Arquitetura de computadores
Arquitetura super escalares
Processamento paralelo
393248

Improving Performance through Speculative Trace Reuse

C N P q 1.03.03.00-6

Maurício L. Pilla[1]
Philippe O. A. Navaux[2]
Felipe M. G. França[3]
Amarildo T. da Costa[4]
Bruce R. Childers[5]
Mary Lou Soffa[6]

Abstract

Trace reuse improves the performance of processors by skipping the execution of sequences of redundant instructions. However, many reusable traces do not have all of their inputs ready by the time the reuse test is done. For these cases, we developed a new technique called Reuse through Speculation on Traces (RST), where trace inputs may be predicted, allowing more traces to be reused than conventional reuse techniques, without increasing hardly the hardware needed. Preliminary results show a potential for speedups of 1.43 over a non-speculative trace reuse technique (harmonic mean over DTM).

1 Introduction

Modern processors devote significant resources to exploit instructio-level parallelism (ILP) from programs, but control and data dependencies still remain a barrier to effectively exploiting large amounts of ILP. Indeed, the additional complexity introduced by very wide issue widths may adversely impact the processor clock rate. What is needed are complexity-effective techniques that can increase performance by overcoming or mitigating the impact of control and data dependencies.

It is well known that programs execute a large amount of redundant or predictable computations (BODIK; GUPTA; SOFFA, 1999; GABBAY; MENDELSON, 1996; LIPASTI; SHEN,

[1]pilla@inf.ufrgs.br PhD Scholarship from CNPq
[2]navaux@inf.ufrgs.br
[3]felipe@cos.ufrj.br
[4]amarildo@cos.ufrj.br
[5]childers@cs.pitt.edu
[6]soffa@cs.pitt.edu

1999; SAZEIDES; SMITH, 1997; SODANI; SOHI, 1998). Many techniques have been developed to take advantage of redundancy to improve performance by not executing redundant computations. Value reuse is one technique that exploits redundant computations by reusing previously computed values. In value reuse, once a computation is executed, future executions can check if the input values needed by the computation are the same as the previous execution. If the inputs match previous inputs, then the result of the computation can simply be reused without computing the result. However, the input values needed by the computation must be available when checking for reuse.

It is also well known that many of the values during program execution can be predicted correctly (GABBAY; MENDELSON, 1996; LIPASTI; SHEN, 1999; SAZEIDES; SMITH, 1997). By predicting values, the impact of true data dependencies can be mitigated by letting more instructions to execute in parallel. Value prediction may also hide high latencies. When value prediction is employed, unlike reuse, the predicted value must be validated by actually computing the value and checking it against the predicted value. On a misprediction, any computation using the predicted value directly or indirectly has to be recomputed.

Value reuse is conservative because computations are reused only when the inputs are known, which may delay or limit the use of value reuse. Value prediction is aggressive, but there may be a high penalty to recover from mispredictions. It is possible for the high misprediction penalty to overwhelm any benefit of value prediction, or in some cases, to even hurt performance.

In this paper, we present our technique, called Reuse through Speculation on Traces (RST), that combines both value reuse and value prediction for instruction traces. The goal of RST is to increase the number of instruction traces that can be reused by predicting the values of trace inputs that are not available when applying reuse. In the next section, we discuss our approach, and in the last session, we provide some final remarks and future works.

2 Reuse through Speculation on Traces

The key idea of RST is to speculate some of the input values of a trace if they are not ready, rather than waiting for their computation to end or not applying reuse. Value prediction is done when some of the input trace registers match stored values and other input values are not available. It is these latter values that are predicted. RST is an integrated mechanism that combines trace reuse and value prediction. It is also designed to be a complexity-effective approach using most of the hardware that is already present for trace reuse.

Traditional value reuse is non-speculative. After the input values of a set of instructions are verified against stored values and a match is found, their results can be reused without executing the instructions. Importantly, resources are not wasted due to reuse and are available to other instructions. The results of the set of instructions that were previously computed and stored can be immediately written to their destinations. The main disadvantage is that reuse

must wait until all the input values are ready to be tested for reuse. Therefore, many cycles that could be saved by reusing instructions may be spent waiting for input values that were not ready at the time of the reuse test.

On the other hand, value prediction can overcome the limits imposed by true data dependencies (LIPASTI; SHEN, 1999; SAZEIDES; SMITH, 1997). Instructions with true data dependencies may be executed in parallel when value prediction is employed. This technique may also hide latencies of instructions accessing memory or with high complexity. The main disadvantage is that mispredictions can incur a high recovery penalty. In fact, the misprediction penalty can be very high when there is significant instruction level parallelism. Another disadvantage is that, since value prediction increases concurrency and demands for resources, instructions executing with mispredicted values may prevent the execution of useful instructions.

Trace reuse has been proposed to improve performance by not computing redundant sequences of instructions (COSTA; FRANÇA; CHAVES FILHO, 2000). The three stages of trace reuse are shown in Figure 1. The *reuse domain* is defined as the set of instructions that can be reused and do not present side effects. First, in Figure 1(a) instructions in the reuse domain are identified (gray circles) and stored. In the next execution shown in Figure 1(b), these instructions are marked as redundant and a trace is formed, until an instruction that does not belong to the reuse domain or is not redundant is found (black circle). This trace is memoized and stored in a memoization table. Figure 1(c) shows the next time execution reaches the beginning of this trace with the same inputs, when the memoized trace is reused; i.e., the previous values are written in the output registers. In this example, the input registers compared are *r1*, *r2*, *r3* and *r9* using stored values for these registers. If the inputs match, the values stored for *r5*, *r6*, *r7* and *r9* are loaded into these registers as the outputs of the trace. Thus, all instructions inside the trace are essentially collapsed into the checking of the inputs and storing of the outputs. The instruction fetch is redirected to the next address after the trace. RST combines the advantages of both value prediction and reuse. Unavailable inputs

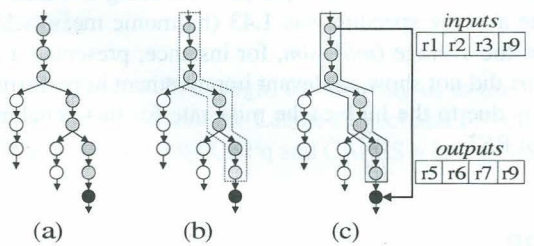

Figure 1: Trace: (a) identification and construction, (b) memoization, (c) reuse

for memoized traces (input and output values stored) are predicted by RST. When traces are

reused speculatively in RST, the output values are sent directly to the commit stage, as well as to the instructions waiting for these values and to the register file. Dispatch, issue, and execution are bypassed for the *entire* trace in a single cycle. Therefore, speculative reuse does not increase but reduces the pressure on valuable resources such as functional units.

Applying reuse and value prediction separately but at the same time could require a prohibitive amount of storage in tables. Because we integrate the techniques, RST does not need extra tables to store values to be predicted. The input context of each trace (the input values of all instructions in the trace) already stores the values for the reuse test, which may also be used for prediction. Thus, our proposed technique minimally increases the hardware needed to implement speculative trace reuse, when compared to the hardware needed for non-speculative trace reuse.

RST may reuse both instructions and traces, but only traces are speculatively reused because they encapsulate many instructions and possibly critical paths, thus allowing more performance improvement than single instructions.

Compared with instruction reuse techniques (ROTH; SOHI, 2000; SODANI; SOHI, 1998), RST has all the benefits of trace reuse, such as the potential for collapsing critical paths into a single cycle, improving branch predictions, and reducing the fetch bandwidth needed. It is also simple to implement as it does not need to involve the compiler or ISA modifications such as needed in block and sub-block reuse (HUANG; LILJA, 2000; WU; CHEN; FANG, 2001), allowing the execution of legacy code without modifications. Unlike other trace reuse mechanisms (COSTA; FRANÇA; CHAVES FILHO, 2000; GONZALEZ; TUBELLA; MOLINA, 1999), RST can speculatively reuse traces when inputs are not ready. Previous value prediction techniques (GABBAY; MENDELSON, 1996; LIPASTI; SHEN, 1999; WANG; FRANKLIN, 1997; SAZEIDES; SMITH, 1997; WU; CHEN; FANG, 2001) use more resources when mispeculations occur, while RST is more conservative: predicted traces are not executed, but speculatively reused.

Figure 2 shows the preliminary speedups obtained with virtually unlimited reuse tables over a non-speculative trace reuse technique (DTM), using a 4-instructions wide pipeline with 20 stages. The average speedup was 1.43 (harmonic mean). Most benchmarks had better speedups than the average (*m88ksim*, for instance, presented a speedup of 2.19 over DTM), while only *art* did not show a relevant improvement in performance, with a speedup of only 1.09. This is due to the high cache miss rate for this benchmark, which hides the performance gains of RST.

3 Conclusion

This paper presented a new approach to reuse traces in processors, Reuse through Speculation on Traces (RST). Our technique can effectively reuse more traces than non-speculative approaches, increasing performance with speedups of about 1.43 over DTM (harmonic mean).

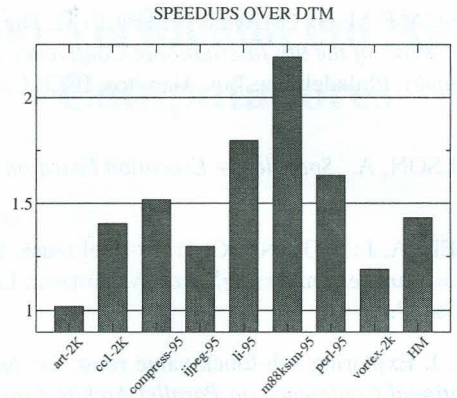

Figure 2: Speedups of RST over DTM

A more detailed study about the limits of our proposal can be found in (PILLA, 2003).

Our technique does not require changes in the instruction set, allowing direct execution of legacy codes. RST does not increase significantly the hardware needed to implement non-speculative trace reuse, therefore it is simpler and more easily implementable than just combining two unrelated reuse and prediction mechanisms. RST is also more conservative than previous value prediction techniques, as traces are not executed but reused. Therefore, more resources are kept free to execute other instructions.

Future works include the definition of effective policies for confidence estimation to limit mispeculation penalties and the study of how different reuse domains affect performance, for example, by allowing loads and stores in traces.

Acknowledgments

We thank Dr. Mossé (Univ. of Pittsburgh) for allowing us to use his cluster to simulate part of the experiments. We also thank the Labtec Project and FINEP for the same reason. This work was partially developed with CNPq and CAPES scholarships.

References

BODIK, R.; GUPTA, R.; SOFFA, M. L. Load-reuse analysis: Design and evaluation. In: *SIGPLAN Conference on Programming Language Design and Implementation*. [S.l.]: New York, ACM, 1999. p. 64–76.

COSTA, A. T. da; FRANÇA, F. M. G.; CHAVES FILHO, E. M. The dynamic trace memoization reuse technique. In: *Proc. of the 9th International Conference on Parallel Architectures and Compilation Techniques*. Philadelphia: Los Alamitos, IEEE Computer Society, 2000. p. 92–99.

GABBAY, F.; MENDELSON, A. *Speculative Execution based on Value Prediction*. Israel, 1996.

GONZALEZ, A.; TUBELLA, J.; MOLINA, C. Trace-level reuse. In: *Proc. of the 28th International Conference on Parallel Processing*. Aizu-Wakamatsu: Los Alamitos, IEEE Computer Society, 1999. p. 30–37.

HUANG, J.; LILJA, D. J. Exploring sub-block value reuse for superscalar processors. In: *Proc. of the 9th International Conference on Parallel Architectures and Compilation Techniques*. Philadelphia: Los Alamitos, IEEE Computer Society, 2000. p. 100–110.

LIPASTI, M. H.; SHEN, J. P. Exceeding the dataflow limit via value prediction. In: *Proc. of the 29th Annual International Symposium on Microarchitecture*. Paris: Los Alamitos, IEEE Computer Society, 1999. p. 226–237.

PILLA, M. L. et al. *The Limits of Speculative Trace Reuse*. Porto Alegre, abr. 2003.

ROTH, A.; SOHI, G. S. Register integration: A simple and efficient implementation of squash re-use. In: *Proc. of the 33rd Annual International Symposium on Microarchitecture*. Monterey: Los Alamitos, IEEE Computer Society, 2000. p. 223–234.

SAZEIDES, Y.; SMITH, J. E. The predictability of data values. In: *Proc. of the 30th Annual International Symposium on Microarchitecture*. [S.l.]: Los Alamitos, IEEE Computer Society, 1997. p. 248–258.

SODANI, A.; SOHI, G. S. Understanding the differences between value prediction and instruction reuse. In: *Proc. of the 31st Annual International Symposium on Microarchitecture*. [S.l.]: Los Alamitos, IEEE Computer Society, 1998. p. 205–215.

WANG, K.; FRANKLIN, M. Highly accurate data value prediction using hybrid predictors. In: *Proc. of the 30th Annual International Symposium on Microarchitecture*. [S.l.]: Los Alamitos, IEEE Computer Society, 1997. p. 281–290.

WU, Y.; CHEN, D.-Y.; FANG, J. Better exploration of region-level value locality with integrated computation reuse and value prediction. In: *Proc. of the 28th Annual International Symposium on Computer Architecture*. Göteborg, Sweden: New York, ACM, 2001. p. 98–108.