Control Flow Prediction with Tree-Like Subgraphs for Superscalar Processors

Simonjit Dutta
Texas Instruments
Semiconductor Group
P.O. Box 655303, M.S. 8316
Dallas, TX 75265, USA
dutta@daldd.sc.ti.com

Manoj Franklin
Dept. of Electrical and Computer Engineering
Clemson University
221-C Riggs Hall
Clemson, SC 29634-0915, USA
mfrankl@blessing.eng.clemson.edu

Abstract

In order to fetch a large number of instructions per cycle, wide-issue superscalar processors have to predict the outcome of multiple branches in a cycle, and fetch instruction blocks from multiple targets. This paper investigates a control flow prediction scheme that predicts the outcome of multiple branches by performing a single prediction. Instead of predicting the outcome of each individual conditional branch, this scheme considers a tree-like subgraph of the control flow graph of the executed program as a single prediction unit, and predicts the target of a subgraph at a time, thereby allowing the superscalar fetch mechanism to go past multiple branches per cycle. This approach is evaluated using the MIPS architecture, for a 12-way superscalar processor, and an improvement in effective fetch size of more than 50%, over an identical processor that uses branch prediction is observed for the SPEC integer benchmarks. No appreciable difference in the prediction accuracy was observed, although the control flow prediction scheme predicted one out of four outcomes.

1 Introduction

To sustain high issue rates, wide-issue superscalar processors must fetch a large number of instructions per cycle. Fetching multiple instructions in a cycle is not difficult so long as the fetched instructions are from a straight line piece of code. However, conditional branches are a major problem, because they are frequent (roughly one in every 5-6 instructions on the average, for non-numeric programs) and can potentially alter the flow of control. To overcome this problem, either (i) the number of branches encountered by the hardware has to be reduced (through techniques such as *conditional execution* [3]) or (ii) the fetch mechanism should fetch multiple basic blocks each cycle despite taken branches in the fetch stream [4] [6]. This paper focuses on the second approach.

The approach proposed in [4], called *control flow prediction*, is to predict the target of a subgraph of the control flow graph of the executed program. By predicting a target, the control flow predictor specifies the next subgraph to be (speculatively) executed, and thereby goes past one subgraph a cycle. A subgraph is allowed to have arbitrary structure, length, and number of targets. Such a control flow prediction scheme, with unrestricted subgraph structures, is more apt for execution models that pursue multiple flows of control such as the *multiscalar processor* [4].

For application in superscalar processors, we need to restrict the subgraph structure, and also identify a path within the subgraph to fetch instructions from. A suitable restricted subgraph structure is a tree-like structure (c.f. Figure 1), similar to the VLIW tree-like instructions [2]. Owing to their well-defined structure, tree-like subgraphs allow the easy selection of a dynamic path through them. The approach considered in [6] to select a path through the tree-like subgraph is to perform multiple branch predictions in a cycle, without determining the addresses of these branches. Based on the prediction values, a *Branch Address Cache* provides the intermediate fetch addresses, which are then used to fetch instructions. Because the intermediate branch addresses are not known when the predictions are made, this scheme uses the condensed history of all branches to make its decisions.

This paper proposes a prediction scheme that selects a dynamic path through the tree-like subgraph by means of a single prediction. The advantage of this approach is that the prediction mechanism need not be any more complex than an ordinary branch predictor. Furthermore, it is not restricted to storing the condensed history of all branches, and this gives better prediction accuracies. The rest of this paper is organized as follows. Section 2 provides the details of our control flow prediction scheme with tree-like subgraphs. Section 3 calculates the hardware cost of this scheme, discusses ways of reducing the cost, and compares the cost with that of existing schemes. Section 4 presents an experimental evaluation of the proposed scheme using the MIPS architecture. Section 5 presents the conclusions.

2 Tree-Level Prediction

2.1 General Operation

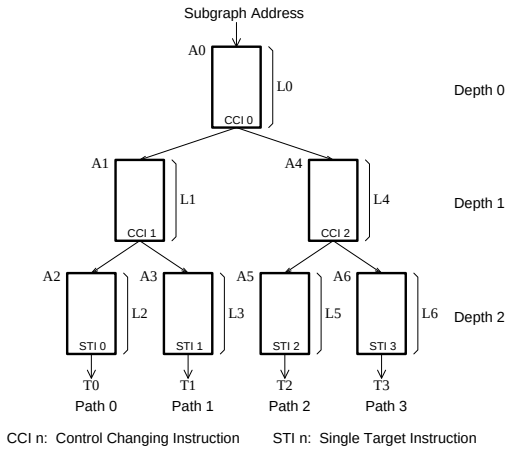

Figure 1: Generic Structure of a Tree-like Subgraph.

Figure 1 shows the generic structure of a tree-like subgraph of depth 3. Each node of the subgraph is a straightline piece of code. The nodes at depth 0 and 1 can be terminated by any control-changing instruction (CCI), such as conditional branches, unconditional branches, and procedure calls. The nodes at depth 2 can be terminated only by single-target instructions (STI), such as unconditional branches, procedure calls, and non-control-changing instructions. Such a definition allows up to 3 conditional branches, 7 nodes, and 4 paths to be encompassed in a subgraph. The subgraph nodes start at addresses A0 ... A6, and have lengths L0 ... L6. The address of the subgraph is the address of the first instruction in the subgraph, A0. The subgraph has 4 targets {T0, T1, T2, T3}, one corresponding to each path. The 4 paths are encoded {0, 1, 2, 3}. Each path has 3 attributes: {node addresses, node lengths, target}. For instance, the attributes of path 0 are {{A0, A1, A2}, {L0, L1, L2}, T0}. The first two attributes aid in fetching the path from the i-cache, and the last attribute gives the address of the next subgraph.

Figure 2 gives a block diagram showing the relationship between the fetch unit, the subgraph predictor, the i-cache, the decode unit, the hardware window, and the functional units in a superscalar processor. The fetch unit sends the current subgraph address (CSA) to the subgraph predictor, which predicts a path through this subgraph and sends the attributes of the selected path to the fetch unit. The fetch unit then fetches this path from the multi-ported i-cache, and forwards the fetched instructions to the decoder. If the fetch unit encounters a particular path of a subgraph for the first time, it also tells the decoder to send that path's attributes to the subgraph predictor for storage.

The decoded instructions are sent to the hardware window structure for performing dynamic scheduling. Multiple functional units, including branch units, are provided to execute multiple instructions per cycle. When a conditional branch is executed by a branch unit, a check is made to see if the actual outcome is in line with the earlier path prediction; if there is a discrepancy, the hardware window discards the instructions following the point of discrepancy, and sends the misprediction information to the fetch unit, which then starts fetching from the correct target.

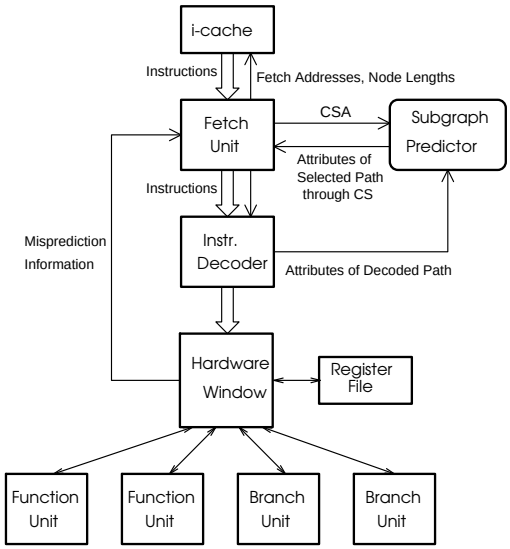

Figure 2: Block Diagram of a Superscalar Processor with Tree-like Subgraph-Level Prediction.

2.2 Size of a Subgraph

We shall now pause briefly to examine some restrictions that need to be imposed on the size of our subgraphs, in particular on the following 2 attributes: (i) the number of control flow outcomes (or paths) a subgraph can have, and (ii) the number of instructions in each path.

The first attribute is decided by the number of branches encompassed in a subgraph path. If a subgraph has only two outcomes, then only one branch is effectively predicted per cycle, and this severely limits the average number of instructions fetched per cycle. On the other hand, if a subgraph has a large number of possible outcomes, then the prediction mechanism has to choose one outcome from many, potentially resulting in lower prediction accuracies. Each time a misprediction is detected, costly repair actions are involved, offsetting any benefit due to fetching a large number of instructions at a time. As a trade off, in this paper we allow up to 4 outcomes per subgraph so as to encompass up to two conditional branches in any single path through a subgraph. The subgraph is extended beyond the second conditional branch by one more level (note that this last level cannot have any conditional branches) to further increase the subgraph size

without increasing the number of paths in it.

Now consider the second attribute, namely the number of instructions in each path of the subgraph. Ideally, the fetch mechanism should be able to fetch all instructions from a selected path in one cycle. This will allow one subgraph prediction to be carried out every cycle. Thus, the length of any path through a subgraph is limited to f (by trimming the last node in that path), where f is the maximum fetch size of the superscalar fetch mechanism.

2.3 Prediction Mechanism

Let us now return to our discussion of subgraph-level prediction. The subgraph path predictions are made with the help of a 2-*level subgraph predictor*[1], similar in spirit to the 2-level block predictor described in [1] for carrying out sequential block-level prediction. Figure 3 shows the block diagram of our 2-level subgraph predictor. It consists of two main structures—the *Subgraph History Table (SHT)* and the *Pattern History Table (PHT)*. The SHT is a cache-like structure that stores information relevant to the subgraphs that have been encountered in the recent past. Each SHT entry has 3 fields—Tag, Path Attributes, and Subgraph History Pattern. The Tag field is for uniquely identifying the subgraph currently mapped to that entry. The Path Attributes field stores the attributes of that subgraph's paths, and the Subgraph History Pattern field stores the last p outcomes ($p = 6$ in the figure) of the subgraph as a $2p$-bit pattern. Because there are 4 possible outcomes for a subgraph, 2 bits are required to store each outcome. For each possible $2p$-bit pattern, a condensed history of the previous outcomes corresponding to the pattern is recorded in the 2^{2p}-entry PHT by means of 4 independent up/down counters $\{C_0, C_1, C_2, C_3\}$.

When the fetch unit requests a prediction for subgraph S, the appropriate SHT entry is selected, and its tag value is checked to see if S is currently mapped to it. If so, the entry's subgraph history pattern is used to select the appropriate PHT entry. The 4 count values in the PHT entry are supplied to a Path Selector, which selects the path corresponding to the maximum count value. The selected SHT entry also contains the attributes of all paths contained in S. Depending on the prediction value selected, the attributes corresponding to the selected path are selected using a MUX, and sent to the fetch unit.

The predictor also performs a speculative update of S's history. The selected PHT entry's counter values are updated in the following manner. The count value corresponding to the predicted outcome is incremented by 3 (or less if the counter saturates), and all other count values are decremented by 1 (if their count values are not already 0). Finally, the subgraph history pattern of the selected SHT entry is shifted left by 2 bit positions, and the latest prediction is entered in the 2 rightmost bit positions. Notice that these updates to the SHT and PHT entries, done immediately after making a prediction, are done in a speculative manner so as to provide up-to-date history for making future predictions. If a prediction is later found to be incorrect, the speculative updates done to the SHT and PHT entries corresponding to the subgraphs discarded by the recovery mechanism must be setback.

[1] It is not mandatory to use the 2-level approach; we describe an implementation using the 2-level approach, as it gives high prediction accuracies.

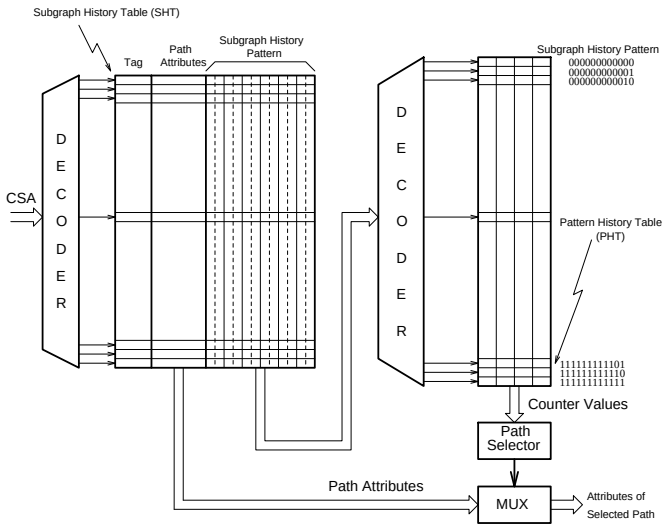

Figure 3: 2-Level Subgraph Predictor.

When the fetch unit requests a prediction for a subgraph S whose entry is not present in the SHT, the predictor returns a prediction of 0, and a temporary set of attributes $\{\{S$'s address, 0, 0$\}$, $\{f$, 0, 0$\}$, S's address + $f\}$ (where f is the maximum fetch size) to the fetch unit. When this path is fetched, the fetch unit unit tells the decoder to send the correct attributes of path 0 to the SHT for storage. The attributes of the remaining paths in S are collected by the decoder when control flows through them for the first time.

When a path prediction is made for a subgraph, the outcomes of the conditional branches contained in the selected path also get indirectly predicted. When the predicted outcome of a conditional branch is later found to be incorrect, recovery actions are initiated. Part of the recovery process includes finding a new prediction for the subgraph that contains the mispredicted branch. If the mispredicted branch is the second branch in its path in the subgraph, then finding a new prediction for the subgraph is straightforward: if the previous prediction is given by the 2-bit binary value $b_1 b_0$, the new prediction value is given by $b_1 \bar{b_0}$. On the other hand, if the mispredicted branch is the first branch in the path, then there is a choice involved in selecting the new prediction value. If the branch is now taken, then the fetch unit requests the predictor to select a value from $\{2, 3\}$, and if the branch is now fall-through, then the fetch unit requests for a value from $\{0, 1\}$.

3 Hardware Cost

3.1 2-Level Subgraph Predictor

SHT: Consider an SHT entry. Its Tag field requires at most W bits, where W is the address width. The Path Attributes field needs to store 4 target addresses (each requiring W bits), 6 intermediate fetch addresses (each requiring W bits), and 7 subgraph node lengths (each requiring 4 bits, allowing each node to have lengths up to 15). Lastly, the Subgraph History Pattern field requires $2p$ bits. Thus, the maximum number of bits required to implement an N-entry SHT is $N \times (11W + 28 + 2p)$ (which is about 24.5Kbytes for $N = 512$, $W = 32$, and $p = 6$).

PHT: Each PHT entry stores 4 count values, each requiring, say c bits. The total number of bits for implementing the 2^{2p}-entry PHT is therefore given by $4c \times 2^{2p}$ (which works out to be 8Kbytes for $c = 4$ and $p = 6$).

3.2 Reducing the Hardware Cost

The cost analysis given above shows that 2-level subgraph prediction, as described so far, has high hardware cost. Fortunately, there are ways to reduce the hardware cost.

SHT: The reason for the significant hardware cost for the SHT is because of the storage required for the Path Attributes field, which in fact contains a lot of redundant information. Referring to Figure 1, for path 0, it is sufficient to store the subgraph address A0, the combined length L0+L1+L2, and the target T0. Similarly, for path 1, it is sufficient to store in addition, address A3, the lengths L0+L1 and L3, and the target T1. By storing information in this manner, the SHT cost can be reduced to $N \times (8W + 28 + 2p)$ bits (which is about 18Kbytes for $N = 512$, $W = 32$, and $p = 6$). Notice that most processors have restrictions on the instruction address space, effectively reducing the value of W by even 25% or more, resulting in an additional 25% savings in hardware cost. Notice also that N can be reduced without reducing SHT hit rate by using set-associative mapping for the SHT.

PHT: The reason for the significant increase in hardware cost for the PHT is because of the factor 2^{2p} in its cost formulae. Notice that, of the 2 bits used to store a previous outcome in the SHT, the first bit encodes the outcome of the first branch in the subgraph, and the second bit encodes the outcome of the second branch in the taken path in the subgraph. By treating these 2 branch outcomes independently, we can significantly reduce the storage requirements of the PHT, as follows. When a prediction is requested for subgraph S, the SHT supplies a $2p$-bit history pattern of previous outcomes of S. Split this $2p$-bit pattern into two p-bit patterns, as shown in Figure 4. Use one of the patterns to consult a PHT to predict the next outcome of the first branch in S, and use the other pattern to consult a different PHT to predict the next outcome of the second branch (in the predicted path) in S. Combining the 2 single-bit prediction values gives the next 2-bit prediction for S. With this scheme, the number of bits required for the PHTs becomes $2 \times c2^p$ bits (which is only 64 bytes for $c = 4$ and $p = 6$).

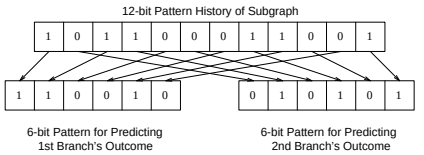

Figure 4: Splitting a $2p$-bit Pattern into Two.

3.3 Comparison to Other Schemes

Let us compare the hardware cost of the split-pattern 2-level subgraph predictor to that of (i) 2-level branch predictor (PAg scheme) [5], and (ii) multiple branch predictions per cycle using 2-level branch predictor [6].

2-Level Branch Predictor

The BHT in a 2-level branch predictor has N entries, each having a Tag field (occupying at most W bits), a W-bit Target field, and a p-bit Branch History Pattern field. The number of bits required for the BHT is $N \times (2W + p)$ (which is about 4.4Kbytes for $N = 512$, $W = 32$, and $p = 6$). This is roughly 25% of that required for the SHT.

The number of bits required for the PHT is $c \times 2^p$ bits, which works out to be 32 bytes (for $c = 4$ and $p = 6$), which is half of that required for the PHTs in the split-pattern subgraph predictor. Overall, *the hardware cost for our split-pattern subgraph prediction scheme is about 4 times that for 2-level branch prediction scheme, if both use the same number of entries at the first level table.* It is important to note, however, that the number of entries in the SHT can be less than that in the BHT, because the hardware encounters fewer subgraphs than branches.

Multiple Branch Predictor

Because of the need to make predictions without knowing the intermediate addresses, for practical purposes, the multiple branch predictor is restricted to using a p-bit global history register in place of the BHT [6]. It instead uses a Branch Address Cache (BAC) for storing the intermediate fetch addresses, node lengths, and targets of subgraphs. An N-entry BAC can be implemented with $N \times (8W + 28)$ bits using the technique given in Section 3.2. This hardware cost is very close to that of the SHT.

A major implication of using a global history register is that a much larger pattern size (typically 14 or more) has to be used to get reasonable prediction accuracies [5] [6]. If $p = 14$, then the number of bits required for the PHT works out to be 8Kbytes (for $c = 4$), which is significantly higher than that for the PHTs in the split-pattern subgraph predictor. Thus, *the hardware cost for our split-pattern subgraph prediction scheme is lower than that for the multiple branch prediction scheme.*

4 Experimental Evaluation

This section presents the results of an empirical study of the performance of subgraph-level prediction. For comparison purposes, we also evaluate the 2-level branch prediction scheme [5], and the multiple branch prediction scheme using 2-level branch predictor [6].

4.1 Experimental Setup

The performance studies are conducted using a simulator that accepts programs compiled for a MIPS-based DECstation and simulates their execution, keeping track of relevant information on a cycle-by-cycle basis.

For benchmarks, we use 5 of the SPEC '92 integer programs. They are compiled using the MIPS C compiler with the optimization flags distributed in the SPEC benchmark makefiles. Notice that the code was compiled for a single-issue processor. The results are subjected to change, if a different compiler or static schedule is used. The benchmarks are simulated up to 100 million instructions, or up to completion (if it completed earlier).

To get good insight on the fetch mechanism's performance, we use the following indirect metrics: (i) *effective fetch size (EFS)*, (ii) *subgraph prediction accuracy (SPA)*, and (iii) *branch prediction accuracy (BPA)*. The first metric (EFS) indicates the average number of *useful* instructions fetched in parallel by the fetch unit. Notice that useless instructions that are discarded due to incorrect speculative execution are not counted while calculating this metric. Notice also that, if a subgraph path had to be fetched in 2 attempts because of a misprediction, then it is counted as 2 fetch attempts. Thus, EFS takes into account the effect of the inaccuracies in subgraph prediction.

The second metric (SPA) indicates the fraction of times a subgraph prediction gives the correct path through a subgraph. Although EFS captures the effect of subgraph prediction inaccuracies, SPA throws more light on why the performance of the predictor is better or worse. We are also interested in knowing if making a 4-way subgraph prediction instead of a 2-way branch prediction results in a poor prediction accuracy. The third metric (BPA) indicates the fraction of times a branch (indirectly) gets correctly predicted when subgraph prediction is performed. BPA is highly dependent on SPA, but is helpful in comparing with the BPAs obtained with other prediction techniques.

The default parameters used in the study are:

- *Hardware Window:* The hardware window size is fixed at 64. When a misprediction is detected at cycle t, the fetch unit initiates fetching from the correct target in cycle $t+1$, and the execution unit recovers completely by the end of cycle $t+2$.
- *Instruction Cache* The instruction cache is 64 Kbytes, 4-way set-associative, and has an access latency of 1 cycle. A cache block can hold 64 MIPS instructions.
- *Maximum Fetch Size (f):* The default value is 12.
- *Branch Predictor:* This scheme is simulated with a *PAg* 2-level branch predictor [5], with a pattern size (p) of 6. The BHT has 2048 entries, and is direct mapped. The PHT entries consist of 3-bit up/down saturating counters. The fetch unit fetches a sequential block of f instructions in each attempt; if the first conditional branch within this block is predicted to be fall-through, the subsequent instructions up to the next conditional branch in the block are also entered into the hardware window.
- *Multiple Branch Predictor:* This scheme is simulated with a *GAg* 2-level branch predictor. The Branch Address Cache has 1024 entries, and is direct mapped. The PHT entries consist of 3-bit up/down saturating counters. Two pattern sizes ($p = 7$ and $p = 14$) were used. Notice that the scheme with pattern size 7 has roughly the same hardware cost as the default subgraph predictor scheme.
- *Subgraph Predictor:* This scheme is simulated with a *PAg* 2-level subgraph predictor. The SHT is also direct mapped, and the default number of SHT entries is 1024. Each PHT entry has four 4-bit up/down saturating counters. The default pattern size is 6.

4.2 Experimental Results

4.2.1 Effective Fetch Size

Table 1 shows the EFS obtained with the default branch prediction scheme, the multiple branch prediction scheme [6], and the split-pattern subgraph prediction scheme. The second column presents the EFS obtained with the branch prediction scheme. The third and fourth columns present the EFS obtained with multiple branch prediction per cycle for $p = 7$ and $p = 14$, respectively. The fifth and sixth columns present the EFS obtained with split-pattern subgraph prediction (with $p = 6$), and the last column presents the percentage improvement in EFS when using subgraph prediction, compared to the branch prediction scheme.

The important observation to note from Table 1 is that the EFS is significantly higher for the subgraph-level prediction scheme compared to that for the branch prediction scheme (varying from 41.8% (for gcc) to 130.8% (for eqntott)). The second observation to note is that the EFS for multiple branch predictor with $p = 7$ (which has roughly the same hardware cost as the default split-pattern subgraph predictor) is also low. This is because the multiple branch predictor uses a global history register, which results in poor prediction accuracy. When p is increased to 14, its performance improves, and it performs only slightly worse than the subgraph prediction scheme (with $p = 6$).

4.2.2 Prediction Accuracy

The prediction accuracies obtained are presented in Table 2. The second column in the table gives the BPA obtained with the branch prediction scheme. The next group of 4 columns give the SPA and BPA results for multiple branch predictor for pattern sizes (p) of 7 and 14. The last group of 4 columns give the SPA and BPA results obtained with the two tree-like subgraph prediction schemes.

Program	*Branch Pred.*	*Mult Br Pred.*		*Subgraph Pred.*	
		$p = 7$	$p = 14$	*EFS*	*Incr. over Br. Pred.*
compress	5.88	6.58	7.52	8.47	44.04%
eqntott	3.54	7.23	8.10	8.17	130.79%
espresso	5.35	7.15	8.02	8.48	58.50%
gcc (cc1)	5.53	6.44	7.32	7.84	41.77%
xlisp	5.44	6.81	8.14	8.16	50.00%

Table 1: Effective Fetch Size for $f = 12$

First, let us look at the results for the multiple branch predictor. The prediction accuracy results obtained with $p = 7$ are somewhat low. For the multiple branch predictor to perform reasonably well, the pattern size has to be much bigger (14 or more), resulting in high hardware overheads. Next, let us look at the results for the subgraph predictor. The BPAs are fairly close to that obtained with the branch prediction scheme (except for espresso). This result is very encouraging, because it shows that predicting one out of 4 outcomes has provided more or less the same accuracy as predicting one out of 2 outcomes. An important observation to make is that the prediction accuracies with the split-pattern scheme are close to that without the split-pattern scheme. This is also very encouraging because the split-pattern scheme has low hardware overhead.

5 Summary and Conclusions

We have proposed a control flow prediction mechanism using tree-like subgraphs for application in wide-issue superscalar processors. Instead of predicting the outcome of each branch, this scheme predicts a path through a subgraph of the control flow graph, thereby enabling the fetch unit to bring in a large number of instructions per cycle despite taken branches in the fetch stream.

We calculated the hardware cost of carrying out subgraph-level prediction, and also proposed ways to cut down the hardware cost. The cost was found to be less than alternate schemes to predict the outcome of multiple branches per cycle. We also conducted a thorough simulation study to verify the potential of this prediction strategy. The results of our experiments with the MIPS architecture help us to conclude that subgraph-level prediction schemes are very profitable for use in wide-issue superscalar processors.

Subgraph-level prediction has several advantages. First, it predicts the outcome of multiple branches per cycle. This is essential to improve the performance of wide-issue superscalar processors for non-numerical programs. Second, only a single prediction is made per cycle. Thus, the prediction mechanism need not be any more complex than an ordinary branch prediction scheme.

Acknowledgements

This work was financially supported by the NSF Research Initiation Award CCR 9410706 and by the Dept of Electrical and Computer Engineering at Clemson University. We thank the reviewers for their many helpful comments.

References

[1] S. Dutta and M. Franklin, "Block-Level Prediction for Wide-Issue Superscalar Processors," *Proc. 1st International Conference on Algorithms and Architectures for Parallel Processing*, pp. 143-152, April 1995.

[2] K. Ebcioglu, "Some Design Ideas for a VLIW Architecture for Sequential Natured Software," in M. Cosnard, *et al.*, (eds.), *Parallel Processing (Proc. IFIP WG 10.3 Working Conference on Parallel Processing, Pisa, Italy)*, North Holland, 1988, pp. 3-21.

[3] S. A. Mahlke, *et al*, "Characterizing the Impact of Predicated Execution on Branch Prediction," *Proc. 27th Annual International Symposium on Microarchitecture (MICRO-27)*, pp. 217-227, 1994.

[4] D. N. Pnevmatikatos, M. Franklin, and G. S. Sohi, "Control Flow Prediction for Dynamic ILP Processors," *Proc. 26th Annual International Symposium on Microarchitecture (MICRO-26)*, pp. 153-163, 1993.

[5] T-Y Yeh and Y. N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," *Proc. 19th Annual International Symposium on Computer Architecture*, pp. 124-134, 1992.

[6] T-Y Yeh, D. T. Marr, and Y. N. Patt, "Increasing Instruction Fetch Rate via Multiple Branch Predictions and a Branch Address Cache," *Proc. 7th ACM International Conference on Supercomputing*, pp. 67-76, July 1993.

Program	*Branch Predictor*	*Multiple Branch Predictor*				*Tree-like Subgraph Predictor*			
		with $p = 7$		*with $p = 14$*		*without split pattern*		*with split pattern*	
	BPA	*SPA*	*BPA*	*SPA*	*BPA*	*SPA*	*BPA*	*SPA*	*BPA*
compress	89.32%	81.20%	83.80%	82.66%	86.52%	85.94%	89.14%	85.62%	88.96%
eqntott	93.96%	81.20%	88.45%	92.85%	95.11%	93.04%	96.13%	93.02%	95.99%
espresso	96.61%	81.65%	88.69%	84.69%	90.15%	88.38%	92.18%	87.39%	91.81%
gcc (cc1)	88.17%	74.36%	80.57%	83.71%	85.63%	86.61%	89.91%	85.09%	88.30%
xlisp	94.79%	83.49%	84.73%	90.18%	90.65%	91.43%	91.90%	90.52%	90.90%

Table 2: SPA and BPA for $f = 12$