dependent on this instruction can be issued. If the dependent instruction has been issued with the predicted operand then the corresponding operand's is_operand_mispredicted bit is not set and the is_operand_ready bit is set.

If all the operands are ready and the operands are mispredicted then the instruction's is_operand_predicted flags are cleared and the operand moves back to the issue stage as shown by (6) in Figure 1. The misprediction recovery for the dependent instructions will take place in the writeback stage, as the predicted flag for this instruction is set and the instruction is executing with the actual operands.

5 Simulations

The simulator used in this study was derived from the SimpleScalar/Alpha 3.0 tool set [13], a suite of functional and timing simulation tools for the Alpha AXP ISA. The timing simulator executes only user-level instructions, performing a detailed timing simulation of an aggressive 4-way dynamically scheduled microprocessor with two levels of instruction and data cache memory. Simulation is execution-driven, including execution down any speculative path until the detection of a fault, TLB miss, branch misprediction, or load misspeculation. The baseline architecture used is shown in table 1.

Instruction fetch	4 instructions per cycle, one taken branch per cycle.
Instruction cache	16K bytes, 2-way set associative, 6 cycles miss latency.
Data cache	16K bytes, 2-way set associative, 6 cycles miss latency.
Branch predictor	Bimodal predictor with bimod table containing 2048 entries
Speculative execution	Decode, dispatch, issue: 4 instructions per cycle, ROB size: 32, LSQ size: 16, Optimistic memory disambiguation with forward.
Functional units	4 integer ALUs, 1 integer mult/div, 1/1 load/store, 4 fp ALUs, 1 fp mult/div.
Functional unit latency (total/issue)	Integer ALU-l/l, load/store l/l, integer MULT 3/l, integer DIV 20/19, FP adder 2/l, FP MULTI 4/1, FPDIV 12/12, FP SQRT 24/24.

Table 1. Baseline Architecture

The benchmark programs analyzed are listed in Table 2 along with their inputs and number of dynamic instructions executed on the timing simulator. Four of the integer programs from SPEC 2000 benchmark suite (gcc, vortex, vpr, twolf) are chosen. The -fastfwd option in SimpleScalar/Alpha 3.0 is used to skip over the initial part of execution. Results are then reported for simulating each program for 1 billion committed instructions.

Benchmark	Input	Number of Simulated Inst.
Gcc	integrate.i	1 billion
Vortex	lendian1.raw	1 billion
twolf	Ref	1 billion
Vpr	net.in, arch.in	1 billion

Table 2. Benchmark Programs

5.1 Results

The simulation results are shown in Figure 2. The maximum speedup is obtained for vortex and is 1.147. The average speedup over base is 10%.

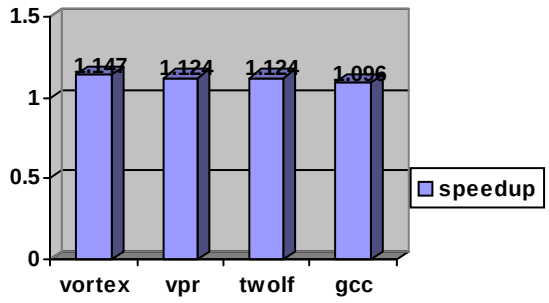

Figure 2 speedup over base

Table 3 shows the percentage of predictions by VP and percentage of hits in IR over the total number of instructions committed. Total predictions refer to percentage of instructions for which confidence is above the threshold. Total mispredictions refer to the percentage of instructions with mispredicted values. Used Predictions refer to the percentage of instructions for which the predicted values are used. Predicted values may be unused if the instruction hits the reuse buffer with actual operands or with predicted operands having higher hit confidence. Reuse Buffer Hits refer to the total number of hits in the reuse buffer. Used reuse buffer hits refer to the number of reuse buffer hits utilized. Reuse buffer hits may be unused if the buffer is hit with predicted operands with less confidence.

Benchmark	vortex	Vpr	twolf	gcc
Total Predictions	17.6	19.7	20.7	2.7
Total Mispredictions	2.4	4.2	3.3	0.04
Used Predictions	16.6	18.9	20.5	2.6
Used Predictions Mispredicted	2.3	4.1	3.2	0.04
Reuse Buffer Hits	4.7	4.5	5.4	6.2
Used Reuse Buffer Hits	4.6	4.4	5.3	6.2

Table 3. Percentage of Predictions and Reuse Buffer Hits

6 Conclusions

This paper proposes an effective way of combination of VP and IR. Adding an additional stage in the pipepline minimizes the value misprediction penalty. It is also possible to access the reuse buffer in the confirm stage so that the reexecution can be bypassed. By using more aggressive value prediction techniques better performance can be achieved. Further the reuse buffer replacements can be guided by the prediction confidence of the operands, which are being updated in the buffer.

Deeper pipelines with largely separated issue and writeback stages may increase the value misprediction penalty. This may seem to be a great disadvantage but the fact that deeper pipelines help in achieving speedup due to reuse buffer hits will alleviate the large misprediction penalty. Therefore it does make a lot of sense to combine both VP and IR.

References

[1] F. Gabbay and A. Mendelson. Speculative Execution based on Value Prediction. Technical Report EE Department TR 1080, Technion - Israel Institute of Technology, Nov. 1996.

[2] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. Value Locality and Load Value Prediction. In *Proc. of 7th International Conference on Architectural Support for Programming Languages and Operating System*s, Sept. 1996.

[3] Y. Sazeides and J. E. Smith. The Predictability of Data Values. In *Proc. of 30th Annual international Symposium on Microarchitecture (MICRO-30*), Dec. 1997.

[4] A. Sodani and G. S. Sohi. An Empirical Analysis of Instruction Repetition. In *Proc. of 8th International Conference on Architectural Support for Programming Languages and Operating System*s, Oct. 1998.

[5] M. H. Lipasti and J. P. Shen. Exceeding the Dataflow Limit Via Value Prediction. In *Proc. of 29th International Symposium on Microarchitectur*e, pages 226–237, Dec. 1996.

[6] A. Sodani and G. S. Sohi. Dynamic Instruction Reuse. In Proc. of 24th Annual International Symposium on Computer Architecture, pages 194–205, July 1997.

[7] A. Sodani and G.S. Sohi. Understanding the Differences Between Value Prediction and Instruction Reuse. In Proc. of the 31-th Annual International Symposium of Microarchitecture, Nov. 30 - Dec. 2, 1998, Dallas, USA.

[8] "Mixed Value Predictor Using Data Speculation and Reuse" Adrian Maxim, Carmen Turinici Cirrus Logic Inc. USA, "Gh. Asachi" University, Iasi, Romania

[9] K. Wang and M. Franklin. Highly accurate data value prediction using hybrid predictors. In *30th Annual International Symposium on Microarchitectur*e, December 1997.

[10] Brad Calder Glenn Reinman Dean M. Tullsen. Selective Value Prediction. In the Proceedings of the 26th International Symposium on Computer Architecture, May 1999.

[11] Implementations of Context-Based Value Predictors Yiannakis Sazeides and James E. Smith Department of Electrical and Computer Engineering University of Wisconsin-Madison

[12] F. Gabbay and A. Mendelson. The effect of instruction fetch band-width on value prediction. In 25th Annual International Symposium on Computer Architecture, 1998.

[13] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-97-1342, University of Wisconsin, Madison, June 1997