Trace Substitution

Hans Vandierendonck, Hans Logie, and Koen De Bosschere

Dept. of Electronics and Information Systems
Ghent University
Sint-Pietersnieuwstraat 41
B-9000 Gent, Belgium
{hvdieren,kdb}@elis.rug.ac.be

Abstract. Trace caches deliver a high number of instructions per cycle to wide-issue superscalar processors. To overcome complex control flow, multiple branch predictors have to predict up to 3 conditional branches per cycle. These multiple branch predictors sometimes predict completely wrong paths of execution, degrading the average fetch bandwidth. This paper shows that such mispredictions can be detected by monitoring trace cache misses. Based on this observation, a new technique called trace substitution is introduced. On a trace cache miss, trace substitution overrides the predicted trace with a cached trace. If the substitution is correct, the fetch bandwidth increases. We show that trace substitution consistently improves the fetch bandwidth with 0.2 instructions per access. For inaccurate predictors, trace substitution can increase the fetch bandwidth with up to 2 instructions per access.

1 Introduction

Many applications have short basic blocks and branches that are difficult to predict. This property makes it difficult to sustain a high fetch bandwidth and thereby limits the maximum IPC. Trace caches offer a solution to this problem. Instructions that are executed consecutively are packed together into a trace and are cached as one unit for future reference. A multiple branch predictor predicts the followed path, which is used to retrieve the correct trace. A trace cache typically requires 3 branch predictions per cycle. The accuracy of the multiple branch predictor is of extreme importance, as trace caches are targeted at very wide-issue machines.

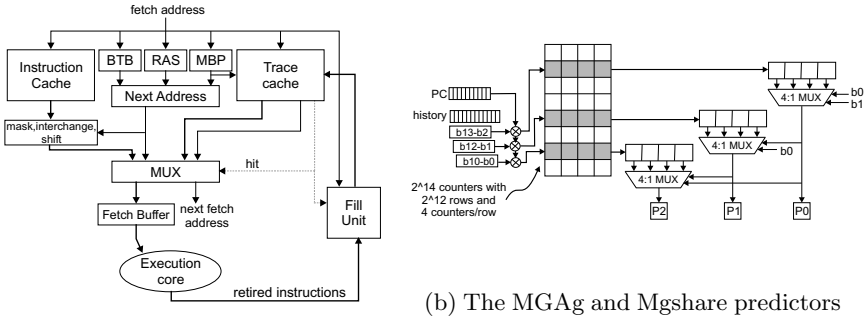
As branches are typically correlated to each other, certain paths or combinations of branches will be very unlikely and will almost never occur during the execution of a program. However, destructive interference in a multiple branch predictor may result in predicting these paths anyway. This paper studies the occurrence of such "obviously wrong" paths and shows that they are identified by trace cache misses. Based on this observation, we propose *trace substitution*, a simple and elegant technique to increase average fetch bandwidth. When an "obviously wrong" trace prediction is detected, a *cached* trace is substituted for the predicted trace. This technique, although very simple and requiring little additional hardware, consistently improves the fetch bandwidth.

The remainder of this paper is organised as follows. Section 2 discusses the trace cache and multiple branch predictor models and introduces trace substitution. Section 3 studies opportunities and limitations of trace substitutions and evaluates its impact for varying trace cache sizes and branch predictor sizes. Related work is discussed in section 4 and section 5 concludes the paper.

2 Trace Caches and Trace Substitution

2.1 The Trace Cache

The trace cache mechanism packs consecutively executed instructions into traces of instructions and stores them in the trace cache as a single unit [1, 2]. The trace cache is typically organised as a set-associative cache. Each trace consists of at most 16 instructions and 3 conditional branches.



(a) The fetch unit

(b) The MGAg and Mgshare predictors

Fig. 1. Block diagram of the fetch unit containing the trace cache, the multiple branch predictor and the instruction cache.

We use the trace cache organisation of [1, 3, 4] (Figure 1(a)). In this organisation, the instruction cache (a.k.a. core fetch unit) and the trace cache are probed in parallel with the fetch address. The multiple branch predictor predicts the direction of three conditional branches. The branch directions and the fetch address are used to perform tag matching in the trace cache, i.e., we use path-associativity [5], meaning that multiple traces can be present starting at the same program counter. When the trace cache hits, the trace is forwarded to the processor. On a miss, 16 instructions in two consecutive cache blocks are fetched from the instruction cache. The branch target buffer (BTB) and the predicted path are used to determine how many instructions should be forwarded to the processor. The instruction cache can deliver only 1 taken branch per access.

When the trace cache misses, the fill unit is instructed to start building a trace. The fill unit only builds traces on the correct path of execution using retired instructions.

2.2 Multiple Branch Prediction

The MGAg multiple branch predictor was proposed in [1, 6] (Figure 1(b)). In the MGAg, the global branch history is used as an index into a pattern history table (the PC is ignored in this case). The MGAg is a direct extension of the GAg branch predictor [7] to trace caches. Conceptually, the MGAg is a GAg that is accessed three times using the global branch history, which is updated between accesses with the predicted branch directions. This idea is implemented with three parallel pattern history table (PHT) accesses [6].

The Mgshare is derived from the MGAg predictor. In a gshare predictor that is iterated three times, the PHT is accessed each time using the exclusive-or (XOR) of the branch address and the global branch history [8]. When accessing a trace cache, only one branch address is available (i.e., the starting address of a trace). Hence, we XOR this address with each of the three PHT indices.

2.3 Trace Substitution

The trace cache only stores traces along paths of execution that have actually been followed. The multiple branch predictor has to predict the future path of execution and sometimes (due to destructive interference) it predicts paths that have not been executed yet. These paths are likely to be wrongly predicted and we can use the trace cache to detect them. We show in section 3.2 that a trace cache miss is a good indicator for incorrectly predicted branches.

As a trace cache miss is a tell-tale for branch mispredictions, its occurrence can be used to improve fetch bandwidth. We argued that traces stored in the trace cache are more likely to be on the correct execution path than other traces. Thus, when a trace cache miss occurs, we return a different trace instead. In particular, we return the most recently used trace with the same starting address. We call this technique *trace substitution*.

Trace substitution requires little additional hardware. First, the hit/miss indicator of the trace cache is reused to trigger trace substitution. Secondly, a second, parallel, tag match is required to find the most recently used trace with the requested starting address. Because the trace cache is indexed using this starting address, only one cache set needs to be searched. The most recently used trace is derived from the LRU bits of the replacement policy in the trace cache. The third hardware addition is that the multiplexor that selects the returned trace has to take into account the hit/miss indicator and the second tag match. This will probably increase the critical path of the trace cache with one multiplexor delay.

The fill unit builds a trace when a trace cache miss occurs. We modify this behaviour and do not build a trace when the substituted trace is on the correct path. It is still possible for new traces to enter the trace cache. When a new

trace is seen, then (i) a trace cache miss occurs, (ii) trace substitution is possibly applied and (iii) the substituted trace (if any) is on the wrong path, so the new trace is built after all and stored in the trace cache. If no trace could be found to substitute, then the trace is built and stored anyway.

3 Evaluation

We evaluate trace substitution using 6 SPECint95 benchmarks running training inputs. The harmonic mean is labelled as H-mean. The benchmarks are compiled with Compaq's C compiler using `-O4` optimisation. At most 500 million instructions are simulated per benchmark. The baseline fetch unit has a 256-entry trace cache organised as a 4-way set-associative cache. Each trace can contain up to 16 instructions and 3 conditional branches. Traces are terminated on indirect branches and system calls. The baseline MGAg predictor uses 12 bits of global branch history and the instruction cache is 32-KB large with 32-byte blocks and is 2-way set-associative.

3.1 The Branch Predictor Changes the Trace Cache Hit Rate

The overall performance measure FIPA (fetched instructions per trace cache access) depends on both the branch prediction accuracy and trace cache miss rate in a complex way. When the branch predictor is improved and the branch prediction accuracy increases, one also expects to see the FIPA increase. This is not always the case. We investigated a 12-bit history MGAg and Mgshare, an 8-kbit hybrid predictor that is optimistically probed three times in succession and a perfect branch predictor. Still other predictors were investigated, but are not shown. The predictors are evaluated in a large 16K-entry 4-way set-associative trace cache in order to mitigate the impact of trace cache misses (Figure 2). For many benchmarks, the Mgshare has a higher branch prediction accuracy but a lower FIPA than the MGAg predictor. The trace cache miss rate seems to correlate better to the FIPA, but the trace cache hit rate for the Mgshare predictor is unproportionally low.

These results clearly show that the branch predictor has a strong influence on the trace cache hit rate. When the branch predictor is perfect, the trace cache almost never misses. As the trace cache is very large (16K entries), it can contain practically any trace that is required, so trace cache misses should not occur. However, when the branch predictor becomes imperfect, the number of trace cache misses increases significantly.

3.2 On a Trace Cache Miss, the Branch Predictor is Likely Wrong

The strong dependence of the trace cache miss rate on the branch prediction suggests that the branch predictor produces, at times, predictions that are so unlikely that the trace cache cannot provide them. This effect is measured for a 12-bit history MGAg predictor (Table 1). The left half of this table shows the

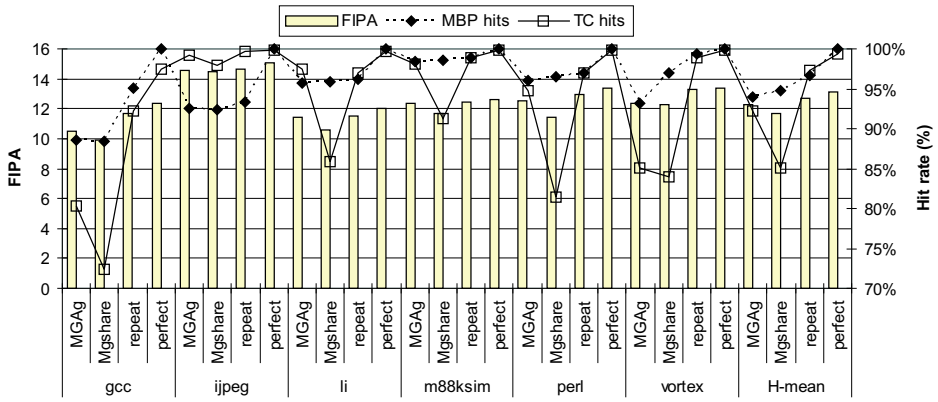


Fig. 2. Fetch throughput (FIPA), branch prediction accuracy and trace cache hit rate in a 16K-entry trace cache.

fraction of trace cache misses that are due to a path misprediction. When the trace cache is sufficiently large, over 80% of trace cache misses is caused by an incorrect branch prediction. In the smaller trace caches, many misses occur due to contention (capacity misses) and are not related to the branch predictor.

Table 1. Fraction of trace cache misses where the path is mispredicted (left) and the fraction of path mispredictions that are caught by monitoring trace cache misses. The trace cache size is varied and the branch history length is 12 bits.

	Fraction of trace cache misses					Fraction of path mispredictions				
TC size	64	256	1024	4096	16384	64	256	1024	4096	16384
gcc	28.3%	34.0%	51.7%	78.5%	86.7%	89.5%	85.8%	78.4%	69.0%	63.6%
ijpeg	7.1%	19.5%	65.7%	84.0%	85.0%	44.1%	27.6%	16.5%	11.4%	10.5%
li	10.8%	35.6%	87.3%	92.5%	92.6%	39.0%	29.6%	24.3%	23.3%	23.2%
m88ksim	9.8%	21.4%	55.9%	81.3%	82.9%	82.5%	63.0%	53.5%	46.1%	46.2%
perl	8.7%	17.6%	51.6%	80.1%	80.4%	84.9%	71.0%	51.5%	45.7%	45.0%
vortex	15.8%	28.0%	50.7%	76.7%	82.3%	98.7%	97.8%	95.0%	91.6%	88.3%
H-mean	12.0%	25.1%	59.3%	82.0%	84.9%	68.9%	55.9%	44.9%	39.0%	37.6%

The right part of Table 1 shows the fraction of mispredicted traces that actually lead to a trace cache miss. In a small trace cache, many misses occur, so there is a large probability of detecting path mispredictions. Up to 70% of the mispredicted paths can be detected. As the trace cache becomes larger, just around 40% of the path mispredictions can be detected.

The branch prediction accuracy also has an important impact on the applicability and utility of trace substitution. We held the trace cache size constant

Table 2. Fraction of trace cache misses where the path is mispredicted (left) and the fraction of path mispredictions that are caught by monitoring trace cache misses. The branch history length is varied and the trace cache is held constant at 256 traces.

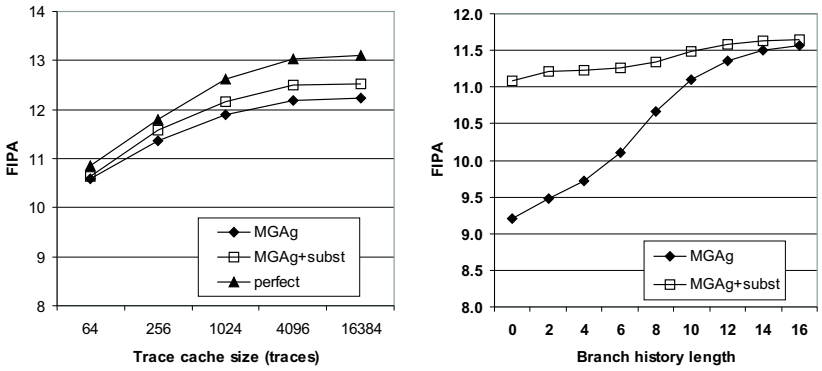
	Fraction of trace cache misses					Fraction of path mispredictions				
hist. length	0	4	8	12	16	0	4	8	12	16
gcc	67.4%	63.6%	57.6%	34.0%	14.8%	88.4%	88.9%	89.4%	85.8%	77.1%
ijpeg	56.0%	46.7%	34.8%	19.5%	12.8%	57.2%	51.4%	41.9%	27.6%	21.6%
li	91.7%	85.2%	63.4%	35.6%	23.2%	67.2%	66.5%	53.4%	29.6%	22.1%
m88ksim	79.4%	80.0%	50.3%	21.4%	10.5%	62.5%	95.3%	84.4%	63.0%	48.7%
perl	77.3%	77.7%	54.3%	17.6%	10.8%	90.3%	91.3%	87.6%	71.0%	58.3%
vortex	57.9%	56.7%	46.0%	28.0%	14.3%	98.8%	98.3%	98.4%	97.8%	96.7%
H-mean	70.5%	66.8%	50.2%	25.1%	13.9%	75.8%	79.9%	72.5%	55.9%	46.5%

at 256 entries and varied the branch history length of the MGAg predictor (Table 2). On average, between 14% and 71% of the trace cache misses indicate path mispredictions. The fraction of mispredicted paths that can be detected by trace cache misses decreases with increasing branch predictor size. This result is somewhat surprising, as one would expect that a higher prediction accuracy combined with a (more or less) constant trace cache miss rate would result in an increase in the fraction of detected mispredictions. However, not all path mispredictions will miss the trace cache. Those that do are "obvious" mispredictions; those that do not are harder to detect. As the branch predictor becomes larger, the amount of "obvious" mispredictions decreases faster than the others.

Trace substitution can only work when the trace on the correct path is present in the trace cache. If it is not, then there is no point in returning another trace that is also on the wrong path. We have found that the correct path trace is present in the trace cache for 35%–87% of the mispredicted paths that miss in the trace cache, for trace caches with 64 to 16K traces and the baseline predictor. When varying the predictor size, the correct trace is available in 6%–38% of the cases. Hence, when a mispredicted path is detected, it will not always be possible to substitute the correct trace. However, even when the correct trace can not be substituted, no harm is done in trying, as both the predicted path and any other wrong-path trace are on the wrong path of execution.

3.3 Trace Substitution

Now that we have discussed the potentials and limitations of trace substitution, let us investigate its impact on performance. We measured the FIPA for a 12-bit history MGAg predictor and trace cache of sizes 64 to 16K traces, all 4-way set-associative (Figure 3(a)). The FIPA is shown for the baseline MGAg predictor, the MGAg predictor with trace substitution and a perfect branch predictor. In the small trace caches, the prominent performance bottleneck is trace cache misses related to the limited capacity of the trace cache. Hence, a perfect branch predictor does not improve performance much. As the trace



(a) Varying trace cache size (b) Varying MGAg size

Fig. 3. The impact of trace substitution on performance (FIPA).

cache becomes larger, the multiple branch predictor becomes a more important bottleneck. Consequently, the gap between the curves for the MGAg and perfect predictors increases with increasing trace cache size.

Trace substitution increases the FIPA obtainable with the MGAg predictor; around 20% to 40% of the gap between the baseline MGAg predictor and the perfect predictor is closed. However, as trace substitution only kicks in on a trace cache miss, the obtainable performance increase strongly depends on the trace cache miss rate. Figure 3 shows that less and less of the performance potential is reached with larger trace caches. The cause of this can be found in Table 1: as the trace cache becomes larger, fewer path mispredictions are detected. The performance increase therefore stagnates as the trace cache becomes larger.

Trace substitution can hide the imperfectness of a poor branch predictor. When varying the history length (and simultaneously the size) of the MGAg branch predictor, the benefits of trace substitution become much clearer. In this experiment, we used a 256-entry trace cache (Figure 3(b)). For the small and inaccurate branch predictors, trace substitution improves performance with 1 to 2 useful instructions fetched per trace cache access. Even for the MGAg with zero bits of global branch history, trace substitution raises the FIPA to the same level as a 10-bit history MGAg without trace substitution.

For the largest of branch predictors, the FIPA is increased over a small amount, e.g., 0.2 instructions per access for the baseline predictor. More importantly, trace substitution enables one to achieve the same performance with a 16 times smaller branch predictor (e.g., 10 vs. 14 bits of history).

Trace substitution depends totally on the co-occurrence of a mispredicted path and a trace cache miss. Hence, the utility decreases for a large trace cache (fewer trace cache misses and fewer possibilities to detect mispredicted paths)

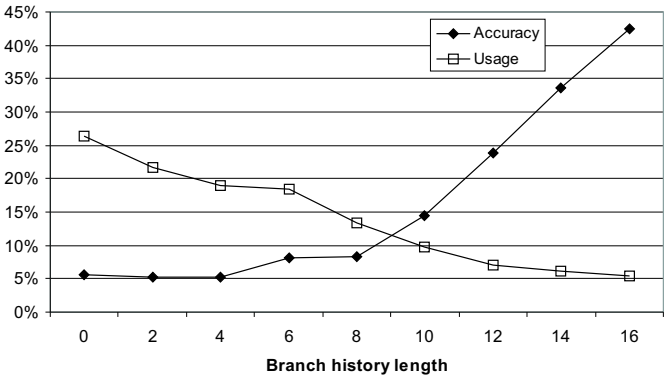


Fig. 4. Accuracy of trace substitution in the baseline trace cache.

and larger branch predictors (fewer path misprediction and thus lack of reason to substitute the trace). Figure 4 shows the usage and accuracy of trace substitution. The usage is defined as the fraction of trace cache accesses for which a trace was substituted. The accuracy is defined as the ratio of the correct substitutions over the total number of substitutions. The usage drops of with larger branch predictors, as more traces are predicted correctly by the branch predictor. On the other hand, when trace substitution is applicable, it is more accurate in larger branch predictors.

Overall, the accuracy of trace substitution is low: the correct trace is returned in less than half of the cases. In the reported experiments, we always returned the most recently used trace having the correct start address. Experiments were made with other choices, e.g., returning a randomly selected trace and a state-based predictor, but these were found to have approximately the same accuracy. Notwithstanding its low accuracy, trace substitution improves the average fetch bandwidth. The reason for having a performance improvement with such low accuracy is that, assuming that the wrong path is predicted on a trace cache miss, it does not matter much whether we return a wrong-path trace from the trace cache, or fetch along another wrong path using the instruction cache. In the end, both paths are wrong.

Another reason for the low accuracy is that the correct-path trace is not always present in the trace cache. Trace substitution is applied as soon as any trace with the correct start address is found but this trace is not always on the correct path of execution. In section 3.2 we estimated that the accuracy is limited to 6%–38%, depending on the branch predictor size, exactly for this reason.

We have also performed the above experiments with the proposed Mgshare predictor. This predictor is by itself less accurate than the MGAg predictor. However, when trace substitution was added, it performed as good as the MGAg with trace substitution. Lack of space prohibits us from treating this in detail.

4 Related Work

Trace caches were introduced to increase the fetch bandwidth for superscalar processors over the single basic block limit [1, 2, 9]. Two approaches for multiple branch prediction can be discerned. The first approach is to adapt a single-branch predictor to multiple branch prediction. This approach was taken in [1, 2, 5, 6, 10] and is also followed in this paper. A second approach is to predict full traces at once and has mainly been investigated in conjunction with block-based trace caches [11–13]. In [13], it is proposed to predict the next trace at completion time. This has the advantage that the predictor can be indexed using non-speculative history information. Trace substitution is not useful when only cached traces are predicted, as in [11, 12].

Selective trace storage does not store *blue* traces (i.e.: traces containing only not-taken branches) in the trace cache, as these can be retrieved equally fast from the instruction cache [3]. Trace substitution can be adapted for selective trace storage by adding additional tags to each set of the trace cache. These tags can only be used for blue traces and indicate that a specific blue path has been followed. The blue trace is still stored in the instruction cache. Trace substitution is not applied when a blue tag hits.

Trace substitution is an idea similar to branch inversion [14], a technique applicable to single-branch predictors. With branch inversion, a confidence predictor is used to detect wrong branch predictions. When a branch prediction is assigned low confidence, the predicted branch direction is inverted. Trace substitution does not require a confidence predictor, but is triggered by trace cache misses. On the other hand, trace substitution is more complicated than branch inversion because in the former a different trace has to be predicted while in the latter, it suffices to simply invert the branch direction.

5 Conclusion

Multiple branch predictors sometimes predict a path of execution that is so unlikely that the trace cache cannot provide it. This paper showed that trace cache misses can be used to detect such mispredictions. We proposed *trace substitution*, a new technique that returns a cached trace when the predicted trace misses in the trace cache. We showed that between 40%–76% of the branch mispredictions can be detected by trace cache misses, depending on the trace cache and branch predictor sizes. Furthermore, between 12%–85% of the trace cache misses are indeed caused by mispredicted branches. The low numbers correspond to small trace caches with many capacity misses.

Trace substitution increases the fetch bandwidth consistently. For small branch predictors, the FIPA is improved with 1 to 2 useful instructions per cycle, obtaining the fetch bandwidth of a 10 to 12-bit history MGAg. For large branch predictors, the FIPA is increased with 0.2 instructions per access or the same FIPA can be obtained with a 16-times smaller branch predictor.

Acknowledgements

The authors thank the reviewers for their helpful comments. Hans Vandierendonck is supported by the Flemish Institute for the Promotion of Scientific-Technological Research in the Industry (IWT).

References

1. E. Rotenberg, S. Bennett, and J. E. Smith, "Trace cache: A low latency approach to high bandwidth instruction fetching," in *Proceedings of the 29th Conference on Microprogramming and Microarchitecture*, pp. 24–35, Dec. 1996.
2. S. Patel, M. Evers, and Y. Patt, "Improving trace cache effectiveness with branch promotion and trace packing," in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pp. 262–271, June 1998.
3. A. Ramírez, J. Larriba-Pey, and M. Valero, "Trace cache redundancy: Red & blue traces," in *Proceedings of the 6th International Symposium on High Performance Computer Architecture*, Jan. 2000.
4. H. Vandierendonck, A. Ramírez, K. De Bosschere, and M. Valero, "A comparative study of redundancy in trace caches," in *EuroPar 2002*, pp. 512–516, Aug. 2002.
5. S. Patel, D. Friendly, and Y. Patt, "Evaluation of design options for the trace cache fetch mechanism," *IEEE Transactions on Computers*, vol. 48, pp. 193–204, Feb. 1999.
6. E. Rotenberg, S. Bennet, and J. Smith, "Trace cache: A low latency approach to high bandwidth instruction fetching," Tech. Rep. 1310, University of Wisconsin - Madison, Apr. 1996.
7. T.-Y. Yeh and Y. N. Patt, "Alternative implementations of two-level adaptive branch prediction," in *Proceedings of the 19th Annual International Conference on Computer Architecture*, pp. 124–134, May 1992.
8. S. McFarling, "Combining branch predictors," Tech. Rep. WRL-TN36, Western Research Laboratory, 250 University Avenue, Palo Alto, California 94301, USA, June 1993.
9. A. Peleg and U. Weiser, "Dynamic flow instruction cache memory organized around trace segments independent of virtual address line," *U.S. Patent Number 5.381.533*, Jan. 1995.
10. T.-Y. Yeh, D. Marr, and Y. Patt, "Increasing the instruction fetch rate via multiple branch prediction and a branch access cache," in *ICS'93. Proceedings of the 1993 ACM International Conference on Supercomputing*, pp. 67–76, July 1993.
11. B. Black, B. Rychlik, and J. Shen, "The block-based trace cache," in *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pp. 196–207, May 1999.
12. Q. Jacobson, E. Rotenberg, and J. Smith, "Path-based next trace prediction," in *Proceedings of the 30th Conference on Microprogramming and Microarchitecture*, pp. 14–23, Dec. 1997.
13. R. Rakvic, B. Black, and J. Shen, "Completion time multiple branch prediction for enhancing trace cache performance," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 47–58, May 2000.
14. S. Manne, A. Klauser, and D. Grunwald, "Branch prediction using selective branch inversion," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pp. 48–56, Oct. 1999.