The Accuracy of Initial Prediction in Two-Phase Dynamic Binary Translators

Youfeng Wu, Mauricio Breternitz, †Justin Quek, ‡Orna Etzion, Jesse Fang

Corporate Technology Group (CTG) ‡Software and Solutions Group (SSG)
Intel Corporation
{youfeng.wu, mauricio.breternitz.jr, orna.etzion, jesse.z.fang}@intel.com

†Department of Electronic and Computer Engineering
University of Illinois, Urbana-Champaign
quek@crhc.uiuc.edu

Abstract

Dynamic binary translators use a two-phase approach to identify and optimize frequently executed code dynamically. In the first step (profiling phase), blocks of code are interpreted or quickly translated to collect execution frequency information for the blocks. In the second phase (optimization phase), frequently executed blocks are grouped into regions and advanced optimizations are applied on them.

This approach implicitly assumes that the initial profile of each block is representative of the block throughout its lifetime. This study investigates the ability of the initial profile to predict the average program behavior. We compare the predicted behavior of varying lengths of the initial execution with the average program behavior for the whole program execution, and use the prediction from the training input as the reference. Our result indicates that, for the SPEC2000 benchmarks, even very short initial profiles have comparable prediction accuracy to the traditional profile-guided optimizations using the training input, although the initial profile is inadequate for predicting loop trip count information for some integer programs and several benchmarks can benefit from phase-awareness during dynamic binary translation.

1. Introduction

Most dynamic binary translators (e.g. IA32EL [2], Transmeta [6], Daisy [7], Dynamo[1], etc) use a two-phase approach to identify and optimize frequently executed code dynamically. In the first step (profiling phase), blocks of code are interpreted or translated without optimization to collect execution frequency information for the blocks. In the second phase (optimization phase), frequently executed blocks are grouped into regions and advanced optimizations are applied on them. For example, the profiling phase in Intel's IA32EL [2] converts each IA32 block quickly into IPF (Itanium Processor Family) code with instrumentation for collecting "use" count, the number of times the block is visited, and "taken" count, the number of times its conditional branch is taken. When the use count for a block reaches a *retranslation threshold*, the block is registered in a pool of candidate blocks. When a sufficient number of blocks are registered or when a block is registered twice, the optimization phase begins to retranslate the candidate blocks. The optimization phase uses the ratio taken/use as the branch probability to form regions (e.g. hyper-block regions and hyper-block loops [15]) for optimizations and instruction scheduling [11].

This approach implicitly assumes that the execution profile of each block in the profiling phase (*initial profile*) is representative of the block throughout its lifetime. In particular, a region is selected for optimization with the assumption that it infrequently takes its side exits and is thus candidate for advanced optimizations. If this assumption is incorrect, however, the optimized regions may often take their side exits, and the program performance will suffer.

Recent studies [3][12][14][16] have shown that some programs exhibit multiple phases. For those programs, a single profiling phase is clearly unable to respond to the phase changes. However, it is still open whether the continuous optimization for capturing phase changes is able to improve performance significant enough to offset the overhead of continuous profiling and re-optimization [10][14]. Therefore, we consider the two-phase approach a practical solution if the initial profile approximates the profile obtained with the training input in traditional profile-guided optimization [4].

Since the objective of the training input is to predict the average program profile, we also compare the initial

Proceedings of the International Symposium on Code Generation and Optimization (CGO 2004)
0-7695-2102-9/04 $20.00 © 2004 IEEE

profile with average program behavior. Notice that the average program profile does not necessarily represent the best branch behavior that the optimization phase can explore. Specifically, the optimization phase may duplicate blocks to explore correlation between the input paths to the output paths of the blocks [22], and this opportunity will not be reflected in the average profile. Still, we consider the average profile a reasonably good base for evaluating the initial profile and the profile from the training input.

In this paper, we will use the term "initial prediction" to refer to the initial profile as a prediction for the average program behavior. We will use the IA32EL infrastructure for this study, and compare the initial prediction of varying lengths of the retranslation thresholds with the average program behavior, and use the traditional profile-guided predictions as the reference. Our result indicates that, for the SPEC2000 benchmarks, the initial prediction with retranslation thresholds as small as 500 to 2000 can have comparable prediction accuracy to the traditional profile-guided optimizations using the training input. This is significant, as the initial prediction with these retranslation thresholds uses only a tiny fraction of profiling operations (e.g. less than 1%) required for the training run.

Our results also indicates that, probably due to limited number of profiling operations, the initial profile seems inadequate for predicting loop trip count information for some integer programs and several benchmarks can benefit from phase-awareness with longer or multiple-phase profiling.

The rest of the paper is organized as follows. Section 2 outlines the methodology for this study. Section 3 describes the technical issues to implement the methodology. Section 4 presents the experimental results, and Section 5 summarizes the paper and discusses the future directions.

2. Methodology

To evaluate the accuracy of the initial profile, we first run a program with a retranslation threshold T, e.g. 500, and output information for regions (i.e. the entry, exits, and member blocks) that are retranslated by the optimization phase as well as the "use" and "taken" values for the blocks in the regions. For blocks that are not included in any region, we output their use and taken counts at the end of the program execution. We call the information the "initial prediction with threshold T", denoted INIP(T). We then run the same program without optimization and output "use" and "taken" count information for all the blocks at the end of the program execution. In this case, the "use" and "taken" counts are the average profile for the entire execution of the program, and we call the information the "average behavior of the program", denoted AVEP. Finally we run the same program without optimization and with the training input (notice that INIP(T) and AVEP are obtained with reference input), and output "use" and "taken" count values for all the blocks at the end of the program execution. We call the information INIP(train).

We compare INIP(T) with AVEP to determine whether or not the INIP(T) accurately approximates AVEP. We also compare INIP(train) with AVEP to obtain a reference from the profile with the training input. From the two comparisons, we can see how the accuracy of the initial prediction compares to that of the traditional profile-guided optimization with the training input.

Notice that all the blocks in INIP(T) have similar execution frequencies (i.e. the "use" counts) between T and 2*T. That is because the optimization phase waits until a block is executed T times before placing it in the candidate pool and stops collecting "use" counts for a block once it is optimized. Therefore, the relative order of the block frequencies in INIP(T) is usually not meaningful. Consequently, many of the well known techniques for comparing profiles that rely on relative order of the profile data, such as the "weigh match" and "key match" [19] and overlapping percentage [8], cannot easily be applied for comparing INIP(T) and AVEP.

The optimization phase uses the branch probabilities (the ratio of taken and use counts) of the blocks to form regions. If the branch probability in the initial phase is different from that in the late execution, the selected regions may not perform well. Therefore, we can use metrics that will only involve branch probabilities in both INIP(T) and AVEP, and block frequencies in AVEP to measure the prediction accuracy. In particular, we measure the standard deviation [9] of branch probabilities between INIP(T) and AVEP. We also measure the standard deviation of region completion probabilities between INIP(T) and AVEP, and the standard deviation of loop-back probabilities between INIP(T) and AVEP. We believe that these measurements directly measure the quality of the profiles, and the comparison of the profiles using the standard deviations can be easily interpreted with statistical intuitions.

In addition to computing the standard deviations (SD), we also collect the range-based matching of branch probabilities and loop-back probabilities to evaluate the initial predictions. These measurements provide further insights into the initial profiles.

2.1. SD of Branch Probabilities

The branch probability (BP) for each block is computed as the ratio of taken/use of the block. We use

Sd.BP(T), the standard deviation of the branch probabilities of INIP(T) (the deviation) from that of AVEP (the average), to measure the accuracy of the initial prediction. Assume a program has N blocks in INIP(T) and AVEP. Assume the branch probability of block i is BT(i) in INIP(T) and BM(i) in AVEP, and the weight of the block i is W(i) = block i's frequency in AVEP, then

$$\text{Sd.BP(T)} = \sqrt{\frac{\sum_{i=1}^{N}(BT(i)-BM(i))^2 * W(i)}{\sum_{i=1}^{N}W(i)}}$$

When Sd.BP(T) is small, e.g. around 0.1, we may statistically say that INIP(T) represents a reasonably good prediction of the average program behavior, as the majority (about 68%) of predicted branch probabilities in INIP(T) are within 10% of the corresponding branch probabilities in AVEP, and most (about 95%) of predicted branch probabilities are within 20% of the corresponding branch probabilities in AVEP.

We also compute Sd.BP(train), the standard deviation between INIP(train) and AVEP. Intuitively, if Sd.BP(T) is close to Sd.BP(train), then the two-phase approach has similar prediction accuracy as the traditional profile-guided optimizations using the training input.

2.2. SD of Completion Probability

The completion probability (CP) for a non-loop region is the probability that the region executes from its entry to the last block without taking any side exit. A region selected for optimization should have a high completion probability, or its performance will suffer when execution frequently takes the paths not anticipated by the optimization.

We compute Sd.CP(T), the standard deviation between the completion probabilities of INIP(T) and AVEP, to measure the accuracies of the initial predictions for the average completion probabilities. Assume a program has M non-loop regions in INIP(T) and AVEP. Assume the completion probability of region j in INIP(T) is CT(j), that of the region in AVEP is CM(j), and the weight of the region is W(j) = the frequency of the entry block in AVEP, then

$$\text{Sd.CP(T)} = \sqrt{\frac{\sum_{j=1}^{M}(CT(j)-CM(j))^2 * W(j)}{\sum_{j=1}^{M}W(j)}}$$

2.3. SD of Loop-back Probability

The loop-back probability (LP) measures the likelihood that a loop region branches back to itself, which is directly related to the average trip count of the loop. Many advanced loop optimizations relies on accurate trip count information to determine their applicability, and the evaluation of loop-back probabilities are important to these optimizations. Notice that some of the non-loop regions identified by IA32EL may contain sub-regions that are loops, and we will not consider them as loop regions.

We compute Sd.LP(T), the standard deviation between the loop-back probabilities of INIP(T) and AVEP, to measure the accuracies of the initial predictions for the average loop-back probabilities. Assume a program has L loop regions in INIP(T) and AVEP. Assume the loop-back probability of loop region j in INIP(T) is LT(j), that of the loop region in AVEP is LM(j), and the weight of the region is W(j) = the frequency of the loop entry block in AVEP, then,

$$\text{Sd.LP(T)} = \sqrt{\frac{\sum_{j=1}^{L}(LT(j)-LM(j))^2 * W(j)}{\sum_{j=1}^{L}W(j)}}.$$

Notice that we do not compute Sd.CP(train) and Sd.LP(train) between INIP(train) and AVEP, as INIP(train) and AVEP are not optimized and thus have no region information for computing CP and LP. In the future, we may apply region formation algorithms [5][11] to construct regions in INIP(train) and compute Sd.CP(train) and Sd.LP(train).

3. Implementations

In the above, we have assumed that INIP(T) and AVEP have the same set of blocks and regions. This may not always be true as INIP(T) may duplicate a block into multiple regions, while AVEP won't form regions and therefore no block will be duplicated. In the following subsections, we address this and a few other implementation issues.

3.1. Normalize AVEP to INIP(T)

Figure 1 (a) shows a code segment with two nested loops from the Mcf SPEC2000 benchmark. The control flow graph for the code is shown in Figure 1 (b) (assuming the loop has been converted to a bottom test loop). Since the block containing Load1 is in both loops, the optimization phase may duplicate it into the two loop regions, as shown in Figure 2 (a), which could be the control flow graph seen in INIP(T). AVEP on the other hand will see a control flow graph as shown in Figure 2 (b). The blocks in Figure 2 (a) are annotated with the branch probabilities in INIP(T), and blocks in Figure 2 (b) are annotated with the branch probabilities in AVEP.

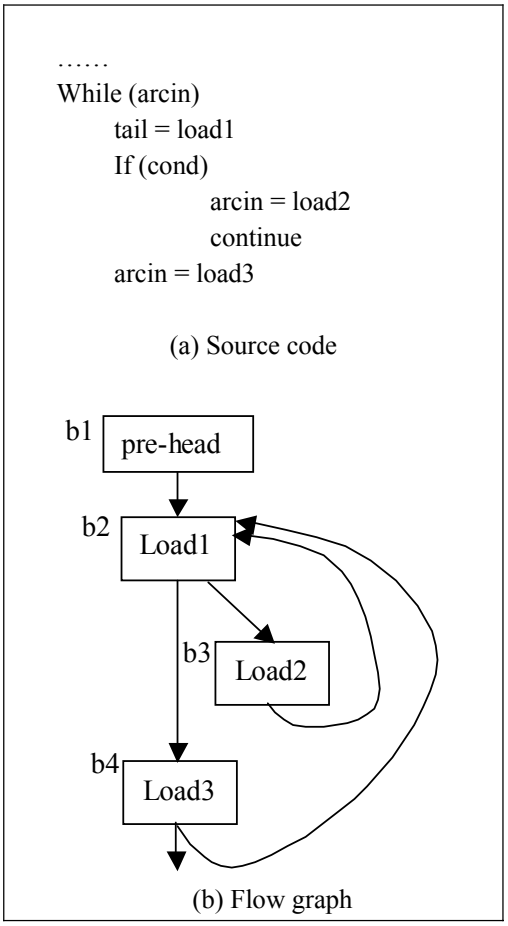

Figure 1. Structure of a loop in price_out_impl of the Mcf benchmark

In order for AVEP and INIP(T) to have the same blocks and regions, we normalize AVEP to the same control flow graph as seen by INIP(T). We refer to the normalized AVEP as NAVEP. The normalization duplicates some blocks in AVEP to multiple copies in NAVEP. We assign to each block in NAVEP the same branch probability as its original block in AVEP. Figure 3 shows the normalized AVEP with three copies for block b2. All blocks in Figure 3 are assigned the same branch probabilities as their original blocks in AVEP.

To compute the standard deviations, we also need to know the block frequencies for some of the duplicated blocks in NAVEP, which are the weights in the standard deviation computation. We cannot simply use the block frequency of the block in AVEP for all the duplicated blocks in NAVEP. Otherwise the weights for the duplicated blocks in NAVEP are higher than their real values since the sum of the block frequencies of all the blocks in NAVEP corresponding to the same block in AVEP should equal to the block frequency of the block in AVEP.

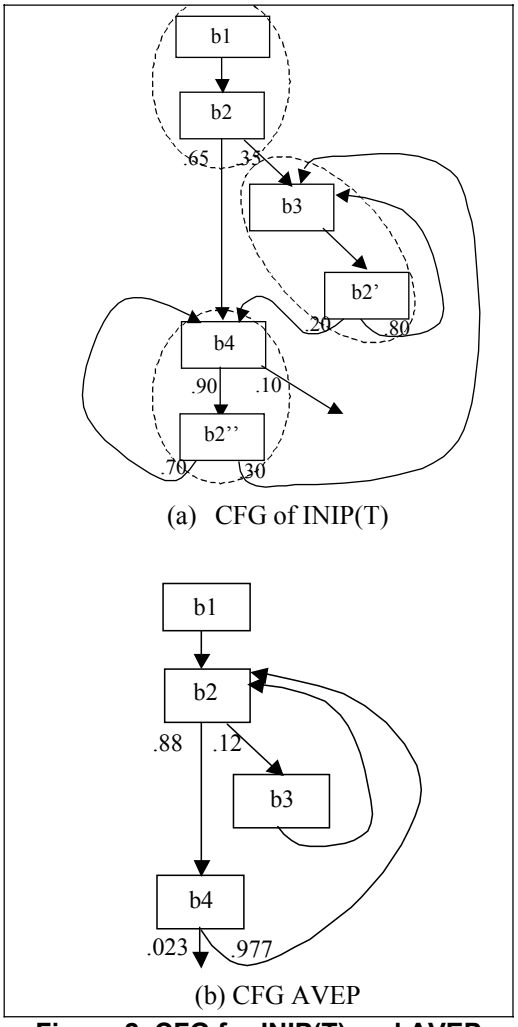

Figure 2. CFG for INIP(T) and AVEP

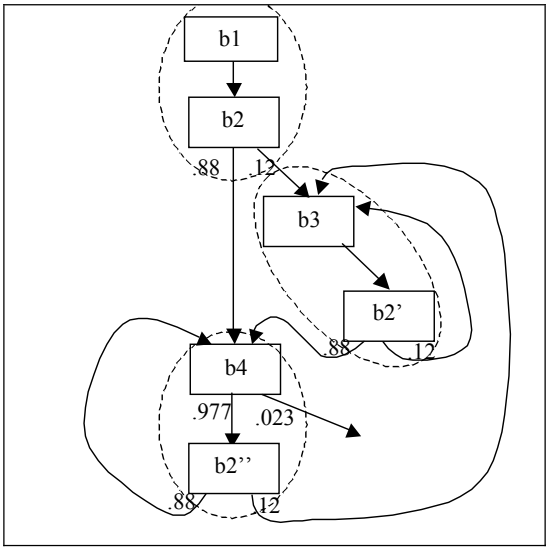

Figure 3. CFG for NAVEP

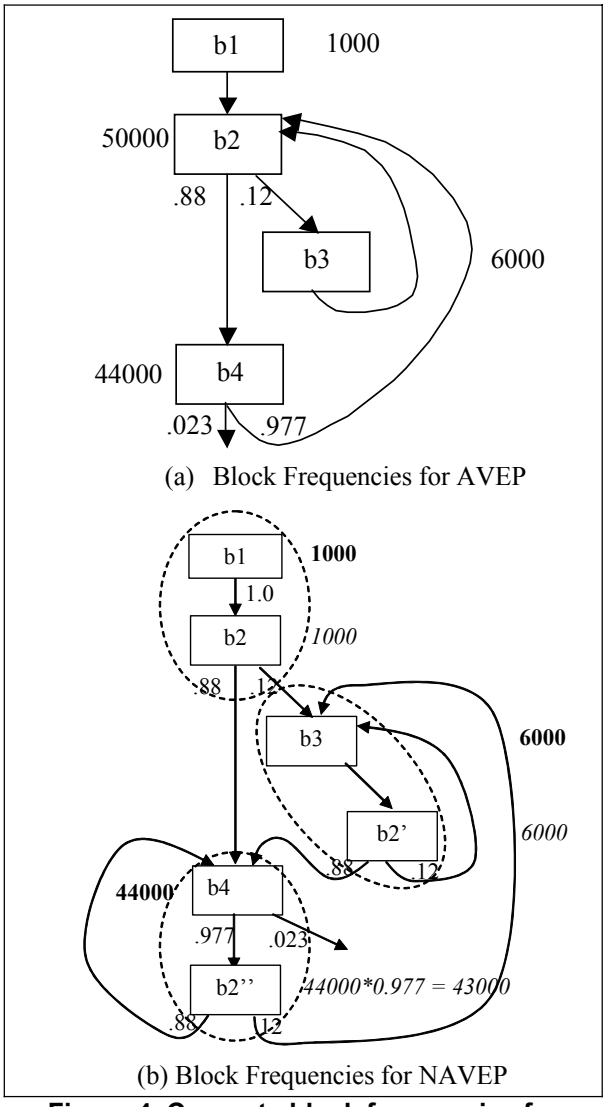

Figure 4. Compute block frequencies for duplicate blocks

We determine the block frequencies for the duplicated blocks by propagating the block frequencies from the non-duplicated blocks to all the duplicated blocks based on the branch probability assignment. For example, in Figure 4 (a), block b1, b3, and b4 are not duplicated. Starting from the block frequencies of these blocks in AVEP (assuming the frequencies are 1000, 6000, 44000 respectively), we can obtain the block frequencies for the three copies of the block b2 in NAVEP, as shown in Figure 4 (b), where the frequencies for non-duplicated blocks are shown in bold face while the propagated frequencies for the duplicated blocks are in *italic*. The sum of the frequencies for the three copies of the block b2 equals to 50000, the same as the frequency of the block b2 in AVEP. Notice that although in this example the region entry blocks are not duplicated, in general, it is possible that a region entry block is a duplicated block.

We use a general method known as the Markov Modeling of Control Flow [18] to determine the block frequencies for the duplicated blocks. In this method, we first formulate a system of linear equations, in which the frequencies of the non-duplicated blocks are the constant coefficients, and the frequencies of the duplicated blocks are the unknowns. From the system of the linear equations, we can find the solutions for the unknowns, which are the block frequencies of the duplicated blocks.

Now that NAVEP has the same set of blocks as INIP(T) and all the blocks in NAVEP are assigned both branch probabilities and block frequencies, we can compute the standard deviations. With respect to the INIP(T) in Figure 2 (a) and the NAVEP in Figure 4 (b), we can compute Sd.BP(T), the standard deviation of branch probabilities for the 4 blocks, Sd.CP(T), the standard deviation of completion probabilities for the non-loop region containing blocks b1 and b2, and Sd.LP(T), the standard deviation of loop-back probabilities for the two loop regions, as shown in Figure 5.

In the above example, we have a simple non-loop region whose completion probability is trivially computed. Each of the two loop regions has only one execution path from the loop entry back to itself, thus the loop-back probability is easily computed as the probability of the execution path. In general, however, non-loop or loop regions may be more complicated. We briefly describe the procedure for computing completion probability and loop-back probability in the next two subsections.

$$\text{Sd.BP(T)}=\sqrt{\frac{(.88-.65)^2*1000+(.977-.90)^2*44000+(.88-.70)^2*43000+(.88-.20)^2*6000}{1000+1000+6000+44000+43000+6000}}=\sqrt{0.045}=0.21$$

$$\text{Sd.CP(T)}=\sqrt{\frac{(1.0-1.0)^2*1000}{1000}}=0$$

$$\text{Sd.LP(T)}=\sqrt{\frac{(0.977*0.88-0.90*0.70)^2*44000+(0.12-0.80)^2*6000}{44000+6000}}=\sqrt{0.076}=0.27$$

Figure 5. Calculations of standard deviations for the sample program

3.2. Compute Completion Probability

Given a non-loop region such that each block is assigned a branch probability, the completion probability of the region is the likelihood that the execution starting at the region entry reaches the last block of the region. As an example, we take a look at the region shown in Figure 6. When the execution starts at block b5, the completion probability of the region is the probability of the execution arriving at block b8. For a region without side exits, the completion probability is always 1. The side exit branches reduce the completion probability of a region.

A general approach for computing completion probability is to assume the entry block of the region has an execution frequency of 1, and propagate that frequency all the way to the last block of the region. Then the frequency of the last block is the completion probability of the region. For the example in Figure 6, with the assumption that block b5 has a frequency of 1, block b6 will have a frequency of 0.4, and block b7 has a frequency of 0.6, and the frequency of block b8 will be 0.4 * 0.8 + 0.6 * 0.9 = 0.86. Therefore, the completion probability of the region is 0.86.

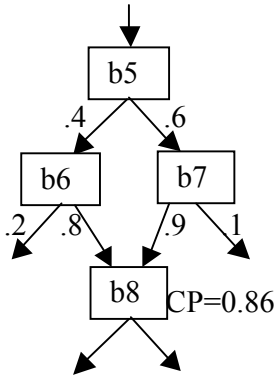

Figure 6. Completion probability for a non-loop region

3.3. Compute Loop-back Probability

Given a loop region such that each block is assigned branch probabilities, the loop-back probability of the region is the likelihood that the execution starting at the loop entry reaches back to the same loop entry block. If we create a dummy block such that the loop back edges are redirected to the dummy block instead of the entry block, then we can compute the loop-back probability by assuming the entry block of the loop has an execution frequency of 1, propagating the frequency all the way to the dummy block. The frequency of the dummy block will be the loop-back probability. An example is shown in Figure 7. The original loop is shown in (a), and the CFG with back edges directed to the dummy node is shown in (b). With the assumption that block *b5* has a frequency of 1, block b7 will have a frequency of 0.6, block b8 will have a frequency of 0.38, and the dummy node will have frequency of 0.38*0.9 + 0.6*0.9 = 0.886. Therefore, the loop-back probability of the loop region is 0.886.

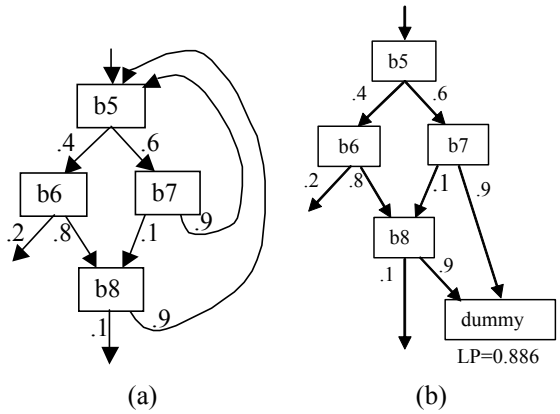

Figure 7. Loop-back probability for a loop region

Notice that, in the above computation of standard deviations, some approximation may occur when we normalize the average profile to an initial profile, for determining the frequencies of the duplicated region/loop head blocks. Our experience indicates that most head blocks are not duplicated and therefore the approximation does not occur often. Furthermore, the comparison between the training profile and the average profile does not involve any approximation. As the result, the approximation would bias against the comparison between the initial profile and the training profile. Since this paper argues in favor of the initial profile, the bias against the initial profile should not be an issue.

4. Experimental results

We report our experimental results using IA32EL for the SPEC2000 INT and FP benchmarks, running on a 900MHz Itanium2 machine with Microsoft Windows operating system. INIP(T) is run with retranslation thresholds T = 100, 200, 500, 1k, 2K, 5K, 10K, 20K, 40K, 80K, 160K, 1M and 4M. Both INIP(T) and AVEP are run with the reference input (or the last reference input for benchmarks with multiple reference inputs). INIP(train) is obtained running with the training input.

After the information for INIP(T), INIP(train) and AVEP are collected into files, we use an off-line tool to analyze the data and calculate the standard deviations. The analysis tool uses the solver for system of linear equations in the Intel's Math Kernel Library [13] to propagate block frequencies for the duplicated blocks in NAVEP and compute CP and LP for regions in INIP(T) and AVEP.

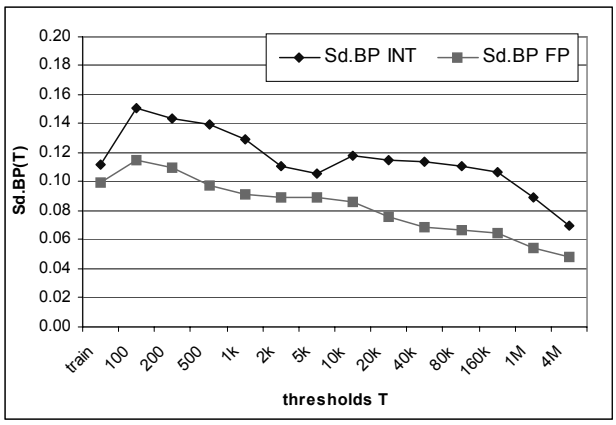

Figure 8. Standard deviations of branch probabilities

4.1. Accuracy of Branch Probabilities

Figure 8 shows the standard deviations of branch probabilities with the training input (Sd.BP(train)) and with the reference input for different retranslation thresholds (Sd.BP(T)). The upper line is for the average over the 12 INT benchmarks. Although the initial prediction of branch probabilities with retranslation thresholds of 100 to 1k are less accurate than the training input, the initial prediction with retranslation thresholds 2k or higher is similar or more accurate than the prediction by the training input. The increase of Sd.BT(T) when T goes from 5k to 10k is due to a phase change in Mcf program (we will discuss a little more about Mcf when we show the results for individual benchmarks).

The lower line is for the averages over the 14 FP benchmarks. The initial prediction is better than the prediction by the training input when the retranslation threshold is 500 or higher. This supports the conclusion that programs running under IA32EL with a relatively small retranslation threshold should be able perform reasonably well comparing to the static binaries compiled by the static compiler with profile-guided optimization using the training input [1].

Individual benchmarks, however, show significantly higher or lower accuracy with the initial prediction than the prediction by the training input. Figure 9 shows Sd.BP(T) for INT benchmarks. For Perlbmk, Twolf, Bzip2, and Eon, the initial prediction with retranslation threshold of 100 or higher is more accurate than the training input. In particular, the initial prediction for Perlbmk is significantly better than the training input. For several other benchmarks, the initial prediction is less accurate than the training input. Noticeably, Mcf and Gzip have much worse prediction accuracy than the training input, even with the retranslation threshold as high as 4M. Also, the Mcf benchmark seems to have phase changes between thresholds 5K and 10K and also between 160K and 4M, and therefore the initial prediction could not predict its branch probability accurately. Furthermore, half of the benchmarks have Sd.BP(T) higher than 0.1, even with a retranslation threshold up to 160K. For these benchmarks, phase awareness with more profiling phases may improve the accuracy.

Compiler optimizations often use a branch probability threshold for identifying likely taken branches to form regions, e.g. the "minimum branch probability" of 70% in

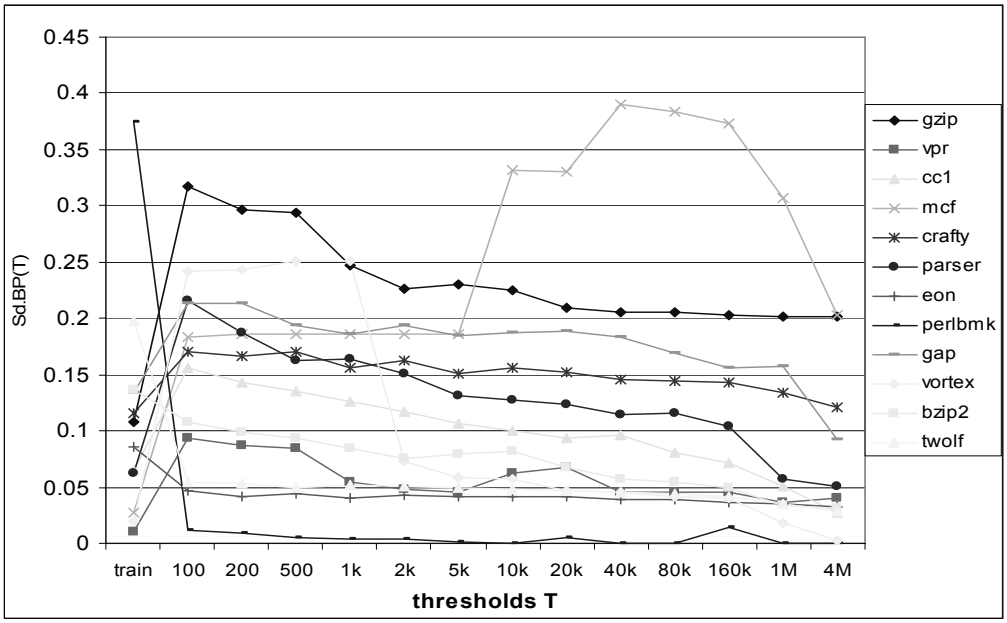

Figure 9. Standard deviations of branch probabilities for SPEC2000 INT

[5]. A small difference between branch probabilities of INIP(T) and AVEP could be significant if the branch probabilities are on different sides of the threshold. For example, the difference between 68% in INIP(T) and 78% in AVEP may cause the optimizer to classify the branch as not likely to be taken (since 68% of taken probability in INIP(T) is less than the threshold), even though the branch is frequently taken in AVEP. On the other hand, the difference between 72% in INIP(T) and 99% in AVEP would not affect the optimization decision at all. To see how the branch probabilities may affect optimizations, we use the following three ranges, [0-.3), [.3-.7], and (.7-1.0], of the branch probabilities to compare the predictions. Namely, a branch probability in INIP(T) "matches" the corresponding probability in AVEP if and only if their values are in the same range. For example, we may consider 0.99 and 0.76 a match, while considering 0.68 and 0.78 a mismatch.

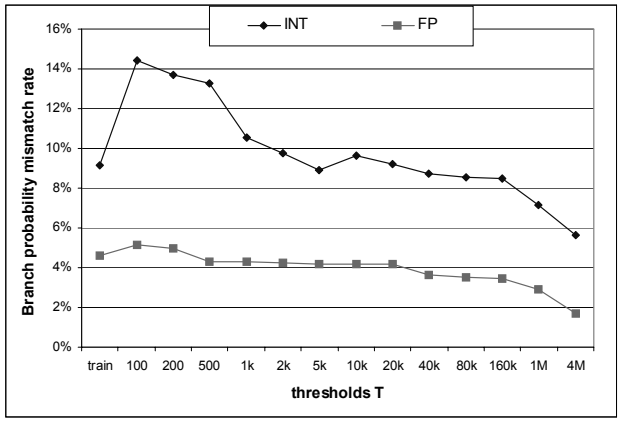

Figure 10. Branch probability mismatch rates

Figure 10 shows the "mismatch" rate (weighted average with block frequencies in AVEP) of the branch probabilities in INIP(T) and those in AVEP. For INT benchmarks, the branch probabilities predicted with the training input matches those in AVEP reasonably well (mismatches only about 9%). We need a threshold of 2k or above to achieve the similar or better matching score. For FP benchmarks, a retranslation threshold of 500 is sufficient to bring out prediction accuracy similar to that by the training input. Overall, FP benchmarks are much easier to predict accurately than the INT benchmarks.

Figure 11 shows the mismatches for individual INT benchmarks. The training input predicts for Perlbmk very badly, with a mismatch rate of about 50%. For Mcf, INIP(T) matches AVEP very poorly, with a mismatch rate of above 30% for most of the thresholds. For Crafty, INIP(T) mismatches AVEP for about 18% of the branches. Gzip is an interesting benchmark. The mismatch rates are high (above 40%) with retranslation threshold at 500 or below, and then drop sharply to about 22% for retranslation thresholds of 1k and above. For the rest of the benchmarks, INIP(T) matches AVEP reasonably well. Notice that most of the lines are relatively flat, except those for Gzip, Gap, and Parser. This indicates that increasing the retranslation threshold dramatically to a large value would only significantly increase the profile accuracy for a few benchmarks.

Figure 12 shows the mismatches for individual FP benchmarks. The training input predicts Lucas and Apsi poorly, with a mismatch rate of 25% and 20% respectively. INIP(T) predicts Wupwise poorly, with a mismatch rate of 20% until the retranslation threshold reaches 1M. For rest of the benchmarks, INIP(T) matches AVEP reasonably well (with mismatch rates less than 10%).

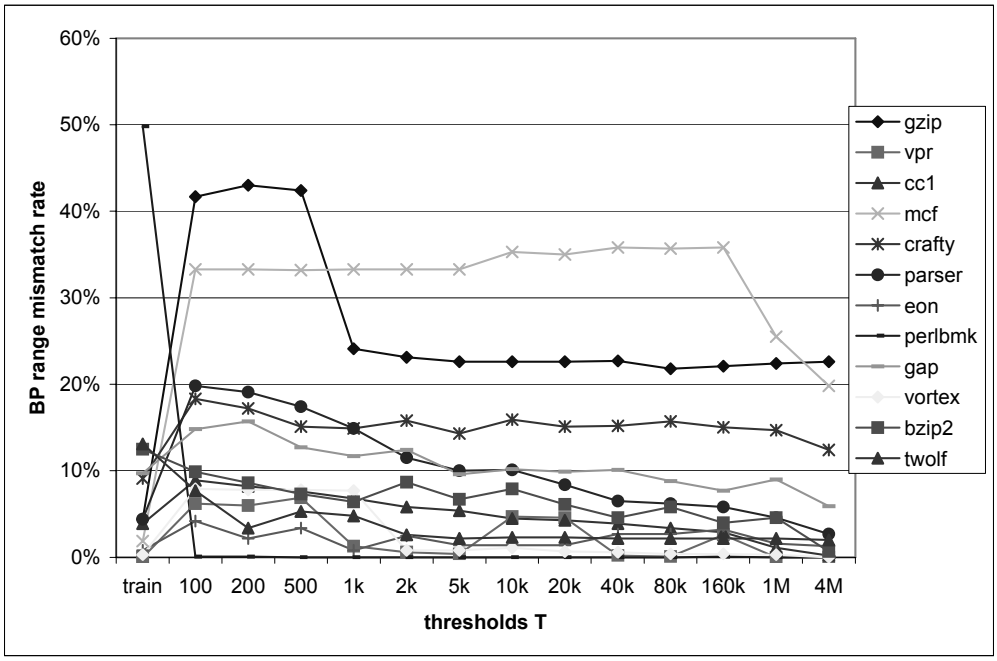

Figure 11. Branch probability mismatch rates (INT benchmarks)

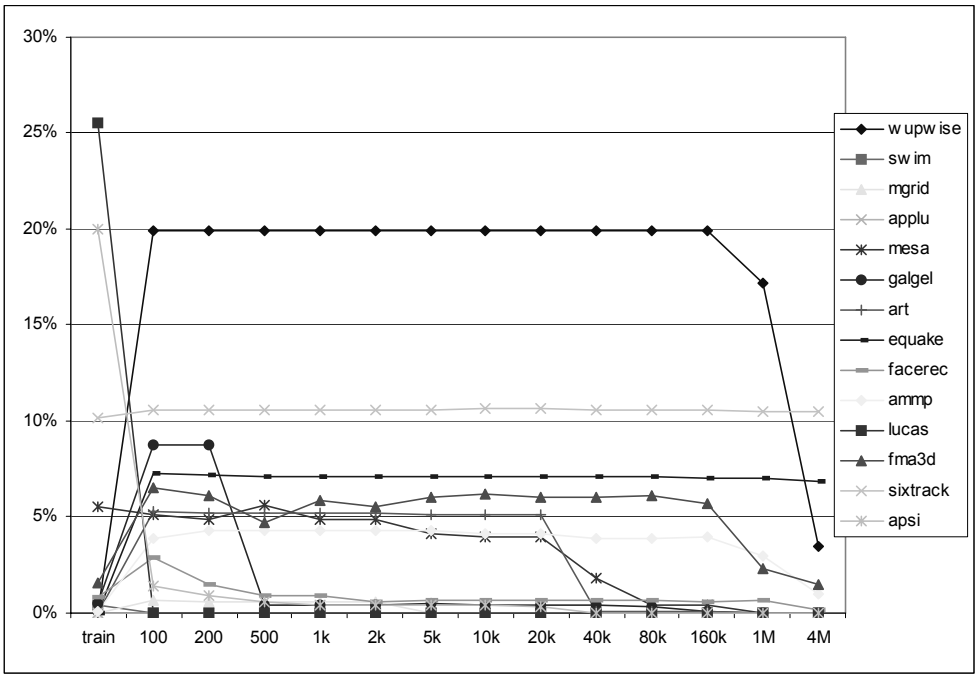

Figure 12. Branch probability mismatch rates (FP benchmarks)

4.2. Accuracy of Completion Probabilities

Figure 13 shows the standard deviation of completion probabilities (Sd.CP(T)). For INT benchmarks, the average Sd.CP(T) is higher than the average Sd.BP(T) for INT shown in Figure 8. This underlines the fact that completion probabilities are much harder to predict accurately than branch probabilities for control intensive INT programs. Even when the branch probabilities of most of the blocks in a region are predicted correctly, a single mis-predicted branch in the region may change the region's completion probability dramatically. A solution would be to continuously monitor the side exits of each region and re-optimize the region when its completion probability changes significantly. Although the average Sd.CP(T) for FP is lower than the average Sd.BP(T) for FP shown in Figure 8, most of the FP benchmarks are loop intensive and only a few easily predictable non-loop regions are formed.

4.3. Accuracy of Loop-back Probabilities

Figure 14 shows the standard deviation of loop-back probabilities (Sd.LP(T)). As expected, the average Sd.LP(T) for FP is noticeably higher than the average Sd.BP(T) for FP shown in Figure 8. Figure 14 also shows that the Sd.LP(T) for FP steadily decreases as retranslation thresholds increases. This suggests that longer profiling period may help loop optimizations.

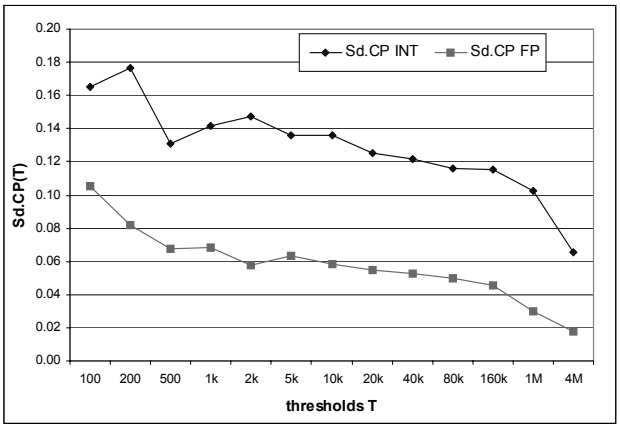

Figure 13. Standard deviation of completion probabilities

Compiler optimizations usually rely on trip count information to determine their applicability. We may classify loops (single or multiple levels) into the following three groups based on their trip count ranges, and consider a prediction a match if and only if the predicted loop trip count in INIP(T) is in the same range as that in AVEP.

Low trip count loops: the loops with trip count less than 10. These loops may be candidates for loop peeling optimizations but neither software pipelining nor data prefetching may be applied profitably. Since LP = (T-1)/T (see [20]), where T is the loop trip count, loops with LP in the range of [0..0.9) are low trip count loops.

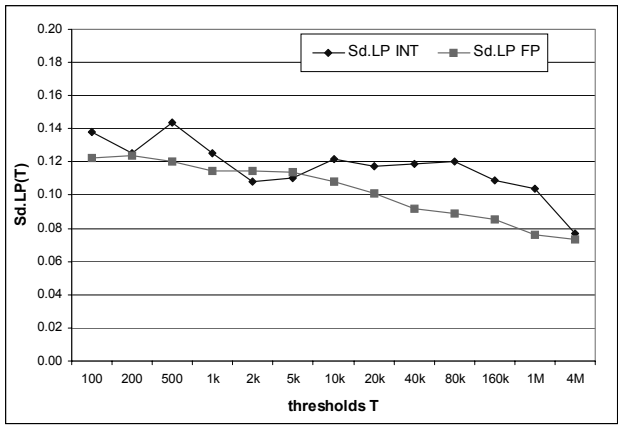

Figure 14. Standard deviation of loop-back probabilities

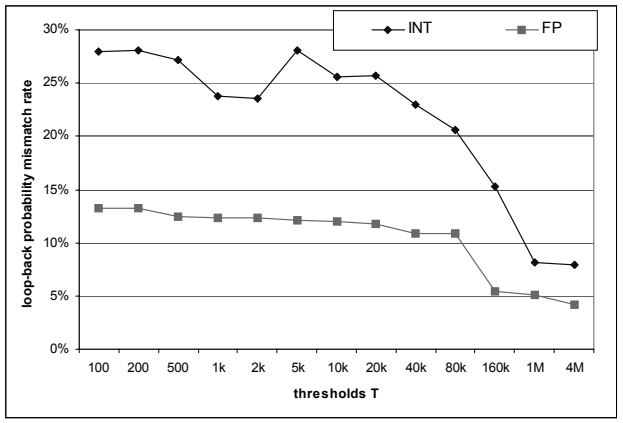

Figure 15. Loop-back probability mismatch rate

Median trip count loops: the loops with trip count between 10 and 50. These loops are possible candidates for software pipelining but not for data prefetching. Loops with LP in the range of [0.9..0.98] are median trip count loops.

High trip count loops: the loops with trip count more than 50. These loops are good candidates for both software pipelining and data prefetching. Loops with LP in the range of (0.98..1.0) are high trip count loops.

Figure 15 shows the average "mismatch" rate (weighted by the loop entry block frequencies in AVEP) between loop-back probability in INIP(T) and that in AVEP. For INT benchmarks, the initial prediction does not predict the ranges of the loop trip count accurately until the retranslation threshold reaches a very high value (e.g. 160k). Thus we will need additional mechanisms to obtain accurate trip count information for advanced loop optimizations. One way is to continuously collect trip count information in the optimized code and adjust loop optimization when trip count characteristic changes. The profiling code for collecting trip count information should not impact the performance of the optimized code as lightweight instrumentation is possible. For FP benchmarks, the initial prediction predicts the ranges of the loop trip count accurately even with a low retranslation threshold of 100.

Figure 16 shows the "mismatch" rate with loop-back probabilities in INIP(T) and that in AVEP for individual SPEC2000 INT benchmarks. The loop classification for Mcf is completely incorrect until the retranslation threshold reaches 10k and above. Our experiment with data prefetching confirms this observation: the high trip count loops identified in Mcf by the initial profile with small retranslation thresholds turns out to be low trip count loops late in the program's execution and the loops having actual high trip counts usually have low trip counts during the initial execution. The classification for Vpr and Cc1 is incorrect more than 50% of the time until the retranslation threshold reaches 80k and above.

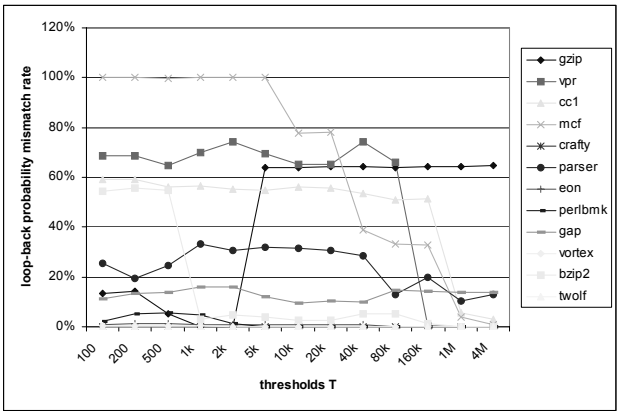

Figure 16. Loop-back probability mismatch rate (INT benchmarks)

4.4. Performance Impact of Initial Profiles

Although our study correlates retranslation thresholds with prediction accuracy, the prediction accuracy alone may not be sufficient to determine the performance of the programs using the profile. Other factors, such as the ILP available in the code, the cost of optimization, and the length of execution of the optimized code, etc, may also impact the ultimate performance.

Figure 17 shows the relative performance of SPEC2000 running with the last ref input for different retranslation thresholds (higher is better). The base performance is obtained the run with the retranslation threshold of 1 (i.e. to optimize all blocks that are executed at least once). For the SPEC2000 INT benchmarks (see the line marked with int), the best performance is obtained when the retranslation threshold is around 1k to 5k, which out-performs the base by nearly 18%. The Perlbmk benchmark has the most dramatic performance improvement with accurate initial profile. The line marked with "int no perl" is the relative performance of

the SPEC2000 INT without the Perlbmk benchmark. In this case, the best performance is obtained around the thresholds of 1k to 5k, which out-performs the base by about 7%

For the FP benchmarks, the best performance is obtained when the retranslation threshold is at 50, 200, or 2k, which out-performs the base by about 2%.

Although the initial prediction with a high retranslation threshold (e.g. 1M and above) is very accurate, the corresponding performance is significantly worse than a less accurate initial profile (e.g. with a retranslation threshold of 2k). This indicates that in a two-phase dynamic binary translator like IA32EL, it is important to optimize a program early, even with less accurate initial profile, to benefit from the optimized execution.

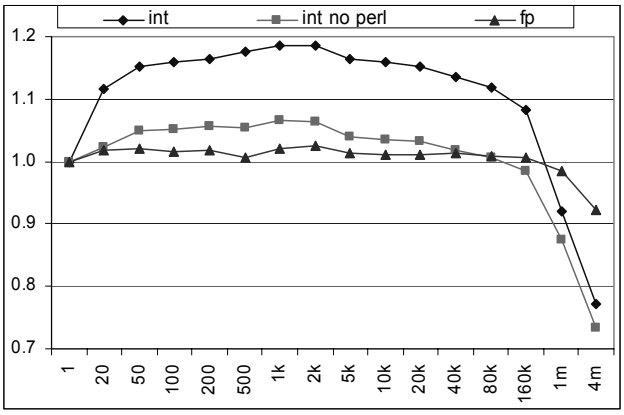

Figure 17. Performance impact of initial profiles

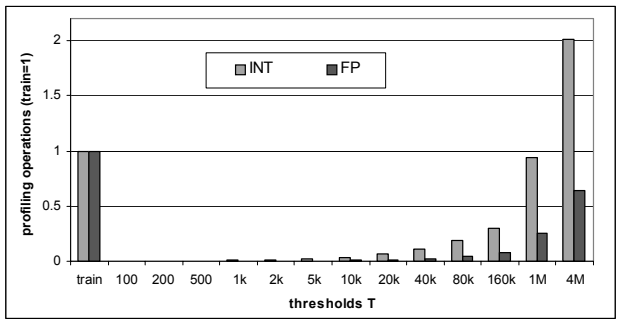

Figure 18. Profiling operations required for training run and for initial profiles

4.5. Overhead of Initial Profiling

Figure 18 shows the total number of profiling operations (sum of all "use" and "taken" count values) for the training run and for the initial profiles (normalized such that training run has a value of 1). The initial prediction with the thresholds of 500 to 2000 takes less than 1% of the profiling operations required for the training run. The training input takes the similar number of profiling operations as the initial profile with a retranslation threshold of more than 1M. Clearly, the dynamic binary translator can obtain similar or more accurate profile with a much shorter profiling period than the static compiler with the training input, due to the benefit of using the early period of same input to predict the future behavior.

5. Summary and Future Work

Our study indicates that the initial prediction with 500 to 2k retranslation thresholds has the accuracy comparable to the traditional profile-guided optimizations using the training input. Specifically, for INT benchmarks, the initial profile obtained with a retranslation threshold of 2000 has the similar prediction accuracy to that by the training input, while for FP benchmarks, the initial profile with a retranslation threshold of 500 is sufficient to achieve the similar prediction accuracy by the training input.

Several benchmarks also show phase behavior and the single profiling phase does not capture the average program behavior accurately. For those benchmarks, longer profiling periods or selective continuous profiling (especially for CP and LP) is beneficial to predict the average program behavior more accurately. Effectively monitoring region side exits to trigger retranslation and phase adaptation looks promising. Inexpensive instrumentation for continuous collection of loop trip count for advanced loop optimizations is possible [21]. Furthermore, retranslation thresholds should be chosen selectively, as different benchmarks (e.g. INT vs. FP) react to retranslation thresholds differently.

In the future, we may continue work in the following areas:

- Characterize the mis-predicted branches and regions. It is an interesting subject to develop heuristics so that the branches and regions that cannot be predicted accurately by the initial profile may be selected for continuous profiling.
- Develop heuristics to select retranslation thresholds for different benchmarks or even for different portions of a benchmark. Although our study correlates retranslation thresholds with prediction accuracy, the prediction accuracy alone may not be sufficient to determine the optimal retranslation threshold. Other factors, such as the ILP available in the code, the cost of optimization, and the length of execution of the optimized code, etc, may also affect the optimal retranslation threshold.
- Construct regions in INIP(train) and compute Sd.CP and Sd.LP between INIP(train) and AVEP to compare the initial prediction of CP and LP with the prediction by the training input.

- Extend the study to compare the initial profile with the specialized (duplicated) program profile in addition to the average program profile. One challenge is to produce comparable specialized program profiles with different inputs (ref and train) and different retranslation thresholds.
- Apply the methodology of this study to real world applications. It will be more challenging to study real world applications, as they may have no training inputs for comparison.

6. Acknowledgements

We would like to thank the IA32EL team at Intel for developing the IA32EL infrastructure used in this experiment and providing us with many technical insights about the internals of the dynamic binary translator. We also appreciate the support from the MKL team at Intel for helping us to obtain and use the MKL library.

7. References

[1] Bala, V.; E. Duesterwald, and S. Banerjia. "Transparent Dynamic Optimization." Technical Report HPL-1999-77, Hewlett Packard Labs, June 1999.

[2] Baraz, Leonid; Tevi Devor, Orna Etzion, Shalom Goldenberg, Alex Skaletsky, Yun Wang and Yigal Zemach, "IA-32 Execution Layer: a two-phase dynamic translator designed to support IA-32 applications on Itanium®-based systems," MICRO-36, 2003.

[3] Barnes, R. D.; E. M. Nystrom, M. C. Merten, W.M. W. Hwu, "Compilation and run-time systems: Vacuum packing: extracting hardware-detected program phases for post-link optimization," MICRO-35, Nov. 2002.

[4] Chang, Pohua P.; Scott A. Mahlke, Wen-mei W. Hwu, "Using profile information to assist classic code optimizations," Software—Practice & Experience, v.21 n.12, p.1301-1321, Dec. 1991.

[5] Chang, Pohua P.; Wen-mei W. Hwu, "Trace Selection For Compiling Large C Application Programs To Microcode," MICRO-21, November 30-Dec. 2, 1988.

[6] Dehnert, J.C.; Grant, B.K.; Banning, J.P.; Johnson, R.; Kistler, T.; Klaiber, A.; Mattson, J.; "The Transmeta code morphing software: using speculation, recovery, and adaptive retranslation to address real-life challenges," CGO-2003. Mar 2003, Pages 15 –24.

[7] Ebcioglu, Kemal; and Erik R. Altman. "DAISY: Dynamic Compilation for 100% Architectural Compatibility, ISCA-24, June 1997.

[8] Feller, P. T.; "Value Profiling for Instructions and Memory Locations, "M.S. Thesis CS98-581, University of California at San Diego, April 1998.

[9] Freedman, David; Robert Pisani, Roger Purves, Ani Adhikari, *Statistics*, 2nd Edition, W.W. Norton & Company, New York, 1991.

[10] Grant, Brian; Markus Mock, Matthai Philipose, Craig Chambers, Susan J. Eggers, "The benefits and costs of DyC's run-time optimizations," ACM Transactions on Programming Languages and Systems (TOPLAS), Sept. 2000.

[11] Hank, R.; W. Hwu, and B. Rau. "Region-Based Compilation: An Introduction and Motivation," MICRO-28, pp. 158–168, November 1995.

[12] Hsu, W.C.; H. Chen; P.C. Yew; D.Y. Chen; "On the predictability of program behavior using different input data sets," Interact-6, Feb. 2002.

[13] Intel, http://www.intel.com/software/products/mkl/, Math Kernel Library.

[14] Kistler, Thomas; Michael Franz "Continuous program optimization: A case study." TOPLAS 25(4): 500-548 (2003).

[15] Mahlke, Scott A.; David C. Lin, William Y. Chen, Richard E. Hank, Roger A. Bringmann, "Effective compiler support for predicated execution using the hyperblock," MICRO-25, Dec. 1992

[16] Sherwood, T.; Suleyman Sair, and Brad Calder, "Phase Tracking and Prediction" ISCA-30, June 2003.

[17] Smith, Michael D. "Overcoming the Challenges to Feedback-Directed Optimization," Proc. ACM SIGPLAN Workshop on Dynamic and Adaptive Compilation and Optimization, invited lecture, Boston, MA, Jan. 18, 2000.

[18] Wagner, Tim A., Vance Maverick, Susan L. Graham, Michael A. Harrison, "Accurate static estimators for program optimization," PLDI'94, June 1994.

[19] Wall, D. W. "Predicting program behavior using real or estimated profiles," PLDI'91, May 1991, ACM SIGPLAN Notices, Volume 26 Issue 6.

[20] Wu, Y., J. Larus, "Static Branch Frequency and Program Profile Analysis," MICRO-27. Nov. 1994.

[21] Wu, Y.; M. Breternitz, T. Devor, "Continuous Trip Count Profiling for Loop Optimizations in Two-phase Dynamic Binary Translators," Interact-8, in conjunction with HPCA-10, Feb. 2004.

[22] Young, Cliff, Michael D. Smith, "Improving the accuracy of static branch prediction using branch correlation," ASPLOS-6, Volume 29, 28 Issue 11, 5, Nov. 1994