Differential FCM: Increasing Value Prediction Accuracy by Improving Table Usage Efficiency

Bart Goeman, Hans Vandierendonck and Koen De Bosschere
Dept. of Electronics and Information Systems, Ghent University
Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
E-mail: {bgoeman,hvdieren,kdb}@elis.rug.ac.be

Abstract

Value prediction is a relatively new technique to increase the Instruction Level Parallelism (ILP) in future microprocessors. An important problem when designing a value predictor is efficiency: an accurate predictor requires huge prediction tables. This is especially the case for the finite context method *(FCM) predictor, the most accurate one.*

In this paper, we show that the prediction accuracy of the FCM can be greatly improved by making the FCM predict strides instead of values. This new predictor is called the differential finite context method *(DFCM) predictor. The DFCM predictor outperforms a similar FCM predictor by as much as 33%, depending on the prediction table size. If we take the additional storage into account, the difference is still 15% for realistic predictor sizes.*

We use several metrics to show that the key to this success is reduced aliasing in the level-2 table. We also show that the DFCM is superior to hybrid predictors based on FCM and stride predictors, since its prediction accuracy is higher than that of a hybrid one using a perfect meta-predictor.

1. Introduction

Current microprocessor architectures use increasingly aggressive techniques to raise the average number of instructions executed per cycle (IPC).

The upper bound on achievable IPC is generally imposed by true register dependencies: instructions that need input from other instructions have to wait until the latter are finished. Value prediction is a technique capable of pushing this upper bound by predicting the outcome of an instruction and executing the dependent instructions earlier using the predicted value. Several studies have shown that register values are indeed predictable [10, 11, 14, 15, 16, 17], others study the achievable speedup [2, 6, 8, 13].

A major problem when designing a value predictor is efficiency: a high prediction accuracy requires large prediction tables. This is especially true for the *finite context method* (FCM) predictor. The FCM is the most accurate of all simple (i.e. non-hybrid) predictors. It is capable of predicting both stride patterns and non-stride patterns fairly accurately. We will show, however, that the FCM is inefficient in predicting stride patterns, since a lot of unnecessary interference between stride and non-stride patterns occurs in the level-2 prediction table. We also show in this paper that this interference can be removed by reducing the number of entries which are occupied by stride patterns.

Stride patterns are characterized by the property that the differences between successive values in a pattern are identical. Put differently, the pattern of differences between values is a series of identical values. Therefore, when the history of the FCM is comprised of *differences* between values, rather than the values themselves, stride patterns map to only one entry in the level-2 table, while irregular repeating patterns remain as predictable as before. The *differential finite context method* (DFCM) is a new FCM-based predictor, that uses this technique. The DFCM achieves a prediction accuracy which can be 33% higher than the prediction accuracy of the FCM.

This paper starts with an overview of the best known value predictors. Section 2 also discusses the behavior of the FCM predictor for stride patterns. The differential finite context method is introduced in section 3. In section 4, the prediction accuracy of the DFCM is evaluated and it is shown that the improvement of the DFCM over the FCM is caused by reduced interference in the level-2 prediction table. The DFCM is also compared to hybrid predictors and the size of the stored strides is varied. Section 5 discusses related work and section 6 summarizes the main conclusions.

2. Value predictors

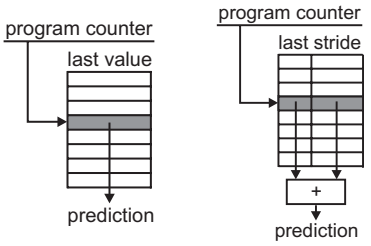
2.1. Last value predictor

The last value predictor was introduced by Lipasti [10, 11]. This predictor assumes the next value that will be produced by an instruction is the same as the previous one. The

Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA'01)
0-7695-1019-1/01 $10.00 © 2001 IEEE

last value predictor works best when the data contains constant patterns.

The last value predictor is instruction based. This means that the program counter of the instruction is used as an index in the prediction table (Figure 1(a)). The prediction table stores the last value which has been produced by the instructions mapping to the corresponding entry.



(a) Last Value Predictor (b) Stride Predictor

Figure 1. Last Value and Stride Predictor

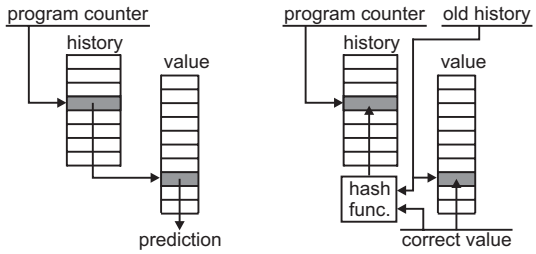
2.2. Stride predictors

The stride predictor (introduced for value prediction by Gabbay [5]) is more complex than the last value predictor. The model underlying the stride predictor postulates that the value pattern produced by an instruction accords to a stride pattern. In a stride pattern, the difference between two consecutive values is always the same constant.

Several flavors of the stride predictor have been proposed. The simplest stride predictor bases its prediction on two values: the last value and a stride (Figure 1(b). The stride is the difference between the last value and the preceding value. The next value is computed by adding the stride to the last value.

A more complex stride predictor is the two-delta method [4]. This stride predictor keeps track of a last value and two stride values (s1 and s2). The two-delta method uses the stride s1 and the last value to compute the predicted value. When the predictor table is updated, the new stride is computed by subtracting the old *last value* from the newly produced result. When this stride equals s2, it is stored in the s1 field. This way, the stride s1 is only updated when the same stride occurs twice in a row. The new stride is always stored in s2. As a consequence, a reset of a loop control variable will only introduce one misprediction.

Throughout this paper, we use a comparable method: a saturating counter is used to measure the predictability, and the stride is only changed if this counter is low. This way, only one stride is needed. The saturating counter is usually already present to track the confidence, so no additional storage is needed.



(a) Predicting a value (b) Updating the predictor

Figure 2. The finite context method.

2.3. Finite Context Method

The finite context method (FCM) is a context-based predictor [15]. In contrast to the stride predictor, no particular relation between the values is assumed. A context-based predictor uses the history of recent values, called the *context*, to determine the next value. The FCM implements this strategy by means of two levels of prediction tables (Figure 2). The first level table stores the context or the recent history of an instruction. The second level table stores for each possible context the value which is most likely to follow it. When a prediction is made, the program counter is used to find the history of recent values corresponding to the instruction in the first level table (Figure 2(a)). This history is then used as an index in the second level table, where the next value is found.

The length of the history, expressed as the number of values stored in it, is referred to as the *order* of the FCM. An FCM predictor is also able to predict constant and stride patterns, although the learning period is longer [15].

To obtain good use of the level-2 table, each history should map to a different entry of the level-2 table. This can be accomplished by using a hashing function. Sazeides gives an overview of different types of hashing functions [14]. The hashing functions typically XOR the different values together. Also, the new history can be computed incrementally, using the old history and the new value to add to it. Therefore, only the hashed history needs to be stored in the level-1 table, and not the complete history.

When the outcome of the prediction is known, the prediction tables have to be updated (Figure 2(b)). The correct value is written in the level-2 table, in the entry where the predicted value was read. Also, the history corresponding to the executed instruction needs to be adjusted. The hash function constructs the new history from the old history and the new value.

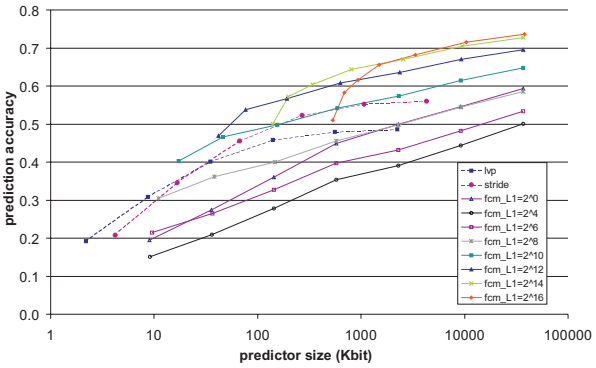


Figure 3. LV, Stride and FCM predictors: accuracy vs. size

Context	Value	$\frac{Accesses}{Iteration}$
0 1 2	3	1
1 2 3	4	1
2 3 4	5	1
3 4 5	6	1
4 5 6	0	1
5 6 0	1	1
6 0 1	2	1

Figure 4. Example stride pattern stored in FCM level-2 table.

2.4. Discussion

Figure 3 gives an overview of the accuracy of the various predictors and the required storage in Kbit (the benchmark programs are described in section 4).For FCM, Each curve corresponds to a fixed number of entries in the level-1 table. The different dots on each curve correspond to different sizes of the level-2 table: 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18} and 2^{20} entries. For the LVP and the stride predictor, the table size ranges from 2^6 up to 2^{16} entries. It is clear that FCM is the most accurate method, but requires *huge* prediction tables. The prediction accuracy starts to saturate for a first level table with 2^{14} entries, but increasing the second level table is beneficial, even when going from 2^{18} to 2^{20} entries. The second level table is obviously not used in the most efficient way. This becomes clear when we look at the behavior of stride patterns.

The FCM predictor as described above is able to predict stride patterns, although the FCM predictor treats stride patterns as if they were context based. Consider the pattern 0 1 2 3 4 5 6 which is continuously repeated. A third order FCM predictor breaks this pattern in histories of length 3 and scatters it over many entries in the level-2 table (Figure 4). For simplicity, we assume the hashing function concatenates the values in the history.

Since each value only occurs just once in each repetition of the pattern, the FCM predictor allocates as many entries in the level-2 table as there are different values in the pattern. As a consequence, a stride pattern with length n will be stored over n different entries in the level-2 table. Should the level-2 table be smaller than n entries, the stride pattern will destructively interfere with its own and many other patterns occurring in the program.

What happens if there are different stride patterns? If the stride is different, or if the range is non-overlapping, they will all require their own set of entries in the level-2 table. It is clear the level-2 table will be crowded with stride patterns, even if they are short, leaving little space for the repeating non-stride patterns, the real aim of the two-level prediction mechanism.

```
void norm( double matrix[200][100])  {
  int i;
  for (i=0; i<200; i++) {
      double max=matrix[i][99];
      int j;
      for (j=0; j<99; j++)
        if (fabs(matrix[i][j]) > max)
          max=matrix[i][j];
      if (max==0.0)
        max=1.0;
      for (j=0; j<100; j++)
        matrix[i][j] = matrix[i][j]/max;
    }
}
```

Figure 5. A function full of stride patterns.

To verify this hypothesis, we tested a small function given in figure 5. Each row of a matrix is scaled according to the highest absolute value in the row. The variables i and j are iteration variables with a stride of 1 and a range of $[0 \ldots 199]$ and $[0 \ldots (98 \text{ or } 99)]$, but the compiler also generates internal variables such as $j * 8$, $\&matrix[i]$ and $\&matrix[i][j]$, which have different strides and ranges. The branches are computed by a *slt* (set less than) instruction, so these instructions follow an (almost) constant pattern. In order to measure the influence of stride patterns on the usage of the level-2 table of the FCM, we checked for each access to the FCM whether it is part of a stride pattern. We used the simple indication that a value is part of a stride pattern if a stride predictor can correctly predict it. Each time the FCM was accessed to predict a value in a stride pattern, we incremented a counter associated with the entry in the level-2 table (4096 entries). Both the level-1 table and the stride predictor have 64K entries. We sorted the counters for the entries in the level-2 table in descending order (Figure 6(a)).

This figure shows how many references to the level-2 table are part of stride patterns and how many entries are allocated to stride patterns (horizontal axis).

The high peak at the left side is the result of the almost constant patterns from the *slt* instruction. The strides are spread over the rest of the table. More than 100 entries

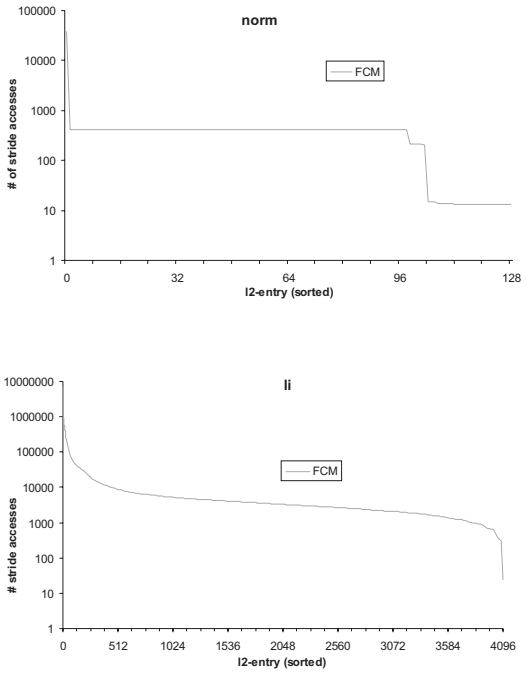
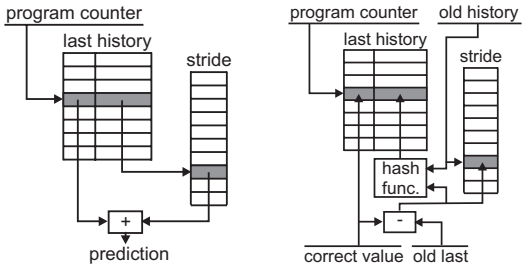


Figure 6. Number of accesses to a particular level-2 entry, based upon a history that is part of a stride pattern.



(a) Predicting a value (b) Updating the predictor

Figure 7. Differential

Context	Value	$\frac{Accesses}{Iteration}$
1 1 1	1	4
1 1 -6	1	1
1 -6 1	1	1
-6 1 1	1	1

Figure 8. Example stride pattern stored in DFCM level-2 table.

are accessed more than 100 times, in fact every entry is accessed at least 5 times.

We applied the same procedure to a real benchmark (*li*, see figure 6(b)). The behavior is similar, only the peak in the beginning is higher (the trace of *li* is longer than the simple function and contains more constant patterns) and the curve is smoother (i.e., the program contains multiple strides and not all loops are executed the same number of times).

3. Differential Finite Context Method

In order to reduce the problems associated with stride patterns, we propose to store the difference between the values in the predictor tables, instead of the values themselves. This way, all stride patterns appear to the FCM as if they were constant patterns, containing only one value.

By storing the differences between values, patterns can become more easily predictable. For a stride pattern (e.g., 0 1 2 3), the DFCM predictor will remember the last value 3 and the history of differences: 1 1 1. The history of differences is now a constant pattern. Therefore, the DFCM can correctly predict stride patterns, even if they have not been repeated yet. Also, stride patterns occupy only one entry in the level-2 table. Only when the pattern ends and it is restarted from the beginning, as in the sequence 0 1 2 3 4 5 6 0 1, are multiple entries in the level-2 table used to store the pattern. The example first shown in figure 4 is now repeated for the DFCM in figure 8. We can see that now all patterns map to the same entry, except those just following a counter reset, which occupy also one entry, but are accessed far less frequently. Furthermore, all stride patterns with the same stride map to the same entries. Therefore, the DFCM can store and predict stride sequences very efficiently.

When the pattern is not a stride pattern, the DFCM predictor retains as much information as the FCM predictor. For the pattern 0 4 2 1, the DFCM stores the last value 1 and a history of differences: 4 -2 -1. Both forms of storing the history are equivalent (i.e., one representation can be derived from the other), although working with the differences requires additional storage space for the last value. However, by constructing the history with differences, non-stride patterns might interfere with each other in the DFCM even when they did not interfere in the FCM, or vice versa.

The DFCM is a two-level predictor, just like the FCM. The level-1 table stores the last value and a hashed history of differences between the recently occurring values. The level-2 table contains the next difference, corresponding to a certain history of differences. The last value is *not* used to determine the index of the level-2 table. Otherwise, stride patterns would be scattered all over the level-2 table again, and nothing would be gained.

The actions taken to predict a value are a combination of the actions taken by the FCM and the stride predictor. Using the program counter of the instruction, the last value and the hashed history of differences is read from the level-

1 table (Figure 7(a)). The history is used as an index in the level-2 table, where the predicted difference is found. This difference is added to the last value, found in the level-1 table, to produce the predicted value.

Updating the DFCM is also a combination of the actions taken by the stride predictor and the FCM (Figure 7(b)). A new difference is computed by subtracting the correct value from the last value. This difference is stored in the level-2 table and is added to the history by the hashing function. The hashing function can be any hashing functions that can be used in the FCM. The last value is also updated with the correct value.

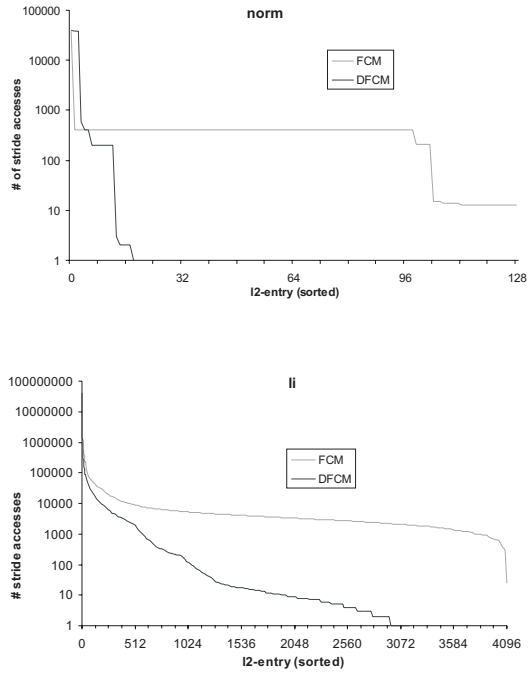


Figure 9. Number of accesses to a particular level-2 entry, based upon a history that is part of a stride pattern.

Earlier, we computed the number of accesses part of a stride to level-2 entries for the FCM and sorted them in descending order. Figure 9 shows the results for the DFCM predictor. The effect of placing strides in the history of the FCM predictor is that all patterns with equal stride map to the same level-2 entries. For the simple function (Figure 9(a)), the result is that a few level-2 entries are used much more frequently for stride patterns in the DFCM than in the FCM. Furthermore, the number of entries involved in stride patterns is much lower. Only 12 entries are used more than 100 times when the DFCM predicts strides. In comparison, the FCM uses more than 100 entries more than 100 times to store strides. The DFCM stores stride patterns in the prediction table using far less level-2 entries. As a consequence, more entries are available to other, harder to predict patterns.

The same trend can be noted for the benchmark *li* (Figure 9(b)). However, *li* has many different strides. Therefore, still a lot of entries are needed to store the stride patterns. The DFCM predictor uses 582 entries more than 1000 times to store stride patterns, while the FCM uses almost all entries (3801 out of 4096) over 1000 times, 7 times more. Thus, the DFCM predictor is far more efficient with regard to stride patterns. However, this fact alone does not make the DFCM predictor a more accurate predictor. We will show in the next section that the entries occupied by stride patterns are also occupied by non-stride patterns, which causes interference between the two. By storing the stride-patterns more efficiently, negative interference can be greatly reduced.

The DFCM predictor has also some disadvantages. More space is needed for the level-1 table, as not only the history but also the last value has to be stored. Because differences between values are used, addition and subtraction operations are required when predicting values and updating the prediction tables. This will lengthen the access time of the DFCM predictor, although the value predictor is usually not in the critical path of the processor, since the prediction can start very early in the pipeline and the prediction is only needed when the instruction enters the instruction window.

4. Evaluation

We evaluate only the value predictor itself, and not its embedding in an actual processor, since this creates many additional parameters and introduces additional effects which are only partially understood (e.g., delay between prediction and update, confidence mechanisms) and this obfuscates the comparison between the predictors.

Thus we were able to use trace-based simulations. The traces were generated on-the-fly for each configuration of the value predictor by a SimpleScalar 2.0 simulator [1] (sim-safe).

The benchmarks were taken from the SPECint95 suite. We did not consider SPECfp95 since these have in general more ILP available and hence have less to gain from value prediction. The benchmarks were compiled with gcc 2.6.3 for SimpleScalar (MIPS instruction set) with optimization flags "`-O2 -funroll_loops`". We use small input files (often the training inputs, see table 1) and simulate only the first 200 million instructions, except for m88ksim where we skip the first 250M. Only integer instructions that produce an integer register value are predicted, including load instructions. For instructions which produce two result registers (e.g. multiply and divide) only one is predicted. Finally, value prediction was not performed for branch and jump instructions. The presented results show the arithmetic mean over all SPECint benchmarks, weighted by the number of predicted instructions.

Benchmark	options, input	predictions
compress	80000 e 2131	140M
cc1	cccp.i	133M
go	30 8	157M
ijpeg	-image_file vigo_ref.ppm -GO	155M
li	7queens.lsp	123M
m88ksim	-c ctl.raw.lit	139M
perl	scrabbl.pl scrabbl7_train.in	126M
vortex	vortex.ref.lit	122M

Table 1. Description of the benchmarks.

Sazeides studied the effect of the hashing function and the order for FCM predictors [14]. We will use his FS R-5 hashing function. This function provided the highest accuracy for most configurations and was close to the highest achieved accuracy for the others. If the level-2 table has 2^n entries, this function folds each value into n bits (by XORing), then each value is shifted to the left by $5 \times age$ bit positions ($age = 0$ for the last value, 1 for the previous one,...) and finally those shifted values are XORed together into the index. Thus, the order varies for different level-2 table sizes: $order = \lceil \frac{n}{5} \rceil$; resulting in the following relation:

L2 size	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
order	2	2	3	3	4	4	4

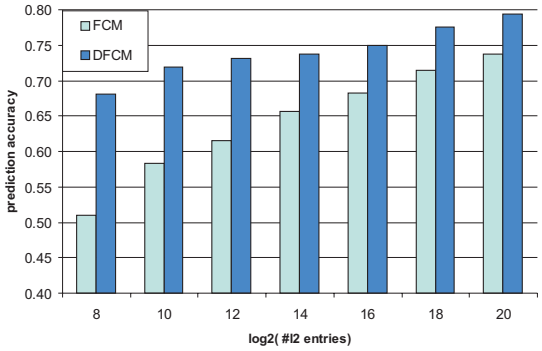
For the DFCM predictor, we used the same hashing function for the history. One could argue that this way, DFCM has a higher order than FCM (n strides and one last value equals the information of $n+1$ last values). On the other hand only the strides are used to index the L2 table. In any case, this is not to the disadvantage of FCM since both order and hashing function (R-5) are (near) optimal, while we did not try to optimize the order and the hashing function for DFCM.

The confidence counter in the stride predictor is a 3-bit counter, which is increased by 1 on a correct prediction and decreased by 2 on a wrong prediction. The stride value is changed whenever the confidence counter is less than 7 (the maximum value).

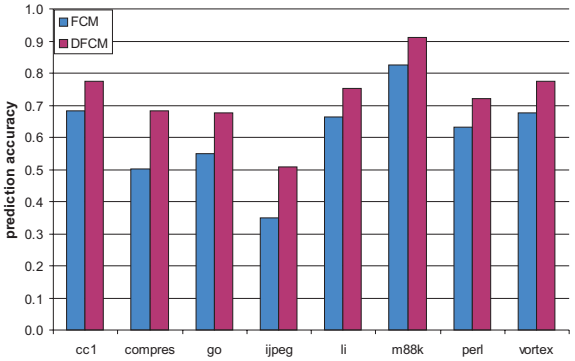
4.1. DFCM vs. FCM

The average value prediction accuracy of the DFCM for the eight SPECint benchmarks is measured and plotted in Figure 10(a). This figure shows the prediction accuracy of the FCM and the DFCM for a level-1 table with 2^{16} entries and multiple level-2 sizes (horizontal axis, $\log_2$(l2-entries)).

By using the DFCM predictor, the number of correct predictions increases by 8% over FCM (from .74 to .79) for very large tables. The increase is more pronounced for smaller, more realistic table sizes and reaches a maximum of 33% (from .51 to .68) This corroborates the hypothesis that the improvement of the DFCM is related to interference in the level-2 table, since this interference deteriorates when



(a) 2^{16} level-1 entries



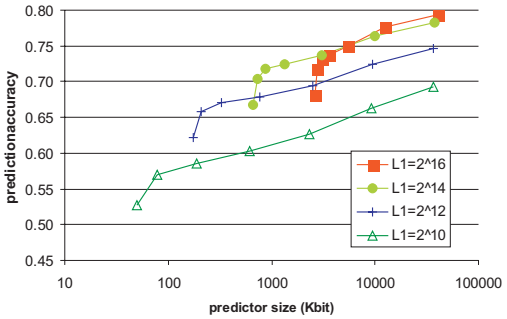
(b) 2^{16} level-1 entries, 2^{12} level-2

Figure 10. Prediction accuracy of the FCM vs. the DFCM.

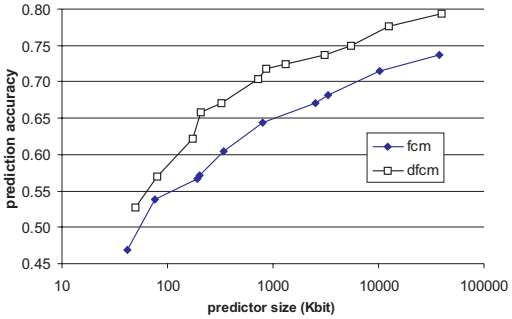
the level-2 table is smaller. Figure 10(b) shows the results for each benchmark for a 2^{12}-entry level-2 table. The average accuracy increases by 19% (.62 to .73); the minimal increase is 8% (m88ksim), the maximum is 46% (ijpeg), all others are in the 13-37% range.

Figure 11(a) plots the prediction accuracy of the DFCM predictor versus the total amount of storage the predictor needs. Each curve corresponds to a fixed size of the level-1 table, while the number of entries in the level-2 table increases from 2^8 to 2^{20}. If we compare this graph with the graph for FCM (figure 3), we observe that (i) the accuracies are higher and (ii) the influence of the level-2 table size diminishes earlier, and the knee is sharper.

To summarize a graph such as figure 11(a), one can only plot the points for those configurations that have a higher accuracy than all other configurations with the same or smaller size (Pareto-graph). We can construct a Pareto-graph for



(a) various DFCM predictors



(b) DFCM vs. FCM: Pareto graphs

Figure 11. Prediction accuracy vs. size

both FCM and DFCM from figure 3 and 11(a), this results in figure 11(b). DFCM increases FCM's accuracy by .06-.09, except for small table sizes. The difference is maximal for realistic table sizes, e.g. .09 for ±200 Kbit (.66 vs. .57, an increases of 15%)

4.2. Interference analysis

We assume the DFCM has a higher prediction accuracy than the FCM because the stride patterns interfere less with other patterns in the level-2 table. To test this, we measured the aliasing in a predictor. We put each prediction in one out of five aliasing categories:

l1 Aliasing at the level-1 table is detected by checking whether all the values recorded in the history — used to access the level-2 table now — were produced by the same instruction that is being predicted now.

hash During an update, for each level-2 entry, the complete history (unhashed) is stored beside the prediction. A subsequent prediction using that level-2 entry then checks if its complete history matches the recorded one.

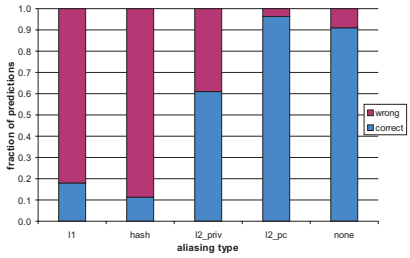


Figure 12. Several aliasing types: prediction accuracy

l2_priv Besides the global level-2 table, a local level-2 table is maintained for each level-1 entry. We check whether using the local table yields the same prediction as using the global one.

l2_pc During an update, each l2-entry is tagged with the PC of the updating instruction. A subsequent prediction using that entry then checks whether the tag matches.

none None of these aliasing detection rules apply.

Only the first rule that applies is counted.

The classification results for 2^{12} entry level-1, 2^{12} entry level-2 predictors are shown in figure 13, while the prediction accuracy (FCM) for each of the aliasing types is shown in figure 12.

As expected, both *l1* and *hash* show a very low prediction accuracy, since the assumed history in the level-1 or level-2 table is different from the actual one, while both *none* and *l2_pc* are very predictable. The accuracy of *none* shows the power of the principle of FCM, the accuracy of *l2_pc* that aliasing between identical patterns originating from different instructions is not destructive. Even the accuracy of *l2_priv* is above 50%, we believe a higher learning time in the case of private level-2 tables is the cause.

Figure 13 shows that *hash* and *l2_pc* are the most common aliasing types. No aliasing at all is rather seldom. Comparing the numbers for FCM and DFCM learns that DFCM has even fewer cases of no aliasing at all. Thus, there is even more aliasing in the level-2 DFCM table! Further on, the *l2_pc* case arises almost twice as often, and the *hash* case decreases.

This translates in the aliasing types observed for the mispredictions (figure 14), as a fraction of all predictions, thus the height of a bar is the global misprediction rate for that benchmark. Only the first three types of aliasing play a significant role, and *hash* is the dominant one.

Apparently, DFCM partially avoids the quasi-random aliasing by the hashing function by mapping more entries intentionally to the same level-2 entry, in such a manner

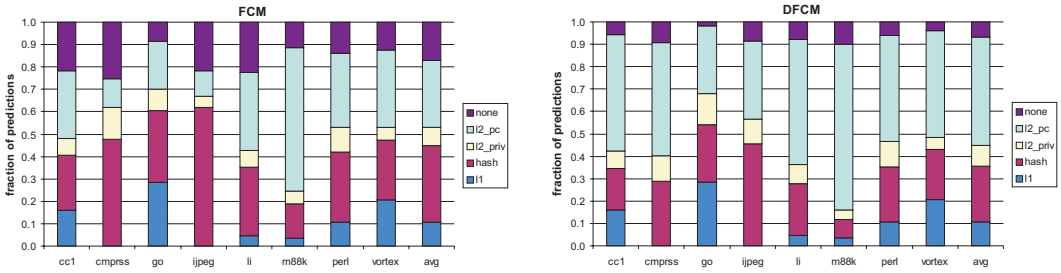


Figure 13. Alias analysis for *all* predictions.

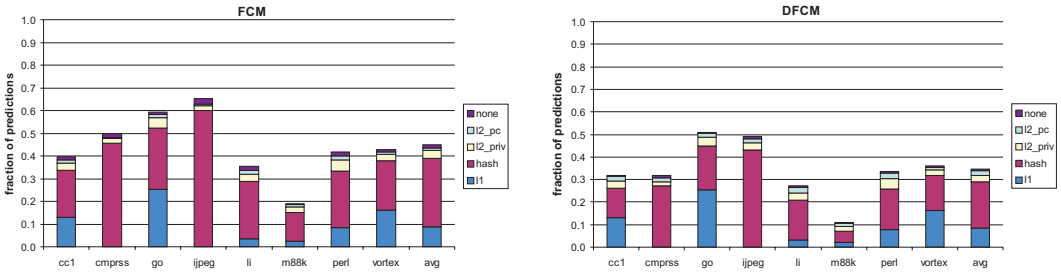


Figure 14. Alias analysis for *wrong* predictions.

that the expected aliasing is neutral. This is the cause of the drop in *none* and the rise in *l2_pc*. Since both types are well predictable, this shift has no negative effect on the prediction accuracy.

Now the hashing function is under less pressure. The *hash* aliasing drops from 34% to 25%, and the total number of mispredictions drops by almost the same amount (10%). This holds for all benchmarks. This means the gain from DFCM is almost completely due to the reduction of the aliasing caused by the hashing function.

We should note however that the hashing function remains responsible for the majority of the mispredictions (59%), there is still plenty of room for improvement. These results suggest that the design of a confidence estimator for a (D)FCM predictor should include tagging the level-2 table with some information to track hash-aliasing, although we have not tested this. Some bits of a second hashing function, orthogonal to the main one, seems to be a good choice for the tag.

4.3. DFCM vs. hybrid FCM/stride

Because the FCM predictor does not predict stride patterns as well as a stride predictor, several researchers have proposed to combine the FCM with a stride predictor in a hybrid predictor [15, 17]. A hybrid predictor can be constructed in the following manner. For each instruction, three predictors are used: a FCM, a stride predictor and a meta-predictor (Figure 15). The meta-predictor is typically a set of saturating counters, indexed by the program counter. Its task is to predict whether the FCM or the stride predictor will make a correct prediction. When an instruction has executed, the FCM and stride predictor are updated with the correct value and the meta-predictor is updated with the information of which predictor was correct.

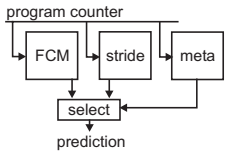


Figure 15. A hybrid predictor.

How a meta-predictor is best devised is not clear. Therefore, we assume a perfect meta-predictor. The perfect meta-predictor always knows which predictor is right. The prediction accuracy of the hybrid predictor is shown in Figure 16 (label STRIDE+FCM). All predictors have 2^{16} en-

tries in their level-1 table and the stride predictor has the same number of entries in its table. The number of entries in the level-2 table of the DFCM, hybrid predictor and FCM is varied (the x axis represents the logarithm (base-2) of the number of entries).

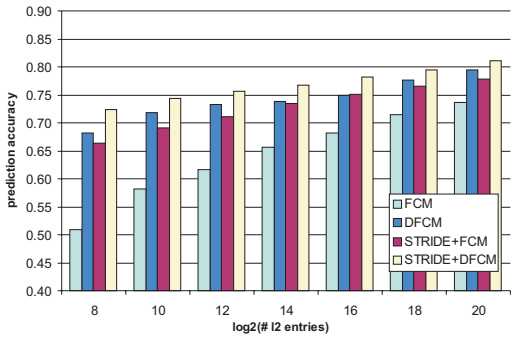


Figure 16. Hybrid predictors.

The DFCM predictor outperforms the perfect hybrid predictor, regardless of the size of the level-2 table, but the difference between the predictors is small. On the other hand, implementing a perfect meta-predictor is impossible. Therefore, the DFCM can outperform any hybrid predictor of the discussed type.

We also show the prediction accuracy of a perfect hybrid predictor, which is composed of the DFCM and a stride predictor (label STRIDE+DFCM in figure 16). This perfect hybrid predictor is only between .02 and .04 better than the single DFCM. From this, we conclude that practically all stride patterns are correctly predicted by the DFCM.

4.4. Size of difference values

The DFCM stores differences between consecutive values in the level-2 table; these values seldomly require the full 32-bit width. Thus, we can try to reduce the storage requirements by storing only a partial difference in the level-2 table. This is not practical for the FCM predictor, as in that case, the level-2 table contains the actual values, often pointers.

Simulations proved that we can indeed omit many bits (by using 16 bits: .01-.03 drop in accuracy, 8 bits: .05-.08 drop), but this proved not very useful:

- for *small* level-2 tables, the storage requirements are dominated by the level-1 table, so the accuracy drops without a significant drop in size
- for *large* level-2 tables, the level-2 table dominates the size, but the prediction is only weakly dependent on the level-2 size (see Figure 11(a)). Reducing the number of entries in the level-2 table by four has the same effect on the table size as using only 8 bits, but has an even lower impact on accuracy.

4.5. Delayed update

Although we did not perform any cycle-accurate simulations, we did want to have an idea of the behavior of DFCM under delayed update. This is usually not done when evaluating predictors [15, 14], but this phenomenon can seriously impact performance.

When modeling a delay d, a prediction is performed, but the update of the tables is only done after d other predictions have been performed. Thus, if the same static instruction occurs twice in the instruction stream within a distance d, the second prediction is based upon stale history information. The average number of instructions d between prediction and update can be estimated to be the average number of instructions present in a processor.

The results are in figure 17for a 2^{16}-entry level-1, 2^{12}-entry level-2 table. Both FCM and DFCM suffer significantly from the delayed update, DFCM slightly more, but the overall behavior is the same for both techniques.

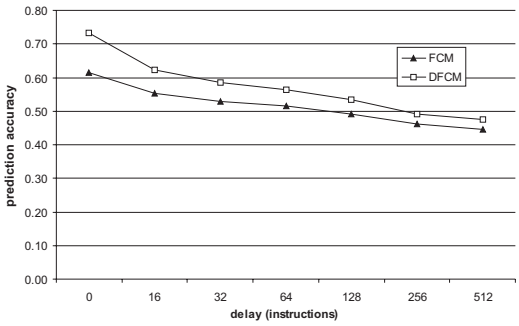


Figure 17. Prediction accuracy under delayed update.

5. Related Work

Several studies have tried to increase the prediction accuracy by combining several predictors and selecting the most confident one [12, 13, 17]. We have shown this technique is not as accurate as DFCM and consumes more hardware.

Others have tried to increase the efficiency by using multiple, separate predictors and assigning each instruction to *one* of them by means of dynamic classification [9, 12]. This technique seems interesting and can also remove the stride patterns from the FCM level-2 table, but has serious drawbacks. A fixed partitioning of the available resources is introduced between the different predictors, while ours can dynamically adjust the partitioning: all constant patterns share only one level-2 entry, stride patterns visit only as many level-2 entries as there are different strides (ignoring boundary effects), the rest is available for FCM patterns. Besides this, the dynamic classification is a difficult

job: Rychlik's classifier [12] marks more than 50% of the instructions as *unpredictable by any predictor*, Lee reports 24% [9]. Only Rychlik reports an overall prediction accuracy: 43%.

Another way to improve efficiency is to selectively apply value prediction: only for loads [2, 11] or by using more sophisticated techniques [3, 7]. This approach is complementary to ours, since selectively predicting values does not improve the behavior of the FCM with regard to stride patterns.

6. Summary and Conclusions

In this paper, we have studied various context based predictors. First, we have shown that traditional FCM predictors can outperform last value predictors and stride predictors for all but the smallest storage capacities and that accurate FCM predictors require huge level-2 tables. Secondly, we have searched for the reason of this dependence on large level-2 tables and have shown that stride patterns are treated in a very inefficient way, and that the easy to predict stride patterns occupy a large part of the level-2 table and interfere with the much harder to predict non-stride patterns.

We have proposed a new context based predictor, the *differential finite context method predictor* (DFCM), that uses the differences between consecutive values as the context instead of the values themselves. This history is used as an index for the level-2 table to obtain a difference prediction, which is added to the last value seen to obtain the prediction.

We have demonstrated that this predictor treats stride patterns more efficiently. We have shown that this predictor is less dependent on the size of the level-2 table and outperforms a similar FCM predictor by up to 33%, and by 15% when accounting for the additional space required by the last values. We have shown that the FCM predictor severely suffers from aliasing introduced by the hashing function, and DFCM tackles this problem, although considerable room for improvement remains.

Finally, we showed that a DFCM predictor is superior to a hybrid FCM-stride predictor that uses a perfect selection mechanism between the stride and FCM prediction.

7. Acknowledgements

The authors wish to express their gratitude to Henk Neefs for the fruitful and inspiring discussions, and Frederik Habils and Bjorn De Sutter for proofreading earlier drafts of this article. Bart Goeman is supported by the Fund for Scientific Research-Flanders (FWO) project 3G003699. Hans Vandierendonck is supported by a grant from the Flemish Institute for the Promotion of the Scientific-Technological Research in the Industry (IWT). Koen De Bosschere is research associate with the Fund for Scientific Research-Flanders.

References

[1] D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: the simplescalar tool set. Technical Report TR-1342, University of Wisconsin-Madison, 1997.

[2] M. Burtscher and B. G. Zorn. Exploring last n value prediction. In *Parallel Architectures and Compilation Techniques (PACT)*, 1999.

[3] B. Calder, G. Reinman, and D. M. Tullsen. Selective value prediction. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 64–74, May 1999.

[4] R. J. Eickemeyer and S. Vassiliadis. A load instruction unit for pipelined processors. *IBM Journal of Research and Development*, 37(4):547–564, July 1993.

[5] F. Gabbay and A. Mendelson. Speculative execution based on value prediction. Technical Report 1080, Technion - Israel Institute of Technology, 1997.

[6] F. Gabbay and A. Mendelson. Using value prediction to increase the power of speculative execution hardware. *ACM Transactions on Computer Systems*, 16(3):234–270, Aug. 1998.

[7] B. Goeman, H. Neefs, and K. D. Bosschere. Increasing the efficiency of value prediction in future processors by predicting less. In *Proceedings of ParCo99*, 1999.

[8] J. González and A. González. The potential of data value speculation to boost ILP. In *Proceedings of the 12th ACM International Conference on Supercomputing (ICS)*, July 1998.

[9] S.-J. Lee, Y. Wang, and P.-C. Yew. Decoupled value prediction on trace processors. In *Proceedings of the Sixth International Conference on High-Performance Computer Architecture*, pages 231–240, Jan. 2000.

[10] M. Lipasti and J. Shen. Exceeding the dataflow limit via value prediction. In *Proceedings of the 29th Annual International Symposium on Microarchitecture*, Dec. 1996.

[11] M. Lipasti, C. Wilkerson, and J. Shen. Value locality and load value prediction. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1996.

[12] B. Rychlik, J. Faistl, B. Krug, A. Y. Kurland, J. J. Sung, M. N. Velev, and J. P. Shen. Efficient and accurate value prediction using dynamic classification. Technical report, Carnegie Mellon University, Microarchitecture Research Team, 1998.

[13] B. Rychlik, J. Faistl, B. Krug, and J. P. Shen. Efficacy and performance impact of value prediction. In *Parallel Architectures and Compilation Techniques(PACT)*, Oct. 1998.

[14] Y. Sazeides and J. E. Smith. Implementations of context based value predictors. Technical Report ECE97-8, Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Dec. 1997.

[15] Y. Sazeides and J. E. Smith. The predictability of data values. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, Dec. 1997.

[16] Y. Sazeides, S. Vassiliadis, and J. E. Smith. The performance potential of data dependence speculation and collapsing. In *Proceedings of the 29th Annual International Symposium on Microarchitecture*, Dec. 1996.

[17] K. Wang and M. Franklin. Highly accurate data value prediction using hybrid predictors. In *4th International Conference on High Performance Computing*, Dec. 1997.