A Detailed Study on Phase Predictors

Frederik Vandeputte, Lieven Eeckhout, and Koen De Bosschere

Ghent University, Electronics and Information Systems Department
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
{fgvdeput,leeckhou,kdb}@elis.UGent.be

Abstract. Most programs are repetitive, meaning that some parts of a program are executed more than once. As a result, a number of phases can be extracted in which each phase exhibits similar behavior. These phases can then be exploited for various purposes such as hardware adaptation for energy efficiency. Temporal phase classification schemes divide the execution of a program into consecutive (fixed-length) intervals. Intervals showing similar behavior are grouped into a phase. When a temporal scheme is used in an on-line system, phase predictors are necessary to predict when the next phase transition will occur and what the next phase will be. In this paper, we analyze and compare a number of existing state-of-the-art phase predictors using the SPEC CPU2000 benchmarks. The design space we explore is huge. We conclude that the 2-level burst predictor with confidence and conditional update is today's most accurate phase predictor within reasonable hardware budgets.

1 Introduction

A computer program execution typically consists of several phases of execution where each phase exhibits its own behavior. Being aware of this large-scale time-varying behavior is key to understanding the behavior of the program as a whole. Phase behavior can be exploited for various purposes, ranging from performance modeling [1], compiler optimizations [2], hardware adaptation [3][4][5][6][7], etc. For example in phase-based hardware adaptation, if we know that particular parts of the processor are unused during some program phase, we can turn off those parts during that phase resulting in a reduced energy consumption without affecting overall performance.

One way of identifying phases is to divide the complete program execution into fixed-length instruction intervals and to group instruction intervals based on the code that is being executed during those intervals—this is often referred to as the temporal approach [1]. This means that intervals that execute the same code will be grouped together in what will be called a phase. When this phase classification scheme is used in a phase-based optimization system, it is important to be able to predict when the next phase transition will occur and what the next phase will be. In other words, the phase predictor needs to predict what the phase ID of the next execution interval will be. This way, the system can proactively respond to predicted phase changes.

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 571–581, 2005.

The first contribution of this paper is to study today's state-of-the-art phase predictors in detail. The design space we explore is huge as we explore a large number of possible design parameters: the phase predictor's type, its size, its associativity, its confidence mechanism, its update mechanism, etc. We do this for two fixed-length intervals lengths, namely 1M and 8M intervals, using the complete SPEC CPU2000 benchmark suite. Our second contribution is that we improve the performance of existing phase predictor schemes by up to 14% by adding conditional update. We conclude that the 2-level burst predictor with confidence and conditional update is today's most accurate phase predictor for reasonable hardware budgets.

2 Previous Work

Duesterwald et al. [8] identify program execution phases based on hardware metrics such as CPI, miss rates, etc. They also evaluate a collection of (statistical and table-based) predictors to predict the behavior of the next phase. There is a subtle but important difference between the predictors studied by Duesterwald et al. and the phase predictors studied in this paper. Phase predictors predict the next phase ID; the predictors studied by Duesterwald et al. predict the hardware characteristics of the next phase.

Sherwood et al. [6] propose a dynamic phase classification method that is based on the code that is being executed during a fixed-length interval of execution. Per interval of execution they compute a code signature which is a hashed bitvector that keeps track of the basic blocks that are being executed. Sherwood et al. also present and evaluate several phase predictors, namely the last phase predictor, the RLE predictor and the Markov predictor.

In a follow-on study, Lau et al. [9] added confidence counters to the phase predictors to improve their accuracy. Confidence counters are n-bit saturating counters which are incremented or decremented on a correct or wrong prediction, respectively. When the confidence counter exceeds a given threshold the phase predictor is followed; if not, the default last phase prediction is taken. In their study, they also made a distinction between phase change prediction – predicting the next phase ID – and phase length prediction – predicting the length of the next phase using run length classes.

In [7], Vandeputte et al. propose an offline phase analysis methode that is capable of efficiently dealing with multi-configuration hardware where a large number of hardware units can be configured adaptively. This offline phase analysis technique determines the phases based on a fused metric that incorporates both phase predictability and phase homogeneity.

3 Methodology

We performed our analyses using the complete SPEC CPU2000 benchmark suite. The binaries were taken from the SimpleScalar website. For all our results, phase classification is done offline by profiling the program using a train

Table 1. The number of phases extracted for the SPEC2000 benchmark suite using 1M and 8M instruction intervals.

	# Phases			# Phases			# Phases	
Benchmark	1M	8M	Benchmark	1M	8M	Benchmark	1M	8M
bzip2	16	16	twolf	6	4	fma3d	2	2
crafty	2	2	vortex	2	2	galgel	9	11
eon	2	3	vpr	6	6	lucas	2	2
gap	25	10	ammp	12	13	mesa	6	13
gcc	31	27	applu	32	32	mgrid	4	18
gzip	19	13	apsi	7	2	sixtrack	6	4
mcf	10	10	art	6	3	swim	26	11
parser	10	4	equake	11	10	wupwise	9	8
perlbmk	2	2	facerec	28	16			

input—we refer to [7] how this is done. All our profiles were collected with SimpleScalar/Alpha [10]. For the phase classification, 1 million and 8 million instruction intervals are used[1]. Once the phases of the program using this training input are determined, we determine the phase sequence of each benchmark while executing the reference input; this is done by assigning a phase ID to each execution interval based on the code that is being executed. The various phase prediction schemes are then evaluated on these reference phase sequences. Table 1 shows the number of phases for the 1M and the 8M instruction intervals for all the SPEC CPU2000 benchmarks. Note that the number of unique phase IDs is fairly small here compared to [1][9]. The reason is that our offline phase analysis approach [7] balances phase predictability and phase homogeneity, whereas in [1][9], the main objective is phase homogeneity.

4 Phase Prediction

In this section, we will discuss a number of existing phase predictors. As mentioned before, the purpose of a phase predictor is to predict when the phase change will happen and to what phase the program will shift. In fact, a phase predictor predicts the phase ID of the next execution interval based on the history of phase IDs seen in recent history. The conception of these phase predictors is based on the observation that many phases tend to be stable for several consecutive execution intervals and that there exist both regular and irregular phase behavior. The predictors presented here exploit this notion. Before detailing the various phase predictors that we explore in this paper, we first want to define a *phase burst* to be a number of *consecutive* intervals belonging to the same phase, i.e. all intervals in a phase burst have the same phase ID.

4.1 Basic Phase Predictors

In this subsection we describe a number of basic phase predictors. In subsection 4.2, we will add extra features to these predictors to further increase the prediction accuracy.

[1] Actually, each interval consists of 2^{20} and 2^{23} instructions, respectively.

Last Value Predictor. The simplest predictor is the last value predictor which predicts that the phase ID of the next interval will be the same as the phase ID of the current interval. This predictor assumes that a phase burst is infinite; the predictor thus never predicts a phase change. As a result, if the average burst length is ℓ, the misprediction rate using the last value predictor is $1/\ell$. For many benchmarks this predictor performs very well. This is because these benchmarks have a rather large average burst length. For example, if the average burst length is 20, the misprediction rate for the last value predictor is only 5%.

N-Level Burst Predictor. The last value predictor gives good results in case the average burst length is large. However, if there are frequent phase changes, the misprediction rate will become very high. For example, if the average burst length is only 2, the misprediction rate of the last value predictor increases to 50%, meaning that there is a misprediction every other interval. For frequently changing phase behavior, we thus need more advanced phase predictors.

The N-level burst predictor as proposed in [6][9] uses the phase IDs of the last N phase bursts (including the current phase ID) as the history information for indexing the prediction table. This history information is hashed and the table is accessed using the lower order bits of the hash. The higher order bits are used as tag in the prediction table.

Each entry in the prediction table stores a burst length ℓ and the next phase burst ID k. This means that the current phase burst will last for ℓ execution intervals and that the following phase burst will be of phase k. In other words, the burst predictor will predict a phase change to phase k after being ℓ intervals in the current phase.

On a phase change, the entry in the prediction table is updated by writing the effective burst length and the next phase ID after the phase change. Obviously, the burst history is also updated.

N-Level RLE Predictor. Another predictor, similar to the N-level Burst Predictor is the N-level RLE Predictor [6]. The N-level RLE predictor uses the N most recent (Phase ID, burst length) pairs as history information for indexing the table. Notice the difference with the burst predictor—the N-level burst predictor only uses the phase IDs of the N most recent phase bursts, not their corresponding burst lengths. This RLE history information is hashed together of which the lower order bits are used to index the prediction table. The higher order bits are used as a tag to determine if there is a tag match. If there is a match, the phase ID stored in the table is used as phase ID for the next interval, i.e. we predict a phase change. If there is no match, the current phase ID is used as a prediction, i.e. we predict no phase change. The predictor table is updated if the actual next phase ID differs from the next phase ID stored in the phase table. A new entry is inserted if there was a phase change but no tag match. An existing entry in the phase table is removed in case it predicted a phase change when there was none.

A possible disadvantage of this scheme over the N-level burst prediction scheme might be that there is now too many history information to be hashed

together for indexing the prediction table. Also, if the pattern of the burst lengths is not very regular, the learning time of all the different combinations of phase IDs with different burst lengths might increase drastically. This will be further discussed when evaluating the different phase predictors.

4.2 Phase Predictor Improvements

The N-level burst and RLE predictor are beneficial in case the phase change pattern is regular. However, if the pattern is rather irregular, predicting phase changes might be difficult using the standard burst and RLE phase predictors. Indeed, in order for a phase change prediction to be correct, both the burst length as well as the next phase ID must be predicted correctly. Mispredicting one of these can result in significant performance degradation or missed optimization opportunities. In some cases, the total number of mispredictions for the burst and RLE predictor might even end up to be larger than the last value predictor. Therefore, a number of improvements have been proposed to enhance the accuracy of these phase predictors.

Confidence. One way to reduce this amount of incorrectly predicted phase changes is to add confidence counters to each entry in the phase table [9], and to only make a phase change prediction in cases we are confident that the prediction will be correct. In all other cases, we predict no phase change. The idea is to verify a number of predictions before accepting them. In other words, only when the confidence counter exceeds a given threshold, a phase change prediction is made. When the confidence counter is lower than the given threshold, the last value prediction is taken. The rationale behind this approach is that it is better not to predict a phase change than to predict an incorrect phase change, since an incorrect phase change may initiate an expensive hardware optimization.

Conditional Update. Until now, we described phase predictors in which the information in the prediction table is updated immediately in case of a phase change misprediction. However, for phase bursts with a regular pattern, it might be useful not to change the prediction information immediately, but to wait and see if the irregularity was not just caused by noise.

This can be accomplished by implementing conditional update. This is done by adding saturating counters, which are updated in the same way as the confidence counters (i.e. plus one on a correct prediction and minus one on an incorrect prediction), and only update the phase table information when the corresponding saturating counter is zero. This way, stable phase information is not changed until we are sure that the misprediction was not caused by noise.

Confidence Combined with Conditional Update. Of course, confidence and conditional update can also be used together to take into account both the irregular patterns as well as the noise within the regular patterns. In this case, a common saturating counter is used that is incremented and decremented in

the same manner as described above. With this scheme, a phase change prediction is only accepted if the counter is above some threshold, and the prediction information is only updated if the saturating counter is zero. As will be shown in the next section, this scheme gives the best results on average, both for the N-level burst predictor as well as for the N-level RLE predictor.

5 Evaluation

In the last section, we described the basic design of a number of existing phase predictors and their improvements. For each of these predictors, there remain however many parameters that can be varied and optimized. A list of these parameters and their range is shown in Table 2: the number of levels that can be used for the history information, the total number of entries in the prediction table, the associativity of the table, the number of saturating bits in case conditional update and/or confidence is used, the confidence threshold, the number of tag bits stored in each entry to identify the information that is stored in that entry, and the number of run-length bits to store the predicted burst length in case of the burst predictors[2]. This results in a very large design space that must be explored to obtain the best phase predictor for a given hardware budget. For example, in case of the N-level burst predictor, when all the parameters are varied according to Table 2, about 250,000 different configurations have to be evaluated on the complete SPEC2000 benchmark suite.

Table 2. The range of each parameter we varied for the different predictors.

Predictor Type	Levels	Entries	Assoc.	Sat. Bits	Confid. Thresh.	Tag Bits	Run-length Bits
N-Level Burst	1–4	1–4096	1–8	0–3	0–2	0–10	1–12
N-Level RLE	1–4	1–4096	1–8	0–3	0–2	0–10	0

Fig. 1 shows the average phase misprediction rates as a function of the hardware cost for the N-level burst predictor with conditional update and confidence[3] for a varying number of levels of history for 1M and 8M instruction intervals. To make a fair comparison between the predictors, we calculated the pareto-optimal predictor configurations by varying the parameters in Table 2. As can be seen in Fig. 1, adding more levels of history has an influence on the overall prediction accuracy. For small hardware budgets, using too much history information leads to higher misprediction rates because of increased aliasing as more patterns must be stored in the table. However, once the table becomes large enough, the effect of aliasing reduces and using more history becomes beneficial. Of course, using more history information also increases the learning time of the predictor. This is why the 4-level predictor does not perform much better than the 3-level predictor.

[2] In case the last burst length appears to be larger than the maximum burst length that can be stored, zero is used to represent ∞.

[3] We used this type of predictor because this turns out to be the best type of predictor.

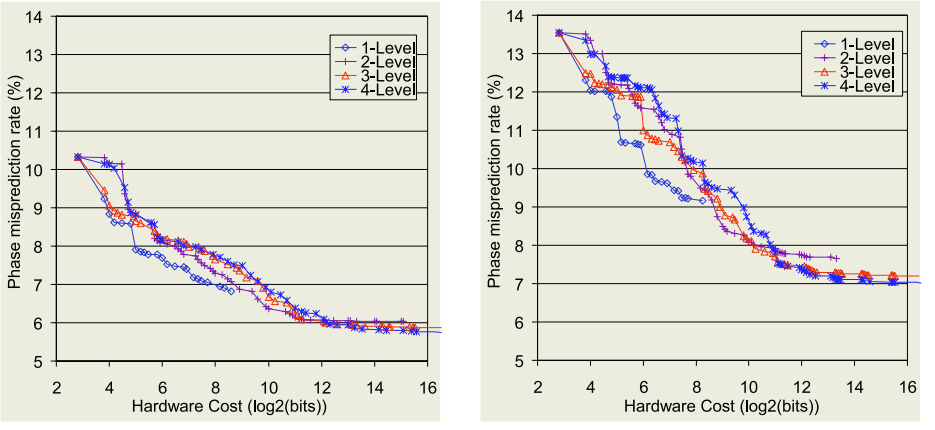

Fig. 1. The average phase misprediction rate of the N-Level burst predictor with conditional update and confidence for different values of N for different hardware budgets. The left graph shows the results for 1M instruction intervals; the right graph for 8M instruction intervals.

Taking into account the misprediction rates and the total hardware costs, one can conclude that an interesting range for the phase predictors is $2^9 \ldots 2^{12}$ bits (i.e. 64 to 512 bytes); smaller predictors result in reduced accuracy and larger predictors result in a larger hardware cost without much gain in predictability. Within this range, the 2-level predictor appears to be the best choice, so we will continue our evaluation with this type of predictor.

Fig. 2 shows the phase misprediction rates of the last value predictor, the 2-level RLE and the 2-level burst predictor (each with conditional update and confidence) for each benchmark for a reasonable hardware budget of 256 bytes. The upper and lower graph show the results for 1M and 8M instruction intervals, respectively. As can be seen, there is a large difference in misprediction rate between the benchmarks and between both interval sizes. For some benchmarks, the misprediction rate is almost zero. This is because these benchmarks have a large average burst length. Fig. 2 also shows that the burst predictor and the RLE predictor do not perform much better than the simple last value predictor for many SPECint benchmarks. For the SPECfp benchmarks however, the reduction of the misprediction rate is quite substantial. The reason for this is that the phase behavior of SPECfp programs is more regular. On average, the 2-level burst predictor and the 2-level RLE predictor can reduce the number of mispredictions by more than 40%. In other words, instead of having a phase misprediction every 7.5 intervals, we now only have a phase misprediction every 14 intervals.

Fig. 3 evaluates the impact of confidence counters and conditional update on the average phase prediction accuracy. For reasonable and large hardware budgets, adding confidence, conditional update or both improves the standard predictor by about 3%, 9% and 17% in case of the burst predictor and 2%, 7% and 13% in case of the RLE predictor. Notice that for large hardware budgets

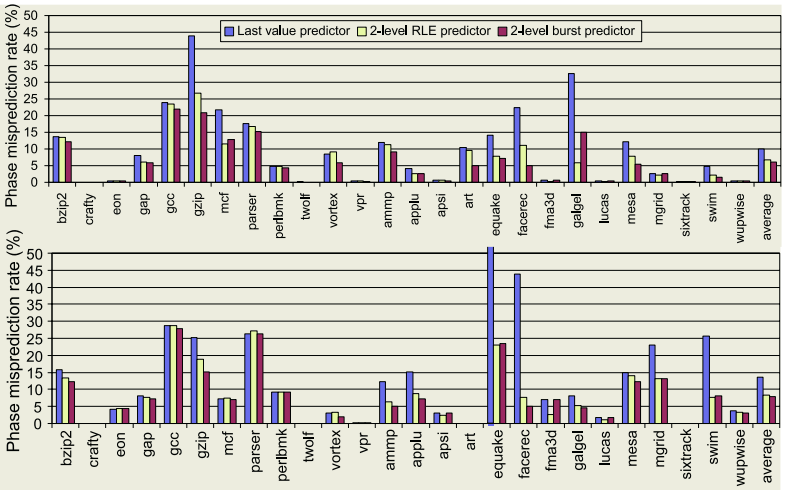

Fig. 2. The phase misprediction rate of the last value predictor, the 2-level RLE and the 2-level burst predictor (each with conditional update and confidence) for a hardware budget of 256 bytes. The upper graph shows the results for 1M instruction intervals; the lower graph for 8M instruction intervals.

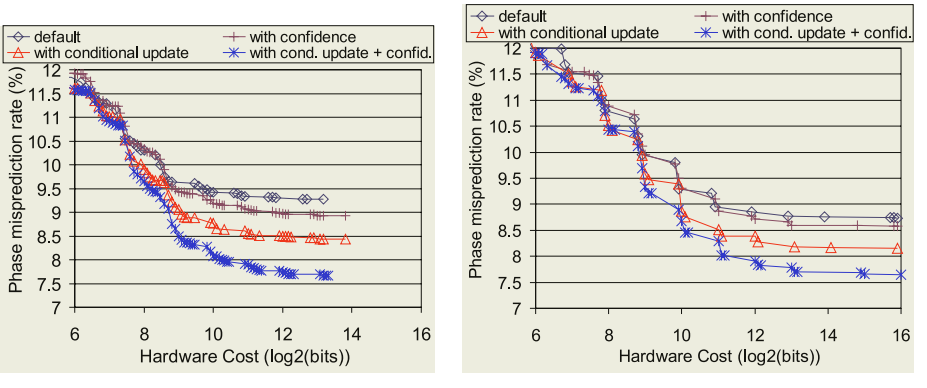

Fig. 3. The effect of applying the different versions of the 2-level burst (left graph) and the 2-level RLE predictor (right graph) on the phase misprediction rate for different hardware budgets. The results shown are for 8M instruction intervals.

the burst predictor and the RLE predictor perform equally well, whereas for smaller hardware budgets the burst predictor performs better. This is because of increased aliasing in case of the RLE predictor, as more history information is used. From these data, we conclude that adding conditional update outperforms the previously proposed predictor schemes by up to 14%.

In Fig. 4, the effect of the number of saturation bits used and the confidence threshold is shown. From this graph, some interesting conclusions can be drawn.

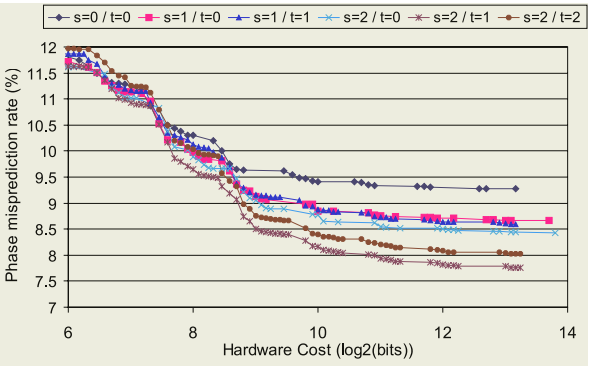

Fig. 4. The effect of the number of saturation bits (s) and the confidence threshold level (t) on the phase misprediction rate for different hardware budgets using a 2-level burst predictor with conditional update and confidence. The results shown are for 8M instruction intervals.

As can be seen, combining conditional update and confidence is only effective when using more than one saturation bit. Also, using a confidence threshold of more than 1 has a negative impact on the prediction accuracy. Using 3 saturation bits (not shown in this graph for clarity) only gives a minor increase in prediction accuracy compared to 2 saturation bits.

Another important aspect is the number of bits b used to encode the history information of the N-level burst and RLE predictor. Using more bits increases the number of bits needed to store the tag for a given table size and associativity.

In case of the N-level burst predictor, the total amount of history information is $p \times N$, where p stands for the number of bits needed for storing the phase ID and N the history depth. These $p \times N$ bits must be mapped onto the available b bits. One way to do this is by using random projection. Another way (which we used in this paper) is to partially overlap the phase IDs by shifting the i-th most recent phase ID over $\lfloor i \frac{b-p}{N-1} \rfloor$ bits ($i = 0 \ldots N-1$), and xor-ing them together.

In case of the N-level RLE predictor, the total amount of history information is $(p+r) \times N$, where r stands for the number of bits used to represent the length of each burst. Mapping this information onto the available b bits is similar to the burst predictor.

In Fig. 5, the effect of varying the number of hashing bits b is depicted. As can be seen, the RLE predictor requires much more bits to be effective than the burst predictor, which is logical, because the former needs to encode much more information. The results shown are only for 2-level predictors; for higher level predictors, the difference is even bigger.

6 Conclusions

Most programs consist of a number of phases in which each phase exhibits similar behavior. These phases can be exploited for various purposes such as performance

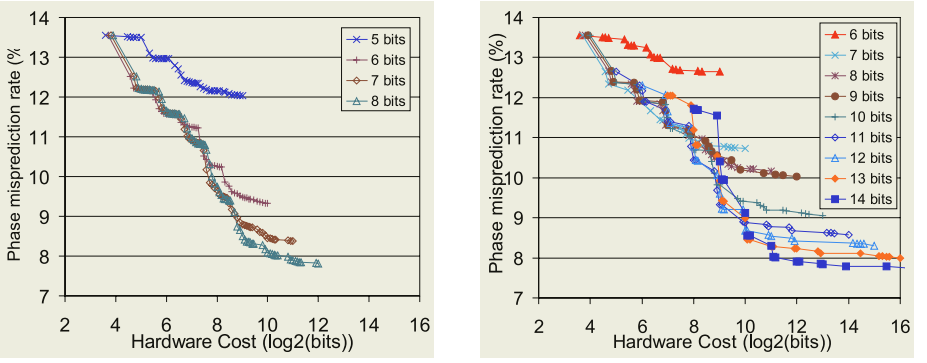

Fig. 5. The effect of the number of bits used to hash the history information used by the 2-level burst (left) and RLE predictor (right) on the phase misprediction rate for different hardware budgets. The results shown are for 8M instruction intervals.

modeling, compiler optimizations, hardware adaptation, etc. When phases are identified by dividing the program execution into fixed-length instruction intervals, and these phases are used in a phase-based optimization system, it is important to be able to predict when the next phase transition will occur and what the next phase will be.

In this paper, we studied a number of today's state-of-the-art phase predictors in detail. The design space we explore is huge as we explored a large number of possible design parameters: the phase predictor's type, its size, its associativity, its confidence mechanism, its update mechanism, etc. We did this for two fixed-length intervals lengths, namely 1M and 8M intervals, using the complete SPEC CPU2000 benchmark suite and concluded that on average the phase predictors show a consistent behavior in terms of phase misprediction reduction. Besides this, we also improved existing phase predictor schemes by 14% using conditional update. We conclude that the 2-level burst predictor with confidence and conditional update is today's most accurate phase predictor for reasonable hardware budgets, reducing the misprediction rate over the last value predictor by more than 40%.

Acknowledgements

This research was funded by Ghent University and by the Fund for Scientific Research-Flanders (FWO-Flanders).

References

1. Sherwood, T., Perelman, E., Hamerly, G., Calder, B.: Automatically characterizing large scale program behavior. In: Proc. of the Internat. Conference on Archit. support for program. languages and operating systems. (2002) 45–57

2. Barnes, R., Nystrom, E., Merten, M., Hwu, W.: Vacuum packing: Extracting hardware-detected program phases for post-link optimization. In: Proc. of the Internat. Symp. on Microarchitecture. (2002)
3. Balasubramonian, R., et al.: Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In: Proc. of the Internat. Symposium on Microarchitecture. (2000) 245–257
4. Dhodapkar, A.S., Smith, J.E.: Managing multi-configuration hardware via dynamic working set analysis. In: Proc. of the Internat. Symp. on Computer Arch. (2002)
5. Huang, M.C., Renau, J., Torrellas, J.: Positional adaptation of processors: application to energy reduction. In: Proc. of the Internat. Symposium on Computer Architecture. (2003) 157–168
6. Sherwood, T., Sair, S., Calder, B.: Phase tracking and prediction. In: Proc. of the Internat. Symposium on Computer architecture. (2003) 336–349
7. Vandeputte, F., Eeckhout, L., De Bosschere, K.: Offline phase analysis and optimization for multi-configuration processors. In: Proc. of the 5th SAMOS workshop. (2005)
8. Duesterwald, E., Cascaval, C., Dwarkadas, S.: Characterizing and predicting program behavior and its variability. In: Proc. of the Internat. Conf. on Parallel Architectures and Compilation Techniques. (2003)
9. Lau, J., Schoenmackers, S., Calder, B.: Transition phase classification and prediction. In: Proc. of the Internat. Symp. on High Performance Computer Arch. (2005)
10. Burger, D., Austin, T.M.: The simplescalar tool set, version 2.0. SIGARCH Comput. Archit. News **25** (1997) 13–25