Reliable Address Translation for Instructions

Ismail Kadayif, Bora Ugurlu

Department of Computer Engineering, Canakkale Onsekiz Mart University, Canakkale, Turkey
kadayif@comu.edu.tr, boraugurlu@comu.edu.tr

Abstract

As a result of technology scaling, spatial multi-bit soft errors have been becoming a big concern for SRAM-based storage structures, such as caches, buffers, and register files, in the design of reliable computer systems. Conventional techniques, such as bit interleaving or stronger coding, cannot provide the designers with effective solutions to the problem of reliable address generation in instruction translation lookaside buffers (iTLB) because of high power and/or latency overheads. In this study, we aim to generate reliable address translation for instructions without compromising either on performance or on power consumption. To do so, we propose to use a pair of identical registers storing the last address translation, which are referred to as context frame registers (CFR). As long as the control flow of programs stays in the same page, address translations are supplied by these two registers, instead of the iTLB. Since two CFRs keep the same address translation, spatial multi-bit errors are detected by comparing their contents. If their contents do not match, we obtain the address translation from the iTLB as usual, which uses strong coding for error detection and correction.

1. Introduction and Background

Soft errors (transient errors) are single event upsets and, in concurrent with continuous technology scaling, have been posing a significant concern for reliable computing systems. Their two primary sources are alpha particles released from packaging materials caused by the effect of excessive heat and highly energetic neuron particle strikes from cosmic rays. These particle strikes can change the minimum charge required to flip the state of devices, resulting in wrong outputs in SRAM-based hardware components such as caches and TLBs.

The need to cope with soft errors have been lead to the development of various approaches for reliable system designs. The hardware-level approaches can be roughly classified into three categories: process technology, circuit, and architectural solutions [1]. The solutions relying on the process technology make use of silicon-insulator (SOI) to protect devices against soft errors. Due to their thinner silicon layer, SOI devices collect less amount of charge from radiation, making strikes less likely to flip the state of SRAM cells. In circuit techniques, radiation-resisted devices are designed by adjusting their capacitance and/or supply voltage, raising the level of the minimum charge required to change the value stored in the cells [2]. There are a wide variety of solutions to mitigate the effects of soft errors at architectural level, including integrity checking techniques such as parity check and error correcting codes (ECC) [3], p bit [4], replicating components such as N modular redundancy (NMR), and bit interleaving [5].

While only single-bit soft errors were a major concern for previous technology generations, spatial multi-bit soft errors (a single particle strike can upset multiple neighbouring SRAM cells) have also been becoming a concern for the current technology generation and the next technology generations as well [6]. Compared to single-bit soft errors, handling spatial multi-bit errors introduces significant performance, power and/or area overheads. For example, a popular ECC technique is SECDED (single error correct double error detect) and implemented by maintaining extra 8-bits for protecting each 64-bit entry. While SECDED can correct single-bit errors, it is not appropriate to be applied into frequently accessed processor component such as L1 caches and TLBs since it may put memory references into critical path –causing performance degradation– and may increase power consumption budget as well. The stronger codes, such as DECTED (double error correct, triple error detect) and TECQED (triple error correct, quad error detect) may present the similar concern, even with more severity.

Another solution proposed to handle spatial multi-bit soft errors is interleaved ECC. In this approach, the data bits belonging to different ECC check words are physically interleaved so that at most one data bit belonging to the same ECC check word is affected by soft errors. Thus, with N-way degree of interleaving, spatial multi-bit errors upsetting up to N continuous bits can be tolerated. However, a prior study shows that interleaving approach cannot be applied to L1 caches in practice because of excessive power requirements [5].

In modern processors, TLBs are used to accelerate virtual-to-physical address translations by keeping recent translations. When an address translation is required, the processor first looks the translation up in the TLB. If there is a hit in the TLB, the translation is supplied from there; otherwise, the page table in the RAM is searched for the address translation under the control of the operating system. Since translations are found in the TLB with a great probability, the operating system's involvement is not required most of the time, making fast address translation possible. Like other SRAM-based structures, TLBs are also vulnerable to high energetic particle strikes causing spatial multi-bit errors. If TLBs are not protected against such particle strikes, severe consequences may emerge; for example, for the iTLB the program control flow can change inadvertently, causing erroneous program outputs or even program crashes.

In this study, we propose a mechanism at architectural level for reliable address translations for instructions without compromising on either power consumption or performance. To this end, we use two identical CFRs (context frame registers) to store the last address translation. These two CFRs always store the same translation. As long as the control flow of the program stays in the same page, address translations are supplied from these two CFRs. With our approach, the error check is carried out by comparing the contents of the CFRs. If the execution stays within the same page and the contents of the CFRs conform to each other, it means the address translation can be

safely obtained from the CFRs. It is extremely rare for two different particle strikes to affect exactly the same bit positions in the two CFRs during the small period when the execution stays in the same page, which makes error detection possible. On the other hand, if the contents of the CFRs do not match, it means at least one of the CFRs is affected by soft errors, so we must not trust the translation stored in either of them. In this case, the translation is supplied from the iTLB as usual. Since we assume that the iTLB is protected against soft errors with strong coding such as TECQED, the translation obtained from the iTLB will be always correct. Moreover, because of the infrequent iTLB accesses in our approach, protecting the iTLB against soft errors introduces negligible performance overheads.

The rest of this paper is organized as follows. The structure of the CFRs and their management is introduced in the next section. In Section 3, we explain how reliable address translation can be done by using the CFRs. Our experimental setup is explained in Section 4. We present our experimental results in Section 5. Finally, our concluding remarks are given in Section 6.

2. Structure of CFRs and Their Management

In our design, the two identical CFR registers store the last address translation and are managed by the hardware intelligently. A similar CFR concept was used in several previous studies. It was used in [7] to reduce the power consumption of dTLB (data TLB) and in [8] to mitigate the performance overhead of process variations on instruction fetches. The structure of a CFR in our work is different from the one used in prior studies in that we need to keep only the physical frame number (PFN) and some protection bits (PB) belonging to the current page in CFRs, as depicted in the following.

$$[< PhysicalFrameNumber < ProtectionBits >]$$

The CFRs can be regarded as a part of the context of a process and saved/restored during the context switch, like regular registers and the program counter. Their content is very similar to a page table entry, except that they do not store virtual frame numbers. Whether the address translation should be obtained from the CFRs or the iTLB is decided by hardware intelligently under the control of the software, as explained in detail next; this is why we do not keep the virtual frame number of the current page in the CFRs.

To manage the CFRs intelligently at run-time, instructions are annotated to give a hint to the hardware where the address translation must be obtained. For this aim, we can use some unused bit positions (slots) of instructions in the ISA to encode the source of address translation. Some of the architectures have already been ported to 64-bit platforms, and there are also considerable on-going efforts to port most of the rest of architectures to 64-bit platforms. So, finding such unused bit slots in the ISAs of modern architectures is not a concern.

According to our scheme, there are two distinct sources where the address translations for instructions can be obtained. The first one is the iTLB, and it is accessed only when the control flow of the program leaves the current page. The CFRs constitute the second source of address translations. As long as the program execution stays in the same page, address translations are supplied from CFRs. Since there are two distinct sources of address translations, only one empty bit position in instructions is needed to encode where the address translation for the succeeding instruction must be obtained: 0 indicates the address translation is supplied from the CFRs, and 1 indicates the address translation is obtained from the iTLB. During run-time, at instruction fetch pipe stage the hardware decide the source of address translation for the succeeding instruction based on this particular annotated bit.

For annotation, we make use of a compiler analysis to build the control flow graph (CFG) of the application and encode the source of address translation for the succeeding instruction. As long as two successive instructions are on the same instruction page, the preceding instruction is annotated with 0 to inform the hardware at run-time that the address translation for the successive instruction will be obtained from the CFRs. On the other hand, there are two ways by which the program execution moves from one page to another: branch instructions whose target is in a different page; and two successive instructions straddling a page boundary, that is, one is the last instruction of a page, and the following instruction is the first instruction of the next page (we refer to this as the BOUNDARY case). In the first case, we try to analyse the target of branch instructions. If a branch instruction is analysable, that is, its target address can be determined at static time, and the target of the branch instruction do not cross the page boundary, the corresponding branch instruction is annotated with 0 to tell the hardware to obtain the address translation from the CFRs for the target instruction. Otherwise, the branch instruction is annotated with 1 to tell the hardware to trigger an iTLB lookup for the address translation of the instruction at the branch instruction's target. For the BOUNDARY case, the compiler always annotates the preceding instruction with 1 to trigger an iTLB lookup for the address translation for the succeeding instruction.

3. Reliable Address Translations in a System with CFRs

In this section, we explain how reliable address translation can be done in a system using our scheme. Figure 1 depicts how our scheme works. The key issue of our scheme is that where the address translation will be supplied from for an instruction is encoded into its immediately preceding instruction using an empty bit slot. In Figure 1, this particular bit is denoted by A (annotated bit). Although our scheme is explained here for a scalar microprocessor (a microprocessor that is capable of fetching and issuing only one instruction at a time), our scheme can be extended to be applied to superscalar machines (microprocessors that can fetch and issue more than one instruction at simultaneously) in a natural way. The only requirement for this is that cache blocks be aligned with page boundaries, that is, a cache block stores instructions from only one page. Some compilers meet this requirement by providing special pragmas for this purpose. Since a cache block cannot span two pages, the instructions fetched at once in a superscalar machine will belong to the same page, thereby address translation for them can be supplied from the same resource, either the iTLB or the CFRs.

3.1. Error Detection

In our scheme, error detection during address translations is done as follows. If the address translation is supplied form the CFRs, their contents first compared. Since in our technique we use two identical CFRs, it can be classified into the category of NMR (N modular redundancy) at architectural level. However, in contrast to traditional NMR approaches, such as triple modular redundancy [9], which introduces huge area overheads, the

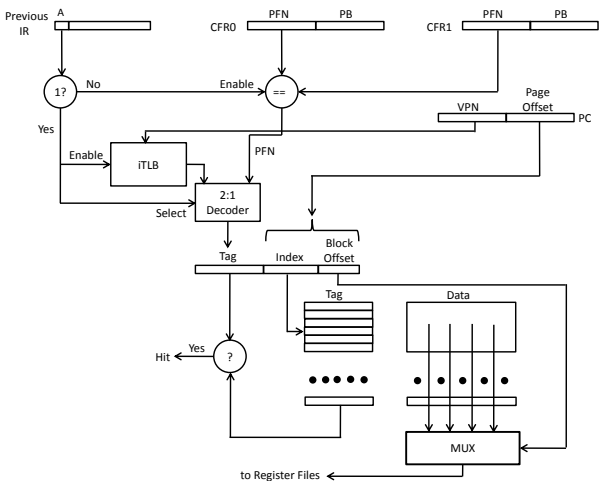

Figure 1. Reliable addres translation for instructions in a system with two CFRs.

area overhead of our technique is negligible, only the two CFRs. As long as the contents of the two CFRs match each other, we assume that we are using reliable address translations. There is very little likelihood that two distinct soft errors corrupt exactly the same bit positions in the two CFRs during a small time period, which makes spatial multi-bit error detection possible. If either the contents of the CFRs are different or the control flow of the program moves to a different page (the A bit is 1), the address translation is done via the iTLB as shown in Figure 1, which provides strong coding for detecting spatial multi-bit errors. Since the frequency of soft errors is rare and the frequency of execution moving one page to another is very low, the performance overhead caused by the error detection mechanism of the iTLB will be negligible, as shown in our experiments next.

3.2. Error Correction

Whenever spatial multi-bit errors are detected in the CFRs, by realizing that their contents are not the same, the iTLB lookup is triggered by the hardware. Even if the entry storing the same address translation in the iTLB is also affected by soft errors, the error correction mechanism of the iTLB can correct the corresponding multi-bit soft error, and then the iTLB and the CFRs are updated with the corrected address translation. Since the error correction mechanism of the iTLB, like the error detection mechanism, is exercised very infrequently, there will be negligible overheads in terms of performance and power consumption as well.

4. Experimental Setup

We evaluated our technique by modifying the SimpleScalar 3.0 simulator [10]. As a commonly used tool set in academic circles, SimpleScalar can simulate application programs on a range of processors and systems using a fast execution-driven simulation, and outputs execution statistics, such as the dynamic number of accesses to components in the memory hierarchy as well as execution cycles. In this study, the sim-outorder component of the SimpleScalar tool set was modified to simulate the integration of our technique into an Alpha-like platform. All compiler analyses regarding extracting the CFG and determining the type of branch instructions as well were done based on pre-compiled Alpha binaries.

Table 1. Major processor configuration parameters and their values used in our experiments.

Processor Core	
Functional Units	4 Integer ALUs, 2 Integer mult./divide, 4 FP add, 2 FP multiply, 2 FP divide/sqrt
RUU size	256 instructions
LSQ size	64 instructions
Fetch/Decode/Issue/Commit width	4 instructions/cycle
Fetch queue size	8 instructions
Cache and Memory Hierarchy	
L1 instruction cache	64KB, 4-way (LRU), 64 byte blocks, three stage pipelined with 3-cycle latency
L1 data cache	64KB, 4-way (LRU), 64 byte blocks, three stage pipelined with 3-cycle latency
L2 cache	8MB unified, 8-way (LRU), 128 byte blocks, 12-cycle latency
Memory	300-cycle latency
Page size	8K
Branch Prediction	
Branch predictor	Combined, Bimodal 4K table, 2-level 2K table, 8-bit history, 4K chooser
Branch target buffer (BTB)	4K-entry, 4-way
Return-address stack	32

Table 2. Benchmarks used in our experiments and their important characteristics.

Benchmark Name	Number of Execution Cycles (in millions)	Number of iTLB Accesses (in millions)
lucas	278.93	236.24
apsi	210.51	189.12
vpr	435.30	307.14
crafty	254.15	150.76
soplex	130.27	109.44
tonto	180.91	107.59
mcf	612.61	548.34
astar	212.18	143.05

In our experiments, we considered a three-stage pipelined cache model which was suggested in [11]. According to this model, set address is decoded in the first stage. Wordline driving, bitline precharging, and monitoring the voltage difference between a pair of bitlines by sense amplifiers take place in the second stage. Driving the output multiplexors and the selected data out of the cache is carried out in the last stage. Since SimpleScalar only simulates nonpipelined caches, we modified its *cache.c* and *cache.h* components to handle pipelined caches.

Table 1 lists the major simulation parameters of our target processor. Our simulated microprocessor is four-issue superscalar machine capable of executing Alpha-like ISA. In our experimental results, we used lucas, apsi, vpr, and crafty applications from the SPEC2000 suite and soplex, tonto, mcf, and astar from the SPEC2006 suite [12]. Since it takes very long time to run any of these applications to completion in the simulation platform taken into consideration, we used the SimPoint Analysis Toolkit [13] to generate some simulation points. The SimPoint Analysis Toolkit is a collection of tools and can be integrated with the SimpleScalar simulator to determine where the simulator should spend its time to get fast, accurate, and representative results. For each benchmark, we fast forwarded a specific number of instructions, as suggested by Sherwood et al. [13], and then simulated the next 500 million instructions on predetermined simulation points.

Table 3. The percentage of CFR updates for each application program with the CFR Scheme.

Benchmark Name	Number of CFRs Updates
lucas	1.67%
apsi	1.05%
vpr	2.80%
crafty	3.87%
soplex	4.14%
tonto	3.07%
mcf	3.48%
astar	5.62%

Table 2 lists the number of execution cycles and the number of iTLB accesses for these benchmarks under the configuration parameters listed in Table 1.

5. Experimental Results

Before presenting our experimental results, we want to explain three experimental setups whose results are under consideration.

- *Base Scheme*: This reflects the case where soft errors do not pose any threat –even single-bit soft errors are not a concern– so the iTLB is not protected against soft errors. This setup corresponds to ideal case. The results given in Table 2 belong to this scheme. We compare the experimental results of other schemes to those of it and assess them accordingly.
- *iTLB with strong coding (iTLB-SC Scheme)*: The iTLB is protected with some error coding technique against soft errors, such as DECTED, TECQED or any other strong coding. Since in each iTLB access the data read out is checked with error detecting codes, one extra cycle delay is incurred. Thus, in this setup we assume that each iTLB access completes in two cycles.
- *CFR Scheme*: It is our proposed scheme in this study. As discussed before, there are two identical CFRs in the hardware, and these two CFRs always keep the same virtual-to-physical address translation, which is the current translation. If the execution stays in the current page, they provide the address translations. Error checking is done by comparing their contents. As long as the execution stays in the same page and their contents match, the address translation takes only one cycle, as in the Base Scheme. Otherwise, the iTLB is accessed and the address translation completes in two cycles, as in the iTLB-SC Scheme. As will be explained next, since program execution moves from one page to another very infrequently and soft errors are very rare events, an address translation takes just one cycle in the common case.

The percentage of CFR updates for each application program with the CFR Scheme is shown in Table 3. These values in the table also correspond to the frequency of program control flow moving from one page to another during the run-time. It is clear that, for each benchmark, the program execution stays in the same page with a great probability.

The performance effects of protecting the iTLB against spatial multi-bit soft errors are indicated in Figure 2. In the iTLB-SC Scheme, due to code checking, each iTLB access takes one extra cycle, completing in two cycles. Delaying address translations for instructions in turn extends the time required for instruction cache accesses, resulting in performance degradation. As can be seen from Figure 2, this performance loss can be as much as 16.2% for the soplex benchmark. The average performance loss caused by checking spatial multi-bit soft errors is around 12.1%, which is not acceptable, particularly for high performance microprocessors. This makes performance-efficient soft error checking mechanisms a necessity for frequently accessed SRAM-based structures like the iTLB.

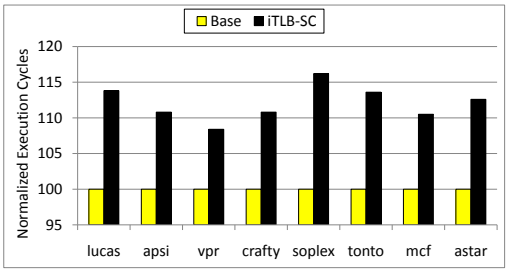

Figure 2. Normalized execution cysles when the iTLB is protected against spatial multi-bit errors with strong error detecting/correcting codes. Each iTLB access takes one extra cycle for checking soft errors.

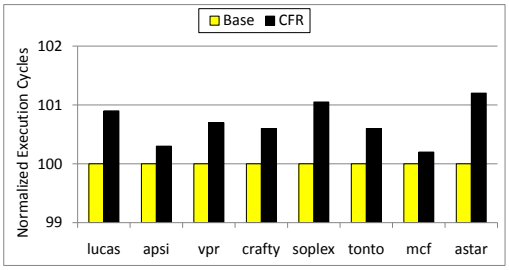

Figure 3. Normalized execution cysles when the iTLB is protected against spatial multi-bit errors when the CFR Scheme is applied. As long as the program execution stays in the sama page, there is no performance loss in address translations.

The performance values of the CFR Scheme are given in Figure 3. All values in the figure are normalized with respect to those of the Base Scheme. When we compare these values with those presented in Figure 2, we can reach the following conclusions. First, if the iTLB is integrated with the proposed CFR Scheme, there will be a huge performance improvement in address translations for any benchmark. For example, the performance losses for lucas is 13.8% and 0.9% when the iTLB-SC Scheme and the CFR Scheme are employed, respectively. Second, when our scheme is used, there are only two application programs (soplex and astar) among the eight tested benchmarks whose performance degradation introduced by checking spatial multi-bit errors is greater than 1%. Third, while the average performance loss with the iTLB-SC scheme is around 12.1%, the average performance loss is around 0.7% when the CFR Scheme is employed.

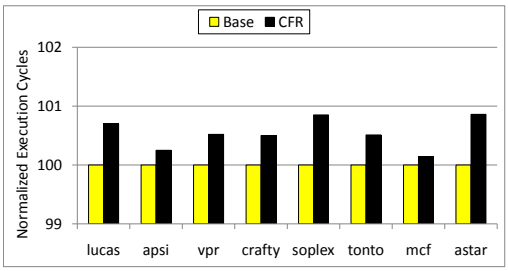

Figure 4. Normalized execution cysles of the CFR Scheme when the page size is raised to 16 KB.

To observe the sensitivity of the performance results of our CFR Scheme to larger page sizes, we repeated some of our experiments for 16 KB page size. Since the page sizes larger than 16 KB are rarely adopted, we carried out the experiments only for 16 KB page size. The normalized performance results of the iTLB-SC Scheme varied negligible, so here we only show the performance results of the CFR Scheme. The results are depicted in Figure 4. The performance of the CFR Scheme, in general, improves for each benchmark when the page size is increased to 16 KB. The reason for this is that the frequency of program control flow moving one instruction page to another decreases with increasing page sizes, which improves the efficiency of the CFR Scheme.

6. Conclusions

High energetic particle strikes can change the charge stored in SRAM cells, causing them to flip their outputs. To make matters worse, the rate of spatial multi-bit errors have been increasing in concurrent with future technology generations. The traditional data protection mechanisms are, in general, do not provide effective solutions to the problem of spatial multi-bit errors, largely because of their performance, area and/or power consumption overheads. This is correct especially for frequently accessed microprocessor units, such as the first level caches and TLBs. In this study, we try to make address translations for instructions reliable without compromising on performance degradation, area overhead, or power consumption. Our proposal consists of having the two CFRs supply address translations under the control of software as long as the program control flow remains in the current page. To this end, the control flow of applications are extracted and an empty bit slot in instructions is used to encode where the next address translation must be obtained, that is, either the CFRs or iTLB. When address translation is obtained from the CFRs, reliable address translation is done by comparing their contents. The reliable address translation is possible since it is extremely rare that two different particle strikes affect the exactly the same bit positions in these two registers. Instruction pages present great locality, so the control flow of programs moves from one page to another infrequently, which allows translations to be supplied from the two CFRs with a great probability. Whenever the program execution moves to another page, which is detected by the hardware by examining the preceding instruction's associated bit, the address translation is obtained from the iTLB. At the same time, the contents of the CFRs are also updated with this translation to allow them to provide address translations in the next. Even if the iTLB is protected against spatial multi-bit soft errors with a strong coding technique, it does not pose any concern from the performance point of view because of huge locality in instruction pages. Our experimental results are quite promising, and indicate that while the average performance loss in a system with the iTLB protected against soft errors with a traditional strong coding is around 12.1%, the average performance loss is less than 1% when the CFRs are employed.

7. References

[1] S.S. Mukherjee, J. Emer, and S.K. Reinhardt, "The soft error problem: an architectural perspective", *Proc. of the International Symposium on High-Performance Computer Architecture*, 2005, pp. 243-247.

[2] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron CMOS technology", *IEEE Transactions on Nuclear Science,* vol. 43, no. 6, pp: 2874-2878, 1996.

[3] D.K. Pradhan, "Fault-tolerant Computer System Design", *Prentice-Hall*, second print, 2003.

[4] C. Weaver, J. Emer, S.S. Mukherjee, and S.K. Reinhardt, "Techniques to reduce the soft error rate of a high performance computer", *Proc. of the International Symposium on Computer Architecture*, 2004, pp. 264-275.

[5] B. T. Gold, M. Ferdman, B. Falsafi, and K. Mai, "Mitigating multi-bit soft errors in L1 caches using last-store prediction", *Proc. of the International Workshop Architectural Support for Gigascale Integration*, 2007, pp. 11-18.

[6] K. Osada, K. Yamaguchi, Y. Saitoh, and T. Kawahara, "SRAM immunity to cosmic-ray-induced multierrors based on analysis of an induced parasitic bipolar effect", *IEEE Journal of Solid-State Circuits,* vol. 39, no. 5, pp: 827-833, May 2004.

[7] I. Kadayif, P. Nanth, M. Kandemir, and A. Sivasubramaniam, "Reducing data TLB power via compiler-directed address generation", *IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems*, vol. 26, no. 2, pp. 312-324, February 2007.

[8] I. Kadayif, M. Turkcan, S. Kiziltepe, and O. Ozturk, "Hardware/software approaches for reducing the process variation effect on instruction fetches", *ACM Transactions on Design Automation of Electronic Systems*, vol. 18, no. 4, October 2013.

[9] C. Carmichael, "Triple modular redundancy design techniques for virtex FPGAs", *Xilinx Application Notes*, XAPP197 (v1.0.1), pp. 1-37, July 2006.

[10] SimpleScalar LLC, http://www.simplescalar.com.

[11] Z. Chishti and T. N. Vijaykumar, "Wire delay is not a problem for SMT (in the near future)", *Proc. of the International Symposium on Computer Architecture*, 2004, pp. 40-51.

[12] SPEC2000 and SPEC2006 Benchmark Suites, http://www.spec.org.

[13] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behaviour and simulation points in applications", *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, 2001, pp. 3-14.