Partially Protected Caches to Reduce Failures due to Soft Errors in Multimedia Applications[1]

Kyoungwoo Lee, Aviral Shrivastava, Ilya Issenin, Nikil Dutt, and Nalini Venkatasubramanian

***Abstract*—With advances in process technology, soft errors are becoming an increasingly critical design concern. Owing to their large area, high density, and low operating voltages, caches are worst hit by soft errors. Based on the observation that in multimedia applications, not all data require the same amount of protection from soft errors, we propose a Partially Protected Cache (PPC) architecture, in which there are two caches, one protected and the other unprotected at the same level of memory hierarchy. We demonstrate that as compared to the existing unprotected cache architectures, PPC architectures can provide 47× reduction in failure rate, at only 1% runtime and 3% power overheads. In addition, the failure rate reduction obtained by PPCs is very sensitive to the PPC cache configuration. Therefore, this observation provides an opportunity for further improvement of the solution by correctly parameterizing the PPC configurations. Consequently, we develop Design Space Exploration (DSE) strategies to discover the best PPC configuration. Our DSE technique can reduce the exploration time by more than 6× as compared to an exhaustive approach.**

***Index Terms*—Memory fault tolerance, Multimedia embedded systems, Partially Protected Cache (PPC), Soft error, Unequal data protection.**

I. Introduction

System reliability is becoming of paramount concern in system design in the deep submicron era [1]. With technology scaling, i.e., smaller feature sizes, reduced voltage level, lower noise margins, etc., future generation of microprocessors will become increasingly prone to soft errors [5]. While soft errors may be caused due to several reasons, radiation-induced faults are responsible for more failures than all the other causes of soft errors combined [1]. A high energy radiation particle, e.g., an alpha particle, neutron, or free proton, may strike the diffusion region of a CMOS transistor and produce charge which can result in toggling the logic value of the gates or flip-flops. This phenomenon of change in the logic state of a transistor is called a *soft error*. Soft errors in memory elements have significantly higher probability of causing a failure than soft errors in combinational logic mainly due to masking effects in combinational logic [9]. In addition, since memory elements may occupy more than majority of the chip area, and the fact that they operate on lower voltages, they are extremely vulnerable to radiations, and radiation induced faults. In fact, according to [10], more than 50% of soft errors happen in memories.

While it is possible to employ simple redundancy based techniques in off-chip memories (e.g., error correction codes or ECC), they are not suitable for caches, which are highly sensitive to performance and power overheads. For example, using Single-bit Error Correction and Double-bit error Detection (SEC-DED) codes may increase the cache access time by 95% [8] and power consumption by 22% [12]. Thus, only a few processors such as the Intel Itanium processor [13] protect L2 and L3 caches with ECC, but we are not aware of any processor employing ECC-based protection mechanism on L1-cache. This is mainly due to high overheads of ECC implementation [11]. Consequently, novel techniques are required for caches that can eventually reduce failure rates while incurring minimal power and performance overheads.

Multimedia systems require soft error protection, since they are increasingly being used in mission-critical and health-care applications, where human life itself may be dependent on them. Examples include exploring and sensing habitats in unreachable regions, and video surveillance in hostile, hazardous, or toxic territories. Soft errors in such systems can put human life in danger.

In this article, we propose an architecture and approach to prevent caches from soft errors for multimedia applications while minimizing performance and power overheads at a minimal loss in quality of service (QoS). The main observation is that in multimedia applications, *not all data is equally failure critical*. The image data or audio data is not as critical for failure as the loop variable or the stack pointer. While the occurrence of a soft error in an image pixel may only result in a slight degradation in the image quality, a soft error in the loop variable may result in a segmentation fault. In such a case, we say that the image pixel is a *failure non-critical* data, while the loop variable is a *failure critical* data.

To exploit the difference in the failure-criticality of the data in multimedia applications, we propose a novel architecture - Partially Protected Cache (PPC). A PPC architecture maintains two caches at the same level of memory hierarchy. One of the caches is protected from soft errors, while the other one is unprotected. By mapping the failure critical data into the protected cache, and mapping the failure non-critical data into the unprotected cache, the failure rate of applications can be drastically improved. Note that this improvement in failure rate is obtained mostly at the cost of QoS with very small impact on power and performance. We find that the effectiveness of PPC architectures is very sensitive to the PPC cache configuration. For example, the failure rate difference among 30 configurations we studied is up to 26×. Furthermore, exploration of the large design space to find out the best cache configuration is intensively time-consuming because of the high number of simulations required, especially for the failure rate. To efficiently find out interesting cache configurations of PPC architectures, we develop several exploration techniques.

Our experiments on an HP iPAQ h4300 [6] like system demonstrate that PPC architectures can reduce the failure rate by 47× at only 1% degradation in performance and 3% increase in the system energy consumption on average over multimedia benchmarks as compared to the existing unprotected cache. Our heuristic algorithms can find best PPC cache configuration solutions while reducing the exploration time by more than 6× as compared to an exhaustive approach.

[1]Manuscript received October 9, 2007; revised February 1, 2008 and May 22, 2008. This work was supported in part by NSF grants CNS-0615438 and CCF-0702797.

K. Lee, I. Issenin, N. Dutt, and N. Venkatasubramanian are with the Department of Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: {kyoungwl, isse, dutt, nalini}@ics.uci.edu)

A. Shrivastava is with the Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85281 USA (e-mail: aviral.shrivastava@asu.edu)

II. Partially Protected Cache (PPC)

To compare the susceptibility of application data on soft errors, we devised a simple experiment. The data that the application accesses is divided into pages. We injected soft errors randomly in only one page, and simulated the application several times to estimate the failure rate. Soft errors were injected with a constant probability per line, and per unit time, only in the lines of the page that are in the cache. Our preliminary experiments show that soft errors in some pages are much more likely to cause an application failure than soft errors in other pages. In fact, for an image processing benchmark - *susan edges* from the MiBench suite [4], soft errors in only some of the pages (9 out of 83) cause an application failure. Soft errors in most of the other pages do not result in a failure. The degradation in quality typically shows up in the form of white/black pixels in the output image. A simple analysis reveals that the pages that do not cause failures are the image data, and the pages that contain stack variables, and other program variables cause failures. Existing solutions that protect the whole cache using ECC is an overkill for these multimedia data pages, especially in situations where a little loss in quality of service can be tolerated. An ideal solution would provide protection to only the pages that may cause application failures, and reduce the overheads by not protecting data that may not cause application failures.

To provide different levels of protection against soft errors, we propose a novel cache architecture, Partially Protected Cache (PPC). Our concept of Partially Protected Cache is derived from the concept of Horizontally Partitioned Caches (HPC) [3]. HPC is a promising technique in which the processor has multiple (typically two) caches at the same level of memory hierarchy, and partitioning the application data wisely between the two caches can improve both performance and energy consumption [3], [14]. Similarly, PPC architectures will have multiple caches at the same level of memory hierarchy, varying in the level of soft error protection they provide. In particular, in this article we consider two data caches at the L1 level, named the protected cache and the unprotected cache as shown in Fig. 1. The protection against soft errors in the protected cache can be provided by any of the existing techniques, e.g., increased transistor size, increased supply voltage, SEC-DED, etc. In this article, we consider that the protected cache has SEC-DED to correct one bit error and detect two bit errors.

The memory is mapped to the two caches at a page level of granularity. Each page has a Cache Mapping Attribute or CMA. CMA defines which cache the page is mapped to. When a new page is requested in the cache, it comes into the cache defined by the CMA. CMA is stored in the Translation Look-aside Buffer (TLB) along with the address mapping. When the processor requests any cache data, first the TLB lookup is performed to see if the page is present in the cache, and if yes, to figure out which of the two caches is selected. Therefore, only one cache lookup is performed per access. Note that our approach for data partitioning into the two caches does not increase the number of pages, nor the number of TLB misses. Every time data is written into the cache, the data has to be encoded, and every time it is read from the cache, the data needs to be decoded and a check needs to be performed for occurrence of soft errors. Thus, the SEC-DED decoder becomes a part of timing critical path, and has power and performance overheads.

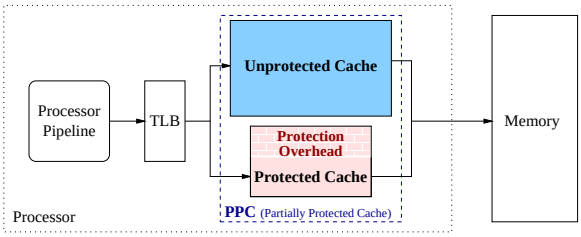


Fig. 1. Partially Protected Cache Architecture - one protected and the other unprotected at the same level of memory hierarchy

In order to minimize the performance impact of the PPC architecture, the protected cache should be small, so that the total penalty of the protected cache and the SEC-DED implementation is less than or equal to that of the unprotected cache. However, since the protected cache is now smaller, it is important to map data very carefully into this cache. Mapping too much data into the protected cache can result in frequent misses, and therefore degrade performance.

To exploit the PPC architecture, the compiler has to categorize and partition the application data into the protected and the unprotected cache. In general, a detailed analysis for each variable is needed to be able to partition the application; fortunately the characteristics of multimedia applications simplify this analysis. In multimedia applications, while a soft error in an image pixel may only cause minor distortion in the image, or negligible loss in QoS, a soft error in the loop control variable may result in memory segment violation, or a failure. Other examples of failures caused by soft errors include system crash and the infinite execution of a loop. For multimedia applications, we define the multimedia data as failure non-critical (FNC), and all the rest of the data as failure critical (FC).

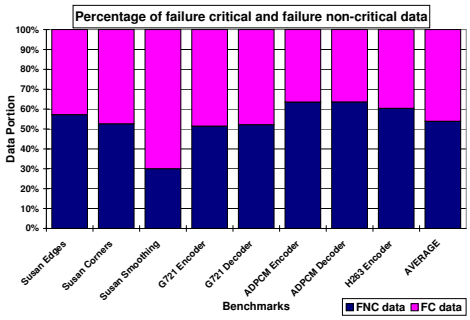


Fig. 2. Size of failure critical and failure non-critical data in applications

Fig. 2 plots the percentage of failure critical and failure non-critical data in the various multimedia benchmarks, as found by our method. We have observed that even this simple strategy can mark between 30% to 63% of data as failure non-critical. A better data analysis technique can discover additional failure non-critical data, and therefore will improve

the effectiveness of our technique. However, even the simple technique of finding the failure critical data is quite effective. Also note that it is very easy for the designer to manually identify the multimedia data, since it is typically present in large arrays.

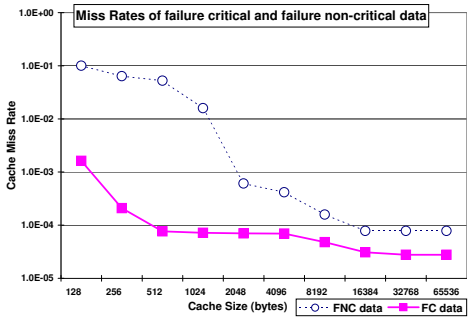


Fig. 3. Cache miss rates of failure critical and failure non-critical data (benchmark - *susan smoothing*)

The other concern with mapping data to the small protected cache in PPC might be the possible negative impact on performance. Fig. 3 plots the cache miss rates of the failure critical and failure non-critical data for various cache sizes. We observe that the slope of the miss rate of the failure critical data is less than that of the failure non-critical data. This implies that the size of the protected cache can be reduced without much performance penalty. The reason behind this observation is that the failure critical data that we have marked comprises of the local variables, function stack, etc. which have much better cache behavior than that of the multimedia data. As a result, we can achieve low failure rates without significant power and performance overheads.

To demonstrate the effectiveness of PPC architecture, we have developed a compiler-simulator-analyzer framework [7], in which a compiler generates a page mapping list and an executable, a simulator (a modified sim-cache simulator from SimpleScalar [2]) runs an executable by mapping pages according to cache configurations, and the outputs are analyzed in terms of performance, power, and failure rate. Our experiments on an HP iPAQ h4300 [6] like system demonstrate that PPC architectures can reduce the failure rate by 47$\times$ at only 1% degradation in performance and 3% increase in the system energy consumption with improved QoS by 54% as compared to the existing unprotected cache. And as compared to previously proposed solution of protecting the whole cache, PPC achieves almost the same failure rate, but with 16% performance improvement and 8% energy reduction at the QoS penalty of 32%.

III. Design Space Exploration

We have demonstrated the superiority of PPC architectures over the traditional cache configurations in Section II. There is a definite need to choose the best configuration of a PPC to satisfy multiple design constraints such as the failure rate, power, and performance. However, those constraints are sensitive to cache parameters such as size and set-associativity; furthermore the design space exploration of PPC configurations has very high computational requirements. In this section, we present our exploration heuristics to find interesting configurations efficiently.

A. Sensitivity of Cache Parameters in PPC

In the context of multi-dimensional constraints, considering only one constraint can lead to very bad configuration in other constraints. For example, the best configuration in terms of the runtime among PPC configurations with 32 KB unprotected cache is 16 KB protected cache in our limited set of experiments. However, this configuration is bad in terms of failure rate since it has about 5 times higher failure rate than the best failure rate configuration. Also, other parameters such as set-associativity cause high variation with respect to performance and energy consumption in our experiments. Indeed, we have observed that overall variations can lead up to 26 times difference in the failure rate, up to 10% in performance, and up to 3.6 times in energy consumption for the benchmark *susan corners* as will be shown in Fig. 5(a) and Fig. 5(b). Thus, there is a definite need to explore the design space and to identify PPC configurations satisfying multiple constraints.

B. Design Space Exploration Problems

In these experiments, we keep the unprotected cache configuration constant, i.e., 32 KB, 32-way set associative, and a 32-byte line. We only vary the protected cache configuration. We vary the protected cache sizes from 1 KB to 16 KB in exponents of 2, and set associativities from 1 to 32-way. Thus, there are only $5 \times 6 = 30$ configurations.

An exhaustive search of these cache configurations requires evaluating failure rates for all the 30 configurations. However, evaluating the failure rate of each configuration requires about 1,000 simulations based on the failure rate estimation in [7].Thus, if each simulation takes about an hour, the exploration would require 30,000 hours, which is approximately 3 years. Although this job is intrinsically parallelable, it is highly complex in computation. Clearly, the simulation time is far too much for design space exploration. Note the computational challenge, even though our design space is extremely small, i.e., only 30 configurations. In reality, the PPC design space can be extremely large, if we also consider variations in the unprotected cache configurations. We chose this design space for feasibility of our experiments. Owing to the immensely time consuming nature of Design Space Exploration (DSE) in failure rate computations, it is very important to develop efficient design space exploration algorithms to determine the best protected cache configurations.

In this section, we present DSE algorithms, for two interesting problems.

Problem 1 *Given maximum runtime and maximum energy consumption constraints, determine the protected cache configuration that will result in minimum failure rate*

Problem 2 *Given maximum failure rate and maximum energy consumption constraints, determine the protected cache configuration that will result in minimum runtime*

C. BFExplore - DSE algorithm for Problem 1

BFExplore(S, A, $R_{constraint}$, $E_{constraint}$)
01: $S_{min} = 1, S_{max} = 16$
02: $A_{min} = 1, A_{max} = 32$
03: **for** ($s = S_{min}; s \leq S_{max}; s = 2 \times s$)
04: **for** ($a = A_{max}; a \geq A_{min}; a = a \ / \ 2$)
05: $R_{s,a} = evalRuntime(s, a)$
06: **if** ($R_{s,a} \leq R_{constraint}$)
07: $E_{s,a} = evalEnergy(s, a)$
08: **if** ($E_{s,a} \leq E_{constraint}$)
09: **return** $\{s, a\}$
10: **endif**
11: **else**
12: $a = A_{min} - 1$
13: **endif**
14: **endfor**
15: **endfor**
16: **return** $NULL$
endBFExplore()

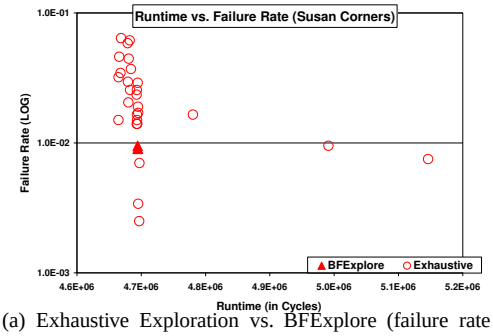
Fig. 4. BFExplore Algorithm

For Problem 1, we develop a heuristic algorithm, BFExplore, described in Fig. 4. This problem is the same as how to find the minimal size of the protected cache satisfying the given constraints of power and performance because the failure rate is determined by the size of the protected cache, and a smaller size results in a lower failure rate. S is a set of sizes of the protected cache from 1 KB (S_{min}) to 16 KB (S_{max}) (Line 01). A is a set of set-associativity from 1-way (A_{min}) to 32-way (A_{max}) (Line 02). BFExplore algorithm starts with a cache configuration with minimal protected cache size and maximal set-associativity (Lines 03-04). If this configuration satisfies the constraints ($R_{constraint}$, $E_{constraint}$), this configuration is returned as the best configuration in terms of failure rate (Lines 05-09) since it is the minimal protected cache size. If only energy constraint is not met (Line 10), it decreases set-associativity and repeats runtime and energy consumption evaluations. But, if the runtime constraint is not satisfied, it increases the protected cache size, and repeats the algorithm (Lines 11-14). If it reaches the end of the algorithm, there is no configuration satisfying the given constraints (Line 16).

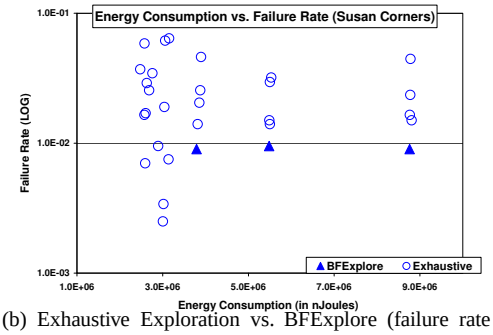
When we explore the design space for the best configuration in terms of the failure rate with the given constraints (e.g., as energy consumption less than 4.0E+6 nJ and runtime less than 4.7E+6 cycles), the exhaustive approach without any observations require 30 explorations for each configuration as shown in Fig. 5(a) and Fig. 5(b), which is about 30,000 evaluations. However, BFExplore does not evaluate the failure rates for each configuration, which results in a huge reduction. In this example, we only evaluate 3 configurations as shown in Fig. 5(a) and Fig. 5(b), indicating that we reduce the number of explored configurations by 10 times and thus decrease the number of evaluations by about 10,000 times.

D. BRExplore - DSE algorithm for Problem 2

For the second problem, we present a heuristic algorithm BRExplore as described in Fig. 6. First, our algorithm finds the largest protected cache to satisfy the given failure rate



(a) Exhaustive Exploration vs. BFExplore (failure rate and runtime)



(b) Exhaustive Exploration vs. BFExplore (failure rate and energy consumption)

Fig. 5. Design Space Explorations to find the best configuration for a PPC in terms of failure rate while satisfying energy consumption ($< 4.0E + 6$) and runtime ($< 4.7E + 6$) (32 KB unprotected cache with varying protected cache size and set-associativity, benchmark - *susan corners*)

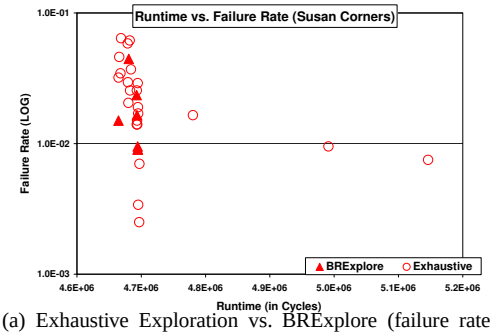
BRExplore(S, A, $F_{constraint}$, $E_{constraint}$)
01: $S_{min} = 1, S_{max} = 16$
02: $A_{min} = 1, A_{max} = 32$
03: **for** ($s = S_{max}; s \geq S_{min}; s = s \ / \ 2$)
04: $F_s = evalFailure(s)$
05: **if** ($F_s \leq F_{constraint}$)
06: **return AssocExplore** (s, A, $E_{constraint}$)
07: **endif**
08: **endfor**
09: **return** $NULL$
endBRExplore()
AssocExplore (s, A, $E_{constraint}$)
10: **for** ($a = A_{max}; a \geq A_{min}; a = a \ / \ 2$)
11: $E_{s,a} = evalEnergy(s, a)$
12: **if** ($E_{s,a} \leq E_{constraint}$)
13: **return** $\{s, a\}$
14: **endif**
15: **endfor**
16: **if** ($s \leq S_{min}$)
17: **return** $NULL$
18: **else**
19: $s = s \ / \ 2$
20: **return AssocExplore** (s, A, $E_{constraint}$)
21: **endif**
endAssocExplore ()

Fig. 6. BRExplore Algorithm

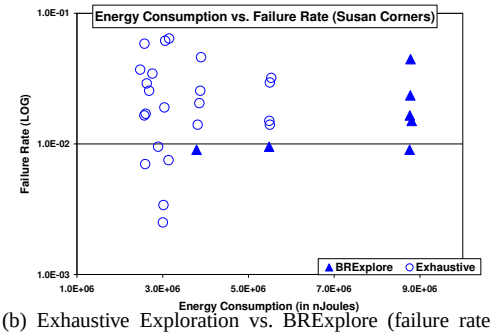
$F_{constraint}$ (Lines 03-08). Once it is found, all the smaller ones satisfy $F_{constraint}$. The second step evaluates the energy consumption of each configuration with a decrease of set-associativity. If it satisfies the given energy consumption $E_{constraint}$, this is the best configuration (Lines 10-14) in

terms of runtime because a larger cache with a larger set-associativity shows the better performance, i.e., the lower runtime. This step for energy consumption evaluation is repeated after decreasing the protected cache size (Lines 19-20) until a satisfied configuration is found. Otherwise, it fails to find the configuration satisfying the given constraints, and returns $NULL$ (Line 09, 17).

The number of evaluations for our BRExplore is 5,006 (5 $\times$ 1,000 for failure rate evaluation and 6 $\times$ 1 for energy consumption evaluation) in the worst case where the minimal protected cache size only satisfies the failure rate and only one set-associativity with the minimal size meets the given energy constraint. In the best case, we can find the best configuration with only 1,001 simulations (1 $\times$ 1,000 for failure rate evaluation and 1 $\times$ 1 for energy evaluation). Thus, our proposed BRExplore algorithm can reduce the number of evaluations by up to 30 times as compared to an exhaustive exploration since the exhaustive search requires about 30,000 evaluations. When we explore the design space for the best configuration in terms of runtime with the given constraints such as energy consumption less than 4.0E+6 nJ and failure rate less than 0.01 (near to the worst case), the exhaustive approach explores the whole 30 points as shown in Fig. 7(a) and Fig. 7(b), which indicate the 30,000 simulations for only failure rate evaluations. However, our BRExplore can find this near-to-optimal one (1 KB protected cache with 8-way set-associativity) through 7 points out of 30 points, as shown in Fig. 7(a) and Fig. 7(b). This is approximately 6 times reduction in terms of the number of evaluations since BRExplore runs 5,000 evaluations for the failure rate and 2 evaluations for energy consumption.



(a) Exhaustive Exploration vs. BRExplore (failure rate and runtime)



(b) Exhaustive Exploration vs. BRExplore (failure rate and energy consumption)

Fig. 7. Design Space Explorations to find the best configuration for a PPC in terms of runtime while satisfying energy consumption ($< 4.0E + 6$) and failure rate (< 0.01) (32 KB unprotected cache with varying protected cache size and set-associativity, benchmark - *susan corners*)

IV. Conclusion

Due to the incessant technology scaling, soft errors are becoming a critical design concern for system reliability. Especially, soft errors in caches are the most important due to large area and low voltage beyond sub-micron technology.

In this article, we propose a novel approach for mitigating failures caused by soft errors in multimedia embedded applications where power, performance, and reliability are all a concern. We present Partially Protected Cache (PPC) architectures and propose a simple technique for mapping data into such caches geared towards their use in multimedia applications. We also propose heuristic algorithms, BFExplore and BRExplore, to efficiently find the best configuration for our proposed PPC architectures from the large design space resulting from considering multiple constraints such as failure rate, runtime, and energy consumption.

Our future work includes finer data partitioning and compiler techniques to intelligently separate failure non-critical data from failure critical data. Further, we plan to extend the applicability of our technique for other components such as instruction caches in general applications, and to develop DSE algorithms to adjust PPC configurations at runtime.

References

[1] R. Baumann. Soft errors in advanced computer systems. *IEEE Design and Test of Computers*, pages 258–266, 2005.

[2] D. Burger and T. M. Austin. The SimpleScalar Tool Set, version 2.0. *SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[3] A. Gonz'alez, C. Aliagas, and M. Valero. A data cache with multiple caching strategies tuned to different types of locality. In *International Conference on Supercomputing*, pages 338–347, July 1995.

[4] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE Workshop Workload Characterization*, Dec 2001.

[5] P. Hazucha and C. Svensson. Impact of CMOS technology scaling on the atmospheric neutron soft error rate. *IEEE Trans. on Nuclear Science*, 47(6):2586–2594, 2000.

[6] Hewlett Packard, http://www.hp.com. *HP iPAQ h4000 Series - System Specifications*.

[7] K. Lee, A. Shrivastava, I. Issenin, N. Dutt, and N. Venkatasubramanian. Partially protected caches to reduce soft error induced failures in multimedia systems. Technical Report TR 06-03, UCI, June 2008.

[8] J.-F. Li and Y.-J. Huang. An error detection and correction scheme for RAMs with partial-write function. In *IEEE International Workshop on Memory Technology, Design and Testing*, pages 115–120, 2005.

[9] P. Liden, P. Dahlgren, R. Johansson, and J. Karlsson. On latching probability of particle induced transients in combinational networks. In *IEEE International Symposium on Fault-Tolerant Computing*, 1994.

[10] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim. Robust system design with built-in soft-error resilience. *IEEE Computer*, 38(2):43–52, Feb 2005.

[11] K. Mohr and L. Clark. Delay and area efficient first-level cache soft error detection and correction. In *IEEE International Conference on Computer Design*, 2006.

[12] R. Phelan. Addressing soft errors in ARM core-based designs. Technical report, ARM, 2003.

[13] N. Quach. High availability and reliability in the Itanium processor. *International Symposium on Microarchitecture*, pages 61–69, Sep–Oct 2000.

[14] A. Shrivastava, I. Issenin, and N. Dutt. Compilation techniques for energy reduction in horizontally partitioned cache architectures. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2005.