2009 International Conference on Computer Engineering and Technology

A feasibility study on hyperblock-based aggressive speculative execution model

Ming Cong, Hong An, Yongqing Ren, Canming Zhao, Jun Zhang
Department of Computer Science and Technology
University of Science and Technology of China
Hefei, 230027, China
Key Laboratory of Computer System and Architecture
Chinese Academy of Sciences
Beijing, 100080, China
mcong@mail.ustc.edu.cn, han@ustc.edu.cn, {renyq, zcm, junzh}@mail.ustc.edu.cn

***Abstract*—*Speculation execution model which executes sequential programs in parallel through speculation is an effective technique for making better use of growing on-chip resources and exploiting more instruction-level parallelism of applications. However, accompanied high communication overheads and roll-back penalties can not be neglected. This paper focuses on analyzing the feasibility of aggressive speculation execution model and finding an appropriate degree of "aggressiveness" under hyperblock-based execution model. We analyze the characteristic of control dependences and data dependences between adjacent hyperblocks, and propose a quantitative analysis method to detect data dependences on hyperblock-based execution model, and then evaluate the feasibility of aggressive speculative execution model on 8 applications from SPEC2K. Our experiments show most applications can get high prediction accuracy on control-flow from hyperblock-based prediction mechanisms, especially SPECFP. Furthermore, we analyze factors which impact expected prediction depth and find depth depends more on application than predictors.*

Keywords-hyperblock; speculative execution; prediction; control dependence; data dependence

I. INTRODUCTION

Modern CMOS technology brings the increasing number of transistors on one chip, so how to effectively utilize the growing resources and exploit more parallelism to accelerate applications is an urgent problem for computer architects. However, to expose potential of instruction-level parallelism (ILP), control-flow and data-flow constraints inherent in a program must be overcome. Speculative execution [1] which executes programs aggressively has become a mainstream technique to reduce the impacts of dependences in high performance microprocessors.

The *block-based* execution model [4] has been proposed to enlarge the instruction window, which may achieve high ILP and high resource utilization. Recent works on computer architectures, such as TRIPS and Multiscalar, use *block-atomic*(tasks in Multiscalar [4]) execution, in which each block is fetched, executed, and committed atomically, behave like a conventional processor with sequential semantics at the block level. A *hyperblock* [2][3] is a set of predicated basic blocks combined by compiler in which control flow only enters from the top, but may exit from one or more locations. With hyperblock, larger instruction window and more ILP can be achieved than basic block.

Similar to the multi-issue in superscalar, *block-based execution model* utilizes additional resources to execute more blocks simultaneously on the processor substrate, with all but one executing speculatively. It's obvious that the feasibility of aggressive execution model depends largely on the effectivity and prediction accuracy of speculation. However, as speculation in high-ILP processors become more aggressive, the number of mis-speculation increases with growing numbers of inflight instructions/ blocks. Although many mechanisms have been proposed for speculative execution, efficiency is still limited because of mis-speculation penalties, high communication overheads and etc.

This paper focuses on the analysis of hyperblock-based aggressive execution model. We concentrate on finding a tradeoff and maximum potential of aggressive execution that can be exploited, and analyze dependent behaviors of applications under this model. At the aspect of control-flow, we evaluate performance of three branch predictors, and estimate the expected prediction depth, and then analyze the effecting factors of aggressive speculation with control-flow. In the view of data-flow, we propose a quantitative analysis of data dependence on hyperblock-based execution model, and analyze the distributions of data dependences between hyperblocks and their impacts on the depth of prediction.

We evaluate the feasibility of aggressive speculative execution model on 8 applications from SPEC2K. Our experiments show that most applications have good predictability, and high prediction accuracy on control-flow can be gained by using hyperblock-based branch prediction mechanisms, especially SPECFP applications. Furthermore, we analyze factors which impact expected prediction depth, and find that it depends more on applications themselves than predictors, and expected prediction depth differs depending on the characteristics of each application.

The rest of this paper is organized as follows. Section II describes the aggressive execution model in detail, and analyzes it in the aspects of both control dependences and data dependences. Section III experimentally evaluates and analyzes the feasibility of aggressive execution, which corresponds to the part in section II. Section IV introduces related works on aggressive execution models. Finally in section V we make our conclusions.

II. HYPERBLOCK-BASED AGGRESSIVE EXECUTION MODEL

Current researches of ILP processors focus on exposing more inherent parallelism in an application to obtain higher performance. To effectively exploit ILP in programs, the

978-0-7695-3521-0/09 $25.00 © 2009 IEEE
DOI 10.1109/ICCET.2009.30

dependences between instructions need to be detected and avoided to prevent pipeline stalls, among which control dependence and data dependence are of the most importance.

A. Speculation execution on the control-flow

1) Differences between hyperblock branch predictor and conventional predictor

Branch predictions are made based on branch history information, so that the accuracy of predictors depends on the frequency of historical representation of previous branch target address. Branch predictions in hyperblock-based model have high parallelism, but their mechanisms are much different with those in conventional superscalar processors from following aspects.

First, in superscalar model, each branch has only one exit so that one bit of exit information is enough for representing *taken* or *not taken* (T/NT); while in block-based model, each block has several exit points, predicting the exit taken out of multiple possible exits is a multi-way branching problem, some bits of exit ID or target address of the branch should be reserved as the exit information.

Secondly, *dedicated adder* in the fetch mechanism and *branch target buffer* (BTB) of RISC architectures can be used to compute PC relative target addresses before they are computed by ALU(s) in execution stage; but in hyperblock-based model, the variable block sizes and different target addresses which exit may correspond to force us to predict target addresses of all exit point.

Thirdly, A *return address stack* (RAS) used for return address prediction makes it more accurate, and the type of branch instruction can be easily got from a pre-decoder in RISC architectures, but the type of exit points in blocks is hard to get before block committing. So we need a mechanism to predict the type of exit points.

2) Design space of hyperblock-based branch predictors

In this section we describe the design space of hyperblock-based branch predictors in detail. Based on conventional branch predictors and the characteristics of hyperblocks, we consider a two-level predictor [5] which predicts the first branch that will be taken in a hyperblock. As shown in Figure 1, the first level predicts the exit point, and then the second level produces the branch target address.

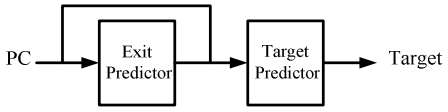

Figure 1. 2-level hyperblock-based branch predictors

a) Exit predictor: Corresponding to branch behaviors of conventional branch predictors, the predictors for exit [7] can be organized around following methods.

Global predictor: Global predictor indexes the table of bimodal counters with the recent global history integrated with branch instruction addresses to get branch behavior. In hyperblock-based model, we replace the T/NT stored in *Pattern History Table* (PHT) with exit numbers of each block; branch behaviors in *History Register Table* (HRT) are also replaced by recent exit history of blocks. In addition, for the accuracy of prediction, we use two HRT, one for updating the prediction information, the other for recovering.

Local predictor: Local branch predictor keeps two tables. One is a local BHT indexed by the low-order bits of each branch instruction's address; it records T/NT history of N most recent executions of each branch, which is different from global predictor. The other is a PHT indexed by the generated value from the branch history in BHT. Local prediction is slower than global prediction because it requires two subsequent table lookups for each prediction, but it may be favorable for deep prediction of a certain local branch.

Tournament predictor: Since different branches may be predicted better with either local or global techniques, tournament predictor uses a choice predictor to dynamically select the best method between two prediction for each branch, it is nearly as accurate as local predictor, and almost as fast as global predictor. The choice prediction is made from the table of 2-bit prediction counters indexed by path history; processor trains counters to prefer the correct prediction whenever the local and global predictions differ.

b) Block target address prediction

Block branch target address determined by the exit instruction is the starting block address of next task. After the exit is predicted, we use both predicted exit and branch address to determine the target. Conventional methods usually use BTB and RAS, in which BTB handles prediction of branch and call exit, RAS predicts address of return exits with known types of each exit, this will certainly degrade prediction accuracy and increase its complexity. So we only use BTB for the prediction, which is easy and versatile.

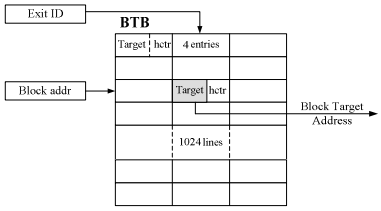

Figure 2. Structure of the BTB

As shown in Figure 2, each item of BTB maintains target addresses of several exit and hysteresis bits, indexed by the block address and exit ID which was predicted by branch predictor, to obtain the Block Target Address.

3) Evaluation of three branch predictors

In this section we evaluate prediction accuracy of exit predictors with block target address predictor presented previously on 8 benchmarks from SPEC2000. Both Global predictor and Local predictor are configured to be 16384

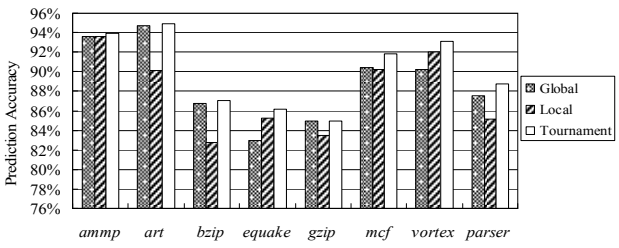

Figure 3. prediction accuracy of three branch predictors

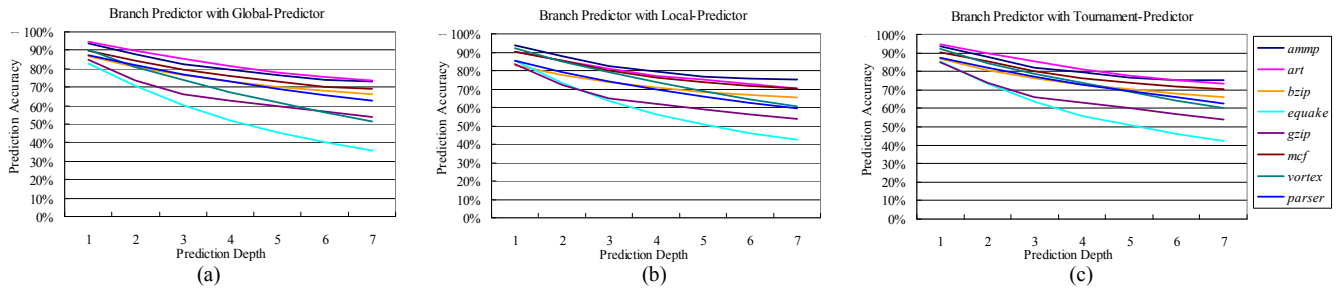

Figure 4. Prediction accuracy of different branch predictor

entries, Tournament predictor contains 8192 entries for both global and local predictor; The BTB has 1024 entries.

From Figure 3, it is clear that global predictor performs better than local predictor on *art*, *bzip*, *gzip* and *mcf* in which global histories is predominant; but *equake* and *vortex* on which local histories have more impact perform better with the local predictor. The tournament predictor takes full advantage of global and local histories, achieves better performance than both of them.

B. *Speculative execution on the Data-flow*

Data dependence is an important factor that influences effectivity of multi-level speculative execution. In this section, we conduct a quantitative analysis of data dependence on hyperblock-based execution model, and analyze its impact on the depth of prediction. To be able to obtain benefits from data speculation, we must execute instruction streams correctly, so we make two assumptions before the analysis: (1) A perfect branch predictor is assumed. (2) No complexity of hardware implementation is taken into accounts. These assumptions would not affect the analysis of the natural characteristic of programs.

The dependent relation between instructions in traditional processors can be described by dependence-distance (the number of instructions between data producers and consumers); but it is no longer applicable for measuring hyperblock-based model, because different impacts of instructions on separating producer and consumer, and instructions in the same block can execute in parallel.

$$Dependence\ depeth = \sum_{i=1}^{\infty} P_i \times i \qquad (1)$$

We define the term "*Dependence depth*" to measure the degree of dependences between blocks. There i denotes the distance from current block; P_i denotes the proportion of instructions which have dependence with instructions in the previous i_{th} block to the total number of instructions within current block. Dependence depth describes the dependence strength between blocks. The larger dependence depth means the weaker dependent strength, so that we can exploit more parallelism from inter-blocks. And it is proportional to the potential depth of speculated execution, which means programs could benefit from speculating more blocks according to larger dependence depth.

III. EXPERIMENTAL EVALUATION

A. *Methodology*

Our experiments are performed on the TRIPS toolchain which supports hyperblock-based multi-level speculation. TRIPS is a block-atomic execution model, the size of instruction window can reach up to 1024 by speculation. It contains compiler (Scale [3]), functional simulator tsim_arch and cycle-accurate simulator tsim_proc [6]. Tsim_proc can generate trace files containing all events of the simulating process. Although our experiments are based on TRIPS, our research is not limited to this architecture but aims at all current hyperblock-based execution models. We use 8 whole benchmarks written in C from SPEC2000 benchmark suite, including 3 float point benchmarks: *art*, *ammp*, *equake*, and 5 integer benchmarks: *gzip*, *mcf*, *parser*, *vortex, bzip2*.

B. *The Speculative execution on control dependence between hyperblocks*

Large instruction window is built to issue more independent instructions per cycle, but the effective size of instruction window is limited by the depth of control-flow speculation, so we evaluate the feasibility of predicting more aggressively and analyze the appropriate depth of prediction.

First, we evaluate prediction accuracy with different prediction depths; then we depict the depth distribution and predictability for all applications; finally, we analyze the expected prediction depth of different applications based on instructions distribution, which can help us adopt appropriate prediction depth while studying aggressive execution model, and we analyze impacts of expected prediction depth on different predictors and configurations.

1) *Evaluate predictors with certain prediction depth*

We evaluate prediction accuracy of global and local predictors with certain prediction depths (range from 1~7), as in figure 4, the prediction accuracy decreases while prediction depth increases, but the depressive gradient slows down with deeper depths, and most applications still keep a high prediction accuracy even the prediction depth is up to 7. This demonstrates deeper prediction is feasible to most applications. Global and local predictor both have strengths on different types of applications with small depths, but local predictor performs better with the increase of depth, which indicates that it is more powerful for deeper speculative execution. Tournament predictor performs best as it can adapt to pattern of applications with both previous predictors.

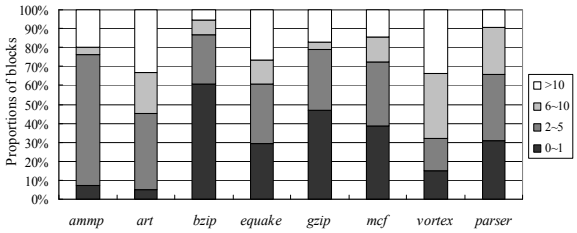

Figure 5. Proportion of blocks VS. prediction depth

2) Distribution of Prediction depth on blocks

We introduce a quantitative method for denoting the potential of deep prediction of distinct applications intuitively. We predict each block with unbounded depths until the prediction cannot continue; thus each application can be divided into several block sequences with various block numbers. According to a statistical analysis of different prediction depths among various sequences, we can estimate the potential of deep prediction for applications.

Figure 5 illustrates the proportion of blocks that predicted with different prediction depths (0~1, 2~5, 6~10, 10~∞) on tournament predictor, among which the depth 0 indicates blocks cannot be predicted. *Vortex, art* and *ammp* have high proportions with deeper prediction depth; *bzip* and *gzip* are just the opposite in which the former group can be predicted aggressively and the latter are less predictable.

3) Expected Prediction Depth

Prediction depth distribution can not reflect actual quality of prediction depth for applications, so we further introduce the *Expected Prediction Depth*, mean value of prediction depth distribution that reflects the magnitude of prediction and even prediction accuracy.

Figure 6 shows the expected prediction depth evaluated under the condition that size of local and global PHT is configured to 16348, we can see that the types of predictors have big impact on the expected prediction depth. For each predictor, applications has distinctive results, we can see values of *ammp, art*, *mcf and vortex* exceed 10, and others are around 5, which also shows that these applications can accommodate well to such prediction model.

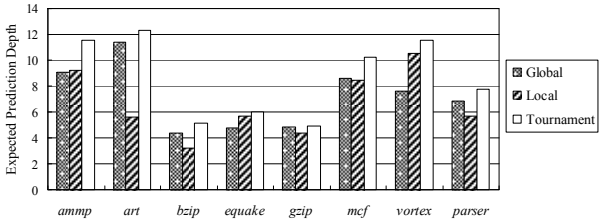

Figure 6. Expected prediction depth

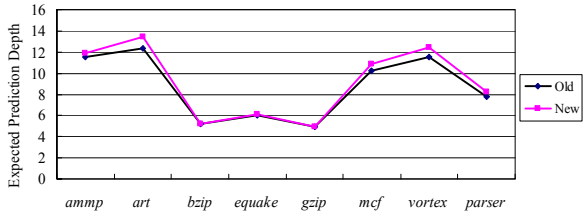

Figure 7. Expected prediction depth with different configured predictor

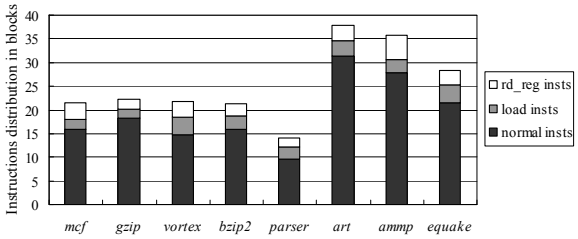

Figure 8. Numbers of dependence instructions

Figure 7 describes the expected prediction depth with tournament predictor which is configured with 16x the size of old PHT. From the results, we can see configurations of predictors have less impact on the expected prediction depth. In most applications, low prediction accuracy is not made by the conflict on PHT, but comes from the characteristic of applications, we should consider more improvements on the predictors in order to better adapt to patterns of applications.

C. The Speculative execution on Data dependence

1) Numbers of dependent instructions in applications

Data dependence between hyperblocks arises from the *load* or *register-read* instructions (dependent instructions). Figure 8 presents a statistic of the numbers of overall instructions, *load* instructions and *register-read* instructions within hyperblocks. Although these values are closely related to compiles, we can still find that the number of dependent instructions is considerable in hyperblock, from approximately 18% in *gzip* to 52% in *ammp*. We would not achieve the anticipated speedup if we only increase speculative depths without considering the impact of data dependence.

2) Distribution of data dependences

Previous statistic of instructions number only reflects the ubiquitous of data dependence under block-level execution model, but cannot completely substitute for the data dependence behaviors. Results may be inequable while a block has data dependences with blocks located in different place, so we further analyze the distribution of data dependence (Figure 9). Most data dependence of a hyperblock comes from its previous adjacent blocks, our experiments show 20% or even 40% [*parse*] of data dependence locates in two adjacent blocks, and an average of 40% or even 60% [*parse*] locates in six contiguous blocks. These differences are totally attributed to the natural characteristics of applications.

The unbalanced distribution on data dependences is not what we expect, as we cannot make subsequent blocks executed even increasing the speculation depth because of the serious dependence between adjacent blocks.

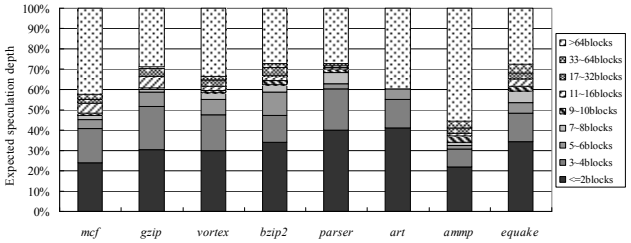

Figure 9. Distribution of data dependence in applications

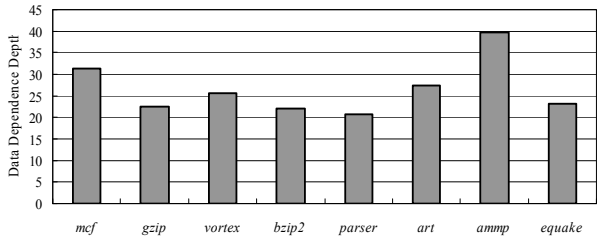

Figure 10. Data dependence depth in applications

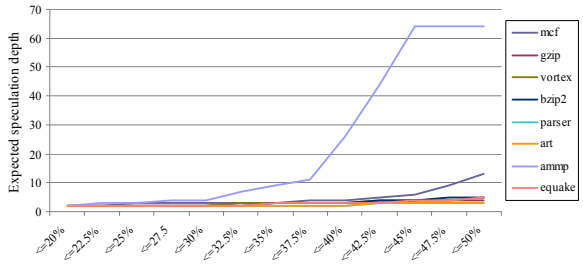

Figure 11. Expected speculation depth VS. Different dependence depth

3) Dependence Depth and Expected Prediction Depth

Figure 10 shows dependence depth of applications, figure 11 shows expected speculation depth under the constraints of certain data dependence proportion. As in *equake*, the expected speculation depth is only up to 5 with less than 47.5% data dependence because data dependence within 6 blocks is 48.696% (Fig. 9), bigger than 47.5%. Applications of SPECINT have low speculation depths less than 10(4~8 on average), in contrast, applications of SPECFP have better speculation depth that up to 64 with 50% data dependence.

IV. Related Work

Speculative executions on hyperblocks mainly focus on two directions. One is resolving control-flow between hyperblocks such as Multiple Branch Predictors and Region Predictors, the other is Predication which converts control dependence to data dependence by merging multiple control flows into a single control flow.

Yeh et al. [5], Seznec et al. and Conte et al. studied multi-branch predictors that typically predict 2 or 3 targets at a time, then Wallace et al. used saturating counters to predict multiple branches. Exit predictor is a region predictor first proposed by Pnevmatikatos et al. [8] in the context of Multiscalar processor, and subsequently refined by Jacobson et al. [7], exit predictors predict only the first branch that will leave a code region (such as a Multiscalar task). The local, global and path-based exit predictors, folding of exit histories and hysteresis bits in PHT we used in this paper are proposed from Jacobson et al.

Predication of individual instructions was first proposed by Allen et al. in 1983 and implemented in the wide-issue Cydra-5. Mahlke et al. [4] first developed the modern notion of hyperblock, extended by August et al. who proposed a framework to balance control speculation and predication through smart basic block selection at hyperblock formation. Dynamic predication [3] predicates dynamically at run-time and allows predicating without predicated ISA support.

V. Conclusions

Hyperblock-based execution model can tremendously improve the size of instruction windows for high ILP and good performances. Since enlarging a hyperblock would become more difficult under the constraints of compiler technology and inherent characteristics of applications, we introduce a preliminary evaluation of aggressive execution model. Our experiments concentrate on the characteristics of dependence distribution and prediction depth in views of speculative execution both on control-flow and data-flow.

Under this model, most applications have good predictability, high prediction accuracy and expected prediction depth with control-flow prediction mechanisms; and expected prediction depth which reflects the degree of predictability mainly depends on applications themselves rather than predictors. Moreover, although prediction accuracy on control-flow decreases while prediction depth increases, it remains high within acceptable bounds. Finally, we introduce a quantitative method for denoting speculative execution potentials for distinct applications intuitively, and analyze the distributions of data dependences between hyperblocks and their impacts on depth of prediction. As we observes, many applications have high expected prediction depth, but whether they are suitable for aggressive executing depends on data dependences between adjacent hyperblocks.

Acknowledgement

This research was supported financially by the National Basic Research Program of China under contract 2005CB321601, the Natural Science Foundation of China grant 60633040, the National Hi-tech Research and Development Program of China under contract 2006AA01A102-5-2, the China Ministry of Education & Intel Special Research Foundation for Information Technology under contract MOE-INTEL-08-07.

References

[1] A. Uht, V. Sindagi, and K. Hall, "Disjoint eager execution: An optimal form of speculative execution," Micro-28, Dec. 1995, pp. 313-325.

[2] T. N. Vijaykumar. “Compiling for the Multiscalar Architecture”. In Doctor of Philosophy at the university of Wisconsin 1998.

[3] Aaron Smith, Jon Gibson, Bertrand A. Maher, Nicholas Nethercote, Bill Yoder, Doug Burger, Kathryn S. McKinley, “Compiling for EDGE Architectures”. In Proceedings of the International Symposium on Code Generation and Optimization, 2006, pp. 185-195.

[4] S. A. Mahlke, D. C. Lin, W. Y. Chen, R. E. Hank, and R. A. Bringmann, “Effective Compiler Support for Predicated Execution Using the Hyperblock”, In Proceedings of the 25th International Symposium on Microarchitecture, Dec. 1992, pp. 45-54.

[5] T.Y. Yeh and Y. Patt. Two-level adaptive branch prediction. In Proceedings of the 24th International Symposium on Microarchitecture, pages 51–61, 1994.

[6] TRIPS toolset, available from: http://www.cs.utexas.edu /~trips/dist/

[7] Q. Jacobson, S. Bennett, N. Sharma, and J. E. Smith. Control flow speculation in multiscalar processors. In Proceedings of the 3rd International Symposium on High Performance Computer Architecture, Feb. 1997.

[8] D. Pnevmatikatos, M. Franklin, and G. S. Sohi. Control flow prediction for dynamic ilp processors. In Proceedings of the 26th Annual International Symposium on Microarchitecture, 1993.