Center for Embedded Computer Systems
University of California, Irvine

Partially Protected Caches to Reduce Failures due to Soft Errors in Mission-Critical Multimedia Systems

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

Technical Report TR-08-06
June 24, 2008

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{kyoungwl, isse, dutt, nalini}@ics.uci.edu, Aviral.Shrivastava@asu.edu
http://www.cecs.uci.edu/

Partially Protected Caches to Reduce Failures due to Soft Errors in Mission-Critical Multimedia Systems

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

Technical Report TR-08-06
June 24, 2008

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{kyoungwl, isse, dutt, nalini}@ics.uci.edu, Aviral.Shrivastava@asu.edu
http://www.cecs.uci.edu

Abstract

With advances in process technology, soft errors are becoming an increasingly critical design concern. Soft errors are manifested as a toggle in Boolean logic, which may result in failure of the system functionality. Owing to their large area, high density, and low operating voltages, caches are worst hit by soft errors. Although Error Correction Code (ECC) based mechanisms have been suggested to protect the data in caches, they have high performance and power overheads. We observe that in multimedia applications, not all data require the same amount of protection from soft errors. In fact, an error in the multimedia data itself does not result in a failure, but often just results in a slight loss of quality of service. Thus, it is possible to trade-off the power and performance overheads of soft error protection with quality of service. To this end, 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$\times$ reduction in failure rate, at only 1% runtime and 3% power overheads. In addition, we observe that the failure rate reduction obtained by PPCs is very sensitive to the PPC cache configuration. Therefore, there exists the scope of further improving the solution by correctly parameterizing the PPC configurations. Consequently, we develop Design Space Exploration (DSE) strategies to find out the best PPC configuration. Our DSE technique can reduce the exploration time by more than 6$\times$ as compared to the exhaustive approach.

Contents

List of Figures

List of Tables

Partially Protected Caches to Reduce Failures due to Soft Errors in Mission-Critical Multimedia Systems

K. Lee[1], A. Shrivastava[2], I. Issenin[1], N. Dutt[1],and N. Venkatasubramanian[1]

[1]Department of Computer Science
School of Information and Computer Sciences
University of California, Irvine, CA 92697, USA

[2]Department of Computer Science and Engineering
School of Computing and Informatics
Arizona State University, Tempe, AZ 85281, USA

June 24, 2008

Abstract

With advances in process technology, soft errors are becoming an increasingly critical design concern. Soft errors are manifested as a toggle in Boolean logic, which may result in failure of the system functionality. Owing to their large area, high density, and low operating voltages, caches are worst hit by soft errors. Although Error Correction Code (ECC) based mechanisms have been suggested to protect the data in caches, they have high performance and power overheads. We observe that in multimedia applications, not all data require the same amount of protection from soft errors. In fact, an error in the multimedia data itself does not result in a failure, but often just results in a slight loss of quality of service. Thus, it is possible to trade-off the power and performance overheads of soft error protection with quality of service. To this end, 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$\times$ reduction in failure rate, at only 1% runtime and 3% power overheads. In addition, we observe that the failure rate reduction obtained by PPCs is very sensitive to the PPC cache configuration. Therefore, there exists the scope of further improving the solution by correctly parameterizing the PPC configurations. Consequently, we develop Design Space Exploration (DSE) strategies to find out the best PPC configuration. Our DSE technique can reduce the exploration time by more than 6$\times$ as compared to the exhaustive approach.

1 Introduction

System reliability is becoming the 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 transient faults [13, 44]. A transient fault results in erroneous program state and incorrect program outputs, but they are random and non destructive, i.e., resetting the device restores normal behavior. While transient faults may be caused due to several reasons such as the intrinsic noise in the circuit and signal interference, radiation-induced faults are responsible for more failures than all the other causes of transient faults combined [5]. The crash of Sun Microsystem's flagship servers [25] and errors in CISCO 12000 line cards [2] have been attributed to soft errors, and have resulted in significant fiscal losses.

A high energy radiation particle, e.g., an alpha particle, a neutron or a 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 an *Upset*. An upset may have catastrophic consequences including the application generating incorrect results, accessing protected memory regions, crashing, or going into an infinite loop. The incorrect or erroneous behavior of an application due to upsets is called a *Failure*. Note that not all upsets result in a failure due to maskings such as electrical masking, logical masking, latching-window masking, and microarchitectural masking. An upset will become a failure if it is not masked by any of these effects. Consequently, research has looked at increasing the effectiveness of each of these masking effects. Owing to the effectiveness of the latching-window masking, upsets in memory elements have significantly higher probability of causing a failure than upsets in combinational logic [10, 23]. 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 [27], more than 50% of soft errors happen in memories.

While it is possible to employ simple redundancy based techniques in off-chip memories, they are not suitable for caches, which are highly sensitive to the performance and power overheads of redundancy based techniques. For example, using *Single-bit Error Correction and Double-bit error Detection* (SEC-DED) codes may increase the cache access time by 95% [21] and power consumption by up to 22% [33]. Thus, only a few processors such as the Intel Itanium processor [35] 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 [18, 28]. Consequently, novel techniques are required for caches that can eventually reduce failure rates while incurring minimal power and performance overheads.

Multimedia applications require soft error protection, since they are increasingly being used in mission-critical 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. In addition, electronic devices that may be sensing or controlling vital human functions, e.g. heart lung machine, pacemaker, etc., need to be very reliable, or they may directly put the human life in danger. Significant research has been accomplished in devising mechanisms

for protection against soft errors, and making digital devices more robust.

In this article, we propose an architecture and approach to prevent caches from soft errors for multimedia applications while minimizing performance, power, and area overheads at a minimal loss in quality of service. 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 variables 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 segmentation fault. In such a case, we say that the image pixel is a *failure non-critical* (FNC) data, while the loop variable is a *failure critical* (FC) data.

To exploit the difference in the failure-criticality of the data of multimedia applications, we propose a novel architecture - Partially Protected Caches (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 rest of the 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 Quality of Service (QoS) with very small impact on power and performance. We find that the effectiveness of PPC architectures is very sensitive on the PPC cache configuration. And exploring a huge possible 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 the best cache configuration of PPC architectures, we have developed several exploration techniques.

Our experiments on an HP iPAQ h4300 [14] 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 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 rates, but with 16% performance improvement and 8% energy reduction. Our heuristic exploration algorithms can find best PPC cache configuration solutions while reducing the exploration time by more than 6$\times$ as compared to the exhaustive exploration approach.

2 Background

2.1 Soft Errors

Poor system design, crosstalk, and radiations can cause upsets or bit-flips in electronic devices such as memory and logic components where these upsets can be manifested as soft errors or transient faults. External radiations are primary contributors to soft errors, which have been investigated since 1970's.

Fig. 1 [26] illustrates the mechanism of a soft error event in a CMOS device. When energetic particles such as alpha particles, neutrons, and protons from packaging material or cosmic rays strike the sensitive area of the silicon device, they generate electron-hole pairs in the wake. The source and diffusion nodes of a transistor can collect these charges, $Q_{collected}$. When $Q_{collected}$ becomes more than some critical value, $Q_{critical}$, the state of the logic device, e.g., a Boolean gate, may invert. Since this logic toggle is temporary, the occurrence of such a defect is called a transient fault or a

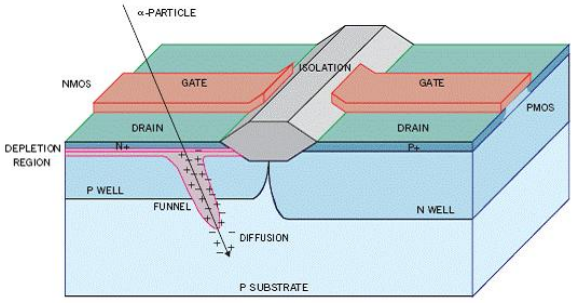


Figure 1: External radiation may induce soft errors

soft error. Neutrons among several radiations are considered as a critical source causing soft errors since they cannot be shielded completely even with most man-made construction. For instance, a radiation can penetrate five feet of concrete [26].

2.2 Soft Error Rate and Vulnerability

The soft error rate (SER) in Fig. 1 is related as

$$SER \propto N_{flux} \times CS \times e^{-(\frac{Q_{critical}}{Q_s})} \quad (1)$$

where N_{flux} is the intensity of the neutron flux, CS is the area of the cross section of the node, and Q_s is the charge collection efficiency [13]. Since $Q_{critical}$ is proportional to the node capacitance C and the supply voltage V, SER has an exponential relationship with the supply voltage as well as the capacitance from Equation (1). Thus, with decreasing supply voltage and shrinking feature size, the rate of soft errors will increase exponentially [13, 44]. In fact, Baumann [5] predicts that the SER in the next generation SRAMs will be up to two orders of magnitude higher while the SER of DRAMs has been saturated. Multiplied by the trend of increasing size of SRAMs in multimedia embedded systems, the SER is becoming an important design concern.

However, not always an upset becomes a soft error due to masking effects. Four kinds of effects naturally mask an upset to translate into a soft error: i) **electrical masking** occurs when the pulse resulting from a particle strike is attenuated by subsequent logic gates due to the electrical properties of the gates to the point that it does not affect the result of the circuit, ii) **logical masking** occurs when a particle strikes a portion of the combinational logic that is blocked from affecting the output due to a subsequent gate whose result is completely determined by its other input values, iii) **latching-window masking** occurs when the pulse resulting from a particle strike reaches a latch, but not at the clock transition where the latch captures its input value, and iv) **microarchitectural masking** occurs when the data that is changed is obsolete and does not affect the program state or the program output.

On the other hand, the unit of SER, FIT (Failures in Time), is the number of soft errors per Mbit for one billion-operation hours. Thus, the SER is linearly proportional to the cache size and the execution time apparently. However, the possibility from a fault (i.e., upset) to an error (i.e., soft

error) is not directly proportional to the cache size and the execution time since all upsets do not propagate errors. Therefore, there is a definite need for an accurate metric to estimate soft errors.

Substantial work has concentrated on estimating the soft error rate, and the corresponding failure rate. Architectural Vulnerability Factors (AVF) was defined in [30] as the probability that a fault in a particular structure will result in a visible error in a final output. However this factor is constant for a given structure, and is not application dependent. This methodology has been widely exploited [27], and also extended to present the vulnerability of the cache systems in [4, 3]. In particular, Asadi et al. presented the critical time of the critical word , the residency time of the word data in caches, and examined the vulnerability of the cache components and the effects of cache policies such as flushing, write-thru and refreshing. However, they did not capture the byte-level residency time, and ignored the effects of the write operation in a word on the other words in the same line at eviction. Similarly, the temporal vulnerability factor (TVF) [43] and the cache vulnerability factor (CVF) [45] have been proposed as metrics to estimate the vulnerability of the caches to soft errors. However, they failed to present the actual relationship of the vulnerability factor and the failures of applications.

2.3 Soft Error Protection Techniques

Solutions to combat the challenge of soft errors have been proposed at various levels of design abstraction from process technology to architectural solutions.

2.3.1 Process Technology Solutions

Radioactive substances such as alpha particles emitted by packaging and wafer processing materials are one of the major sources of radiations that cause soft errors in semiconductors. Thus, advances in process technology such as purification of packaging materials, radiation hardening, and elimination of Boron-10 (B^{10}) impurities are expected to mitigate the soft errors [6]. However, the effects of interaction between high energetic cosmic particles (e.g., neutrons) and radioactive materials cannot be prevented completely [26].

Process technology solutions such as SOI (Silicon On-Insulator) processes [31, 38] have been proposed. In order to mitigate the soft errors, they extend the depletion region or raise the capacitance, which increases the critical charge of semiconducting devices. However, process engineering technology may require the cost of additional process complexity, the loss of manufacturability, and extra substrate cost [5].

2.3.2 Software and Compiler Solutions

Reis et al. [36] presented the software-implemented fault tolerance (SWIFT) for soft error detection by exploiting unused resources and enhancing control-flow checking. Also, Luccetti et. al [24] proposed software mechanisms to tolerate soft errors by leveraging virtual machine and memory sharing techniques. However, they are limited to detecting errors, and must be used in conjunction with recovery techniques.

Through the user-specified annotations, the compiler can separate and map data elements in programs to reliable domain which has protection against soft errors, and to unreliable domain without protection [9]. But it required the annotation for important data by user specification.

2.3.3 Microarchitectural Solutions

At the processor architecture level, error detection and correction codes (EDC and ECC) have been widely investigated and implemented as the most effective schemes in order to detect and correct soft errors in memory systems like cache and main memory [34]. However, an ECC system implementation consists of an encoding block as well as a decoding block responsible for detection and correction, and of extra bits storing ECC values. Whenever data is written, an encoding block need to code data and store the error correction codes for this data at an extra storage. And also every read operation requires to decode data, which can correct errors by comparing ECC values at the extra storage. Thus, ECC consumes extra energy and incurs performance delay as well as additional area cost [3, 21, 33, 47].

In order to reduce high overheads due to ECC, many architectural techniques have been studied. Kim and Somani [17] proposed the parity cache with the check codes and the shadow cache with multiple replicas for higher reliability. They protect the cache data partially using data locality so the coverage for protection may be small, or they have overheads to keep replicas. Zhang et al. [47] proposed in-cache replication where the dead cache block space is recycled to hold replicas of the active cache block. However, since the load of replication is limited, it cannot provide the full protection. And Zhang [46] presented replication cache where a small fully associative cache is added to keep the replica of every write to the L1 data cache, but it incurs overheads in terms of power, perforamance, and area to keep the replica and to detect as well as correct errors. A cache scrubbing technique [29] has been proposed, which can fix all single-bit errors periodically and prevent potential double-bit errors. However, this technique may have high performance, power, and area overheads due to ECC implementation. An adaptive protection algorithm has been proposed for low power caches, which protects clean data less than dirty data blocks [22]. For high reliable caches, Neuberger et al. [32] presented a combined method with Hamming and Reed-Solomon codes in order to correct at least double-bit transient faults in an energy-efficient way. Kim [18] also proposed the combined approach of parity and ECC codes to generate the reliable cache system in an area-efficient way. Recently, Cai et al. [8] explored cache parameters, especially cache sizes, to increase reliability while considering power and performance. However, they all exploit expensive error correcting codes in order to protect all the data in caches, which may waste power and performance for unnecessary protection in multimedia applications.

2.4 Our Contributions

This work starts from the observation that not all data require the same amount of protection against soft errors. We propose a new cache architecture which is able to provide different levels of protection for application data. By categorizing and mapping the data wisely to these caches, effective protection from soft errors can be achieved at minimal power and performance overheads. Also we present heuristic exploration algorithms to find out a PPC configuration under the multiple constrints

since cache parameters cause high impacts on failure rate, power, and peformance, and exploring design space is intensively time-consuming.

Our approach has several novel features: i) we try to reduce the failures due to soft errors, instead of reducing the soft errors themselves, ii) our approach reduces failures due to soft errors in data caches at minimal impact on power or performance, however at the cost of QoS degradation, iii) our technique is very effective for multimedia applications, iv) our selective data protection can be used in conjunction with previously proposed soft error protection mechanisms, and v) our heuristic algorithms can efficiently explore a design space to find the best configuration satisfying multiple constraints.

3 Motivation

To compare the susceptibility of application data on soft errors, we devised a simple experiment. As shown in Fig. 2, 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 are injected with a constant probability per line, and per unit time, only in the lines of the page that are in the cache.

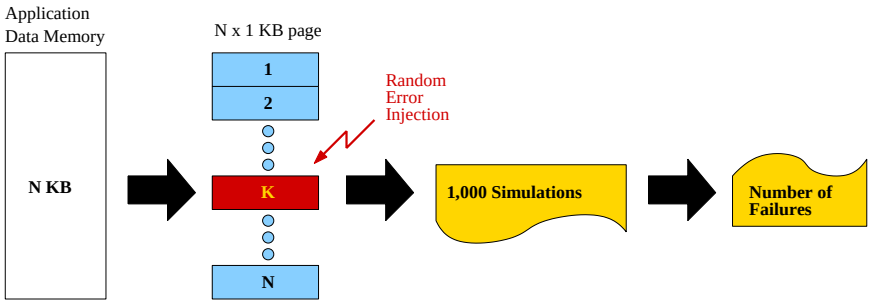


Figure 2: Failure rate due to soft errors in each page (= number of failures out of 1,000 runs)

Fig. 3 plots the failure rate of the *susan edges* benchmark from MiBench suite [12] when soft errors are injected in one page at a time, on all pages accessed by the application. For these experiments, the soft error injection rate was 10^{-6} errors per line per page, the page size was 1 KB, and data cache was a 32 KB 32-way set associative cache with 32 byte linesize (similar to the Intel XScale [15]). We ran simulations 1,000 times, and the failure rate was estimated as the number of runs out of 1,000 simulations that the application failed. *susan edges* is an image processing application, which takes a pgm image and marks the edges in a given image. The output is also a pgm image. The simulation is declared as a failure, if the output image file cannot be opened by the *xv* [16] image viewer application.

Fig. 3 shows that soft errors in some pages are much more likely to cause an application failure than soft errors in other pages. In fact, for this image processing application, only soft errors in some of the pages, 9 pages 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

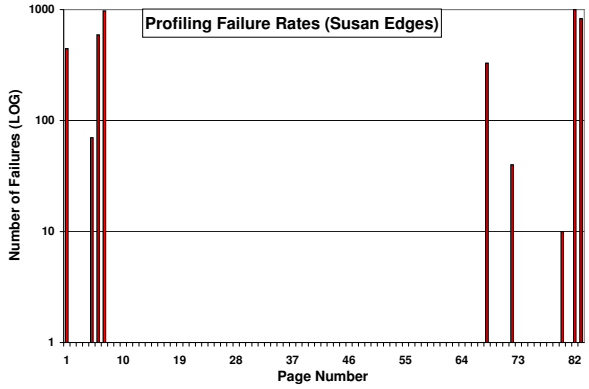


Figure 3: Failure rate distribution (benchmark : *susan edges*) - failures are reported in occurrences of soft errors at only 9 pages out of 83

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 a 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.

4 Our Approach

4.1 Architecture Model and Approach

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) [11]. 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 the performance and the energy consumption [37, 19, 40]. 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. 4. 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 one of the two caches. 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, and 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 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.

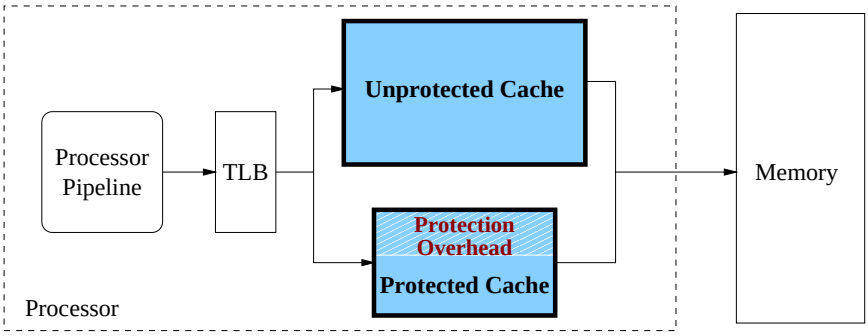


Figure 4: Partially Protected Cache Architecture - one protected and the other unprotected at the same level of memory hierarchy

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 the unprotected cache. However, now since the protected cache is 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 performance degradation.

4.2 Application Data Partitioning

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, 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 an infinite loop the application might go into. For multimedia applications, we define the multimedia data as failure non-critical, and all the rest of the data is failure critical.

Fig. 5 plots the percentage of failure-critical and failure non-critical data in the various multimedia benchmarks, as found by our method. The plot shows that even this simple strategy can mark from 30% to 63% of data as failure non-critical. A better data analysis technique can discover more failure non-critical data, and therefore will improve the effectiveness of our technique. For example, we can apply the vulnerability metric as discussed in Section 2.2 for further data partitioning. However, even this simple technique of finding the failure-critical data is quite effective. Also note that it is very easy for the designer to identify the multimedia data. It is typically present in large arrays.

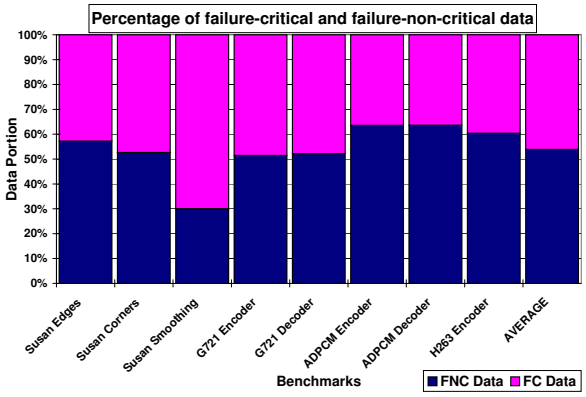


Figure 5: Size of failure critical and failure non-critical data in applications

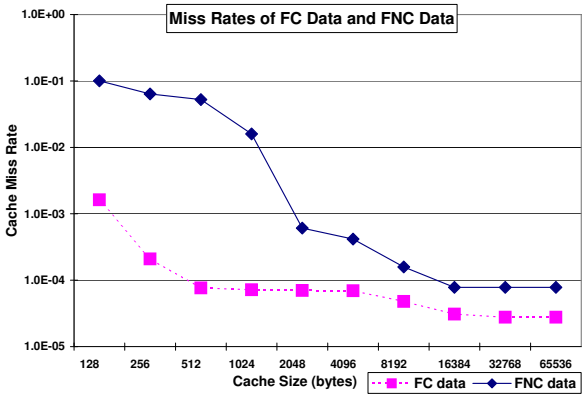


Figure 6: Cache miss rates of failure critical and failure non-critical data (benchmark - *susan smoothing*)

The other issue with mapping data to the small protected cache in PPC might be the possible performance impact. Fig. 6 plots the 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.

5 Experimental Framework

In this section, we present the experimental framework to demonstrate the effectiveness of our proposed architecture. We have developed a compiler-simulator-analyzer framework, in which a compiler generates a page mapping list as well as an executable, a simulator runs an executable by

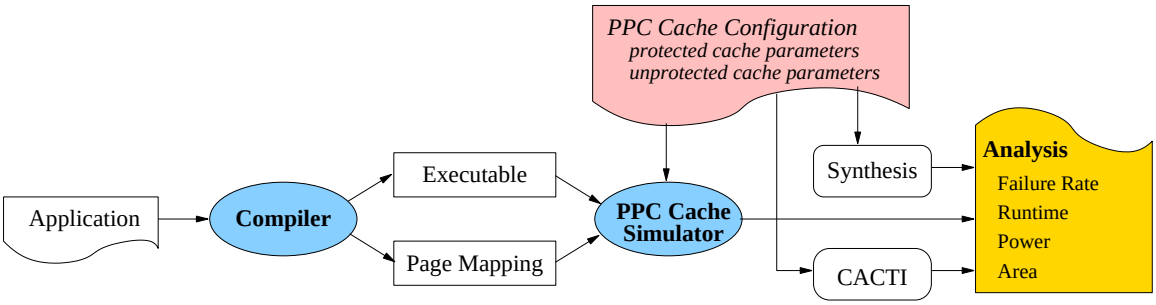


Figure 7: Experimental Framework

mapping pages according to cache configurations, and outputs are analyzed to evaluate our PPC architecture, as described in Fig. 7.

5.1 Compiler Platform and Benchmarks

We use a suite of several multimedia benchmarks from different sources for our simulations. In the domain of image processing applications, we select three image filters, namely *susan edges*, *susan smoothing*, and *susan corners*, from MiBench [12]. In the domain of audio applications, we use *adpcm encoder* and *adpcm decoder* from MiBench, and *G721 encoder* and *G721 decoder* from MediaBench [20]. As a representative video application, we choose *H263 encoder* generated from the PeaCE project [42]. These examples form a representative set of typical multimedia applications.

We mark all the arrays that contain the multimedia data with the "__FNCdata__" keyword, and make them as global variables. We would like to stress again that it is very easy for an application developer to identify the multimedia data. Then, our compiler identifies all the marked data as FNC data, and generates a page mapping file as well as an executable for an application. And all other data including dynamically allocated data are considered as FC data. This page mapping file maintains a list of virtual addresses for FNC data, so FNC data are going to be mapped into the unprotected cache and FC data into the protected cache in a PPC during simulations. Note that the page level granularity of data partitioning can cause some pages consisting of FNC data and FC data, which are considered FC and mapped into the protected cache. So our data partitioning for PPC architectures do not occur any extra overheads of space, performance, and energy consumption.

5.2 Simulation Platform

We select a simulation platform similar to the HP iPAQ h4300 [14], which has a popular Intel XScale-cored PXA255 at 400 MHz with a partitioned data cache architecture – a 32 KB main data cache and a 2 KB mini cache. The cycle-accurate simulator sim-cache from the SimpleScalar [7] has been modified to support our PPC architectures and soft error injections explained below. Our PPC architecture consists of the unprotected main data cache and the protected mini data cache with an implementation of a SEC-DED technique. Thus, the simulator takes not only the executable but also the page mapping file for data partitioning to a PPC as inputs generated by our compiler. While running an application, the simulator can map FNC data into the unprotected main cache, and FC

data into the protected mini cache in the PPC. After simulations, our PPC cache simulator returns the number of failures and outputs such as the number of instructions for a benchmark and the number of accesses as well as misses for each cache, which are used for estimating the performance and the energy consumption for this configuration and the benchmark.

Soft errors are modeled and injected randomly during simulations with an accelerated SER in order to see the results within a reasonable amount of time. Thus, our experiments relatively compare the failure rate and QoS between our PPC architecture and the existing caches with protection or without protection. Note that the other analytic models such as performance, energy consumption, and area are not affected by this accelerated SER. The base SER for single-bit errors (SBE) is 10^{-9} per instruction per KB of cache. We also consider double-bit errors (DBE) with 100 times less rate than that of SBE [44, 22], i.e., 10^{-11} per instruction per KB of cache for a DBE. For every instruction, the simulator tries to generate a soft error in a randomly selected bit in a data cache at a certain amount rate according to a cache size if it is valid in the cache. Also, DBE occurs in the protected cache as well as in the unprotected cache, while SBE occurs only in the unprotected cache since the implemented SEC-DED in the protected cache can automatically correct all single bit soft errors.

5.3 Cache Configurations

In order to evaluate our PPC architectures in terms of failure rate, power, and performance, we define three cache configurations: 1) *Unsafe*, 2) *Safe*, and 3) *PPC*. The first configuration, *Unsafe*, has a unified data cache and does not protect data at all. Thus, it is highly vulnerable to soft errors, so it will show the high failure rate, which is bad. But *Unsafe configuration* is good in terms of both performance and power since it has no overheads for data protection while the others do. The second configuration, *Safe*, has a unified data cache with implementation of a SEC-DED for data protection, and protects all data, so it is high reliable in terms of the failure rate with high overheads of both power and performance, which are bad. Our proposed *PPC configuration* protects only FC data by mapping FC data into a protected mini cache in a PPC architecture. This configuration will present the failure rate close to that of *Safe configuration* and the performance and the energy consumption close to those of *Unsafe configuration*, which is very good. All cache parameters for L1 data cache other than the size and the set-associativity are fixed in our experiments such as 32 bytes of linesize, FIFO (First-In First-Out) for replacement policy, and write-back policy.

5.4 Failure Model

Soft errors in FC data may cause failures. We define the concept of a *failure* in a simulation as the occurrence of one of these conditions: i) application crash – the application execution generates an abnormal exception or attempts to access a different memory segment, ii) infinite loop – the application does not complete even after 10 times its expected execution time, iii) bad header – the application produces incorrect header, because of which it cannot be opened by any viewer, and iv) incorrect name and size – if the output file is of different name, or of different size. Note that the degradation in the quality of service of multimedia applications due to soft errors is not defined as a failure in our study.

The occurrence of a soft error does not imply an application failure. If an error injected in a variable will not be used, then the error does not matter. However, if the erroneous value will be used in the future, then it will result in a failure. To estimate the failure rate, several simulations need to be performed, and the mean is reported as the failure rate.

Assuming that each simulation is an independent *Success* /*Failure* event, X is the number of *Success* in simulations, and we perform n experiments, then if the probability of a failure is p, then X is a binomially distributed random variable, which follows the binomial distribution with parameters n and p ($X \approx B(n,p)$). Therefore, the mean of X will be $\mu = np$, and the variance of X will be $\sigma^2 = np(1-p)$. The error $E = |\overline{X} - \mu|$ is less than or equal to $z_{\alpha/2}\sigma/\sqrt{n}$ with confidence $100(1-\alpha)\%$, where α is a confidence level. Therefore, the sample size to be able to state with $100(1-\alpha)\%$ confidence that the error $|\overline{X} - \mu|$ will not exceed a specified amount E is $N = (\frac{z_{\alpha/2}\sigma/\sqrt{n}}{E})^2$. Thus, for 95% confidence, $z_{\alpha/2} = 2$, and confidence interval .01%, the sample size, $N = 40,000p(1-p)$.

To estimate the probability of a failure, suppose the probability of a failure is 2%, then $N = 40,000 \times 0.02 \times 0.98 = 784$. Also the number of simulations needed depends on the probability of a failure p. For example, if p is 2%, then we need about 784 simulations, while if p is 20%, then we need 6,400 simulations. To overcome this, the simulations are typically set up so that the failure rate will be less than 10%. This is done by a manual process of tuning the upset injection rate for each simulation. Thus, an application is first simulated 1,000 times, which is for about 2% to 3% failure rate, and then the upset injection rate is adjusted to see the results within about 1,000 simulations.

5.5 Performance Model

For performance comparison, the runtime of an application is estimated using the statistics generated by the sim-cache simulator. Assume that the Instructions Per Cycle (IPC) of the processor to be 1. Our *Unsafe cache configuration* resembles the Intel XScale memory subsystem with cache access latency of 2 cycles, and cache miss penalty of 25 cycles as in [40]. Thus, the runtime for the *Unsafe cache configuration*, R_{unsafe}, is estimated as $R_{unsafe} = I + A_{unprotected} \times 2 + M_{unprotected} \times 25$, where I is the number of instructions executed, and $A_{unprotected}$ and $M_{unprotected}$ are the number of accesses and the number of misses, respectively, to the unprotected cache. For the *Safe cache configuration*, we consider one extra cycle penalty due to the SEC-DED check. Note that this experiment does not consider the speculative operation of ECC to reduce the critical path[1]. Thus, the runtime for the *Safe cache configuration*, R_{safe}, is estimated as $R_{safe} = I + A_{protected} \times 3 + M_{protected} \times 25$, where $A_{protected}$ and $M_{protected}$ are the number of accesses and the number of misses to the soft error protected cache, respectively. For the *PPC cache configuration*, we keep the protected cache size smaller than the unprotected cache size to ensure that the access times for both caches are the same, and equal to 2. Thus, the runtime for *PPC configuration*, R_{ppc}, is estimated as $R_{ppc} = I + (A_{protected} + A_{unprotected}) \times 2 + (M_{protected} + M_{unprotected}) \times 25$. Note that our performance model using sim-cache simulator is well matched with the runtime output from sim-outorder simulator, which generates accurate performance for multi-issue pipelined processors, but demands longer simulation time.

[1]Implementing speculative operation of ECC can reduce once cycle performance penalty, and result in better performance of PPC than Unsafe configuration

5.6 Energy Consumption Model

We estimate the energy consumption of the whole system including the processor pipeline, caches, memory and the off-chip buses. The power value per access of the unprotected cache ($E_{unprotected}$) is estimated using CACTI [39] cache energy models. In this study, the protected caches implement a Hamming code (32,6) [34] as a SEC-DED technique, and therefore consume additional energy for coding (or decoding) and writing (or reading) extra control bits (effectively increasing the linesize) for each access. To estimate the energy per access for the protected cache with the extra coding bits $E_{protected}$, we use CACTI with increased linesize value parameter from 32 to 38 bytes, where 6 bytes are control data for 32 bytes. To estimate the energy overhead due to the SEC-DED coder and decoder, we synthesize them using the Synopsys Design Compiler [41] with *lsi10k* libraries of 0.5 *u*m technology. Note that the decoder is much more complex than the coder, however, since the decoder is a part of "read operation", it should be faster. After scaling the power numbers to 0.18 *u*m technology, the decoder of our implementation consumes $E_{dec} = 0.39\ nJ$ per decoding, and the coder consumes $E_{cod} = 0.2\ nJ$ per coding. Every access to a protected cache incurs not only $E_{protected}$ but also E_{dec} to detect and correct a soft error, so the energy consumption for each protected cache access is $A_{protected} \times (E_{protected} + E_{dec})$. And also $M_{protected} \times E_{cod}$ is modeled as energy consumption for every miss to a protected cache since the data copy from memory to a protected cache demands the encoding for SEC-DED. Note that we do not consider a read and a write operation separately, and do not consider the difference between energy consumption of a clean line and a dirty line in our simulation study[2]. As in [40], we estimate that the off-chip bus consumes about $E_{bus} = 10\ nJ$ per access, while the memory access energy is $E_{mem} = 32\ nJ$ per burst. For the processor, we assume that it consumes 400 mW operating at 600 MHz, which is the normal operating mode of the Intel XScale processor [15]. Thus, E_{proc} is 0.67 *nJ* per instruction.

Using the above-mentioned energy models, we estimate the total system energy E as, $E = E_{processor} + E_{cache} + E_{memory}$ where $E_{processor} = E_{proc} \times I$, $E_{cache} = (A_{unprotected} \times E_{unprotected}) + A_{protected} \times (E_{protected} + E_{dec}) + (M_{protected} \times E_{cod})$, and $E_{memory} = (M_{unprotected} + M_{protected}) \times (E_{bus} + E_{mem})$.

5.7 Area Model

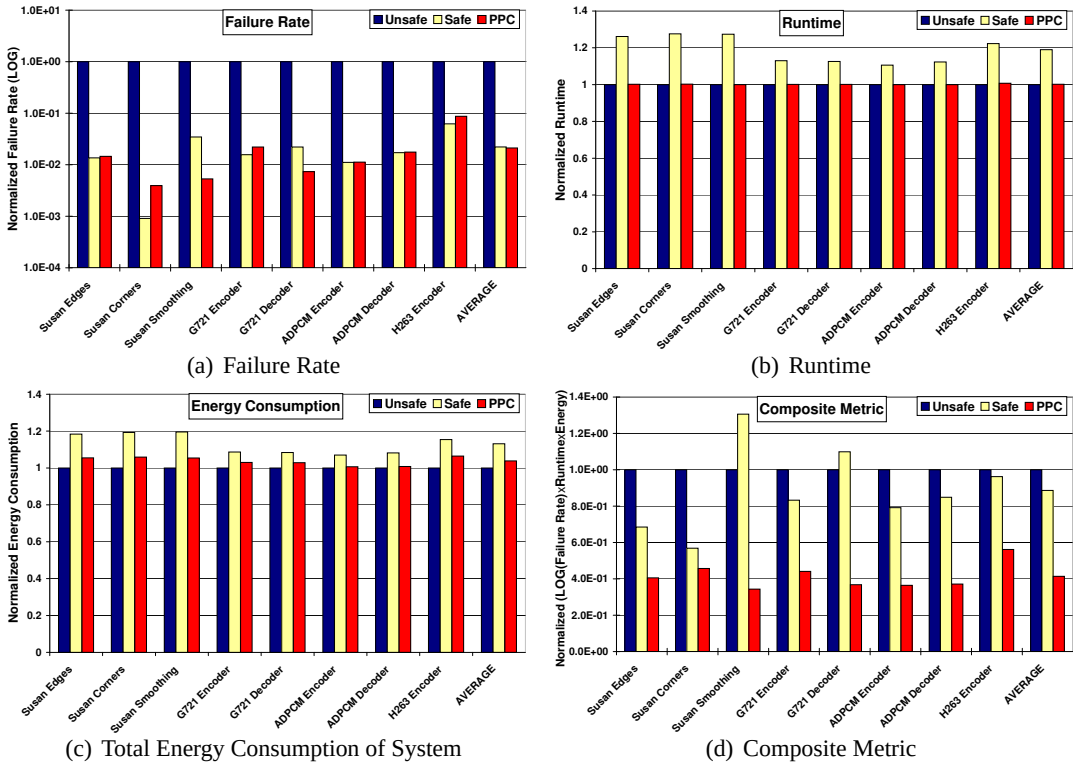
We estimate the area of the unprotected cache, $AR_{unprotected}$ using CACTI. The area overhead due to SEC-DED implementation in the protected caches is due to three sources, i.e., the coder, the decoder, and the storage required for the extra control bits. We estimate the area with the extra bits of the protected cache, $AR_{protected}$, in the cache using CACTI. The Synopsys area results scaled to 0.18 *u*m technology show that the area overhead of the coder + decoder, AR_{ecc}, is $0.55mm^2$. Thus, the area of the *PPC configuration* is computed as $AR = AR_{unprotected} + AR_{protected} + 0.55mm^2$.

[2]We evaluated the more detailed and accurate energy consumption model considering read/write operations and clean/dirty cases but the differences are insignificant

5.8 Quality of Service Model

QoS is very application specific, and the definition of QoS itself varies with applications. For QoS comparison in image processing applications, we use the PSNR (Peak Signal to Noise Ratio) metric in dB defined as: $PSNR = 10LOG_{10}(\frac{MAX^2}{MSE})$, where MAX is the maximum pixel value and MSE is the Mean Squared Error, which is the mean of the square of differences between the pixel values of the erroneous output and the correct output. QoS for audio applications is similarly defined. For video applications, the PSNR is averaged over the image frames. Please note that the QoS values in this article are calculated with exaggerated SERs, which is mainly because of observing the failure rates within a reasonable amount of simulation time, and does not affect the performance, energy consumption, and area while it affects the QoS results significantly. Thus, the reported QoS is meaningful to indicate the relative superiority rather than the absolute value of quality.

6 Effectiveness of PPC

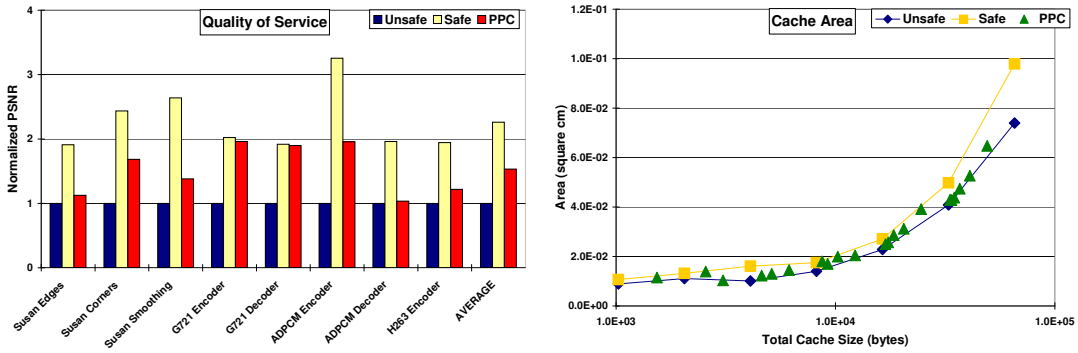


(a) Failure Rate (b) Runtime

(c) Total Energy Consumption of System (d) Composite Metric

Figure 8: Effectiveness of our PPC architectures - PPC achieves minimal failure rates with minimal energy and performance overheads

This section demonstrates the effectiveness of our approach as compared to the traditional unprotected cache configuration (Unsafe) and protected cache configuration (Safe).



(a) Quality of Service (b) Area

Figure 9: Evaluation of Quality and Area

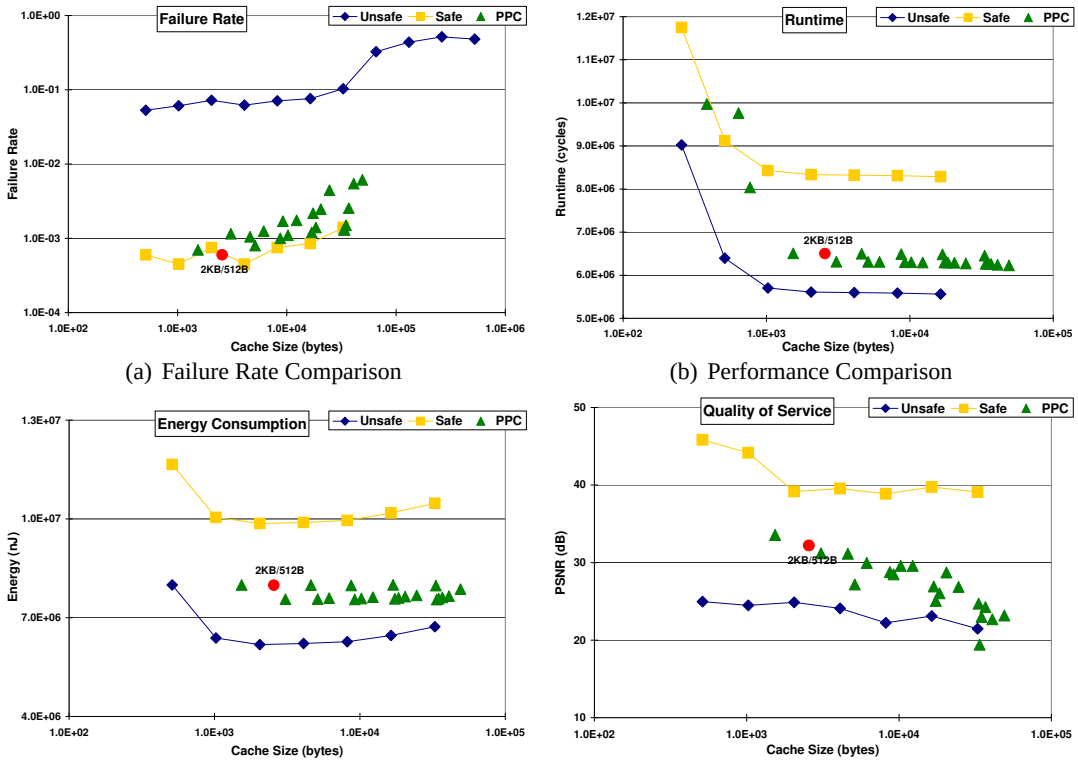
We divide this section into two parts. In the first set of experiments, we compare the failure rate, runtime, energy, area, and QoS for all the benchmarks for Safe, Unsafe and PPC configurations for fixed cache parameters. In the second set of experiments, we demonstrate the applicability of our technique over various cache sizes.

6.1 Impact of Our Approach

Similar to caches in the Intel XScale architecture, the Unsafe cache configuration consists of a 32 KB unprotected cache, the Safe cache configuration consists of a 32 KB protected cache, and the PPC configuration consists of a 32 KB unprotected cache and a 2 KB protected cache.

Fig. 8(a) shows that the PPC configuration achieves failure rates close to those of the Safe cache configuration, and both of these configurations achieve failure rates about 47$\times$ less than that of the Unsafe cache configuration on an average. Fig. 8(a) plots the failure rates achieved by the three cache configurations on a logarithmic scale, and they are normalized to the failure rate of the Unsafe cache configuration. In both the Safe and PPC configurations, failures occur only due to double-bit soft errors since all the single-bit errors are corrected by SEC-DED implementation in protected caches while both single-bit soft errors and double-bit soft errors cause failures in the Unsafe configuration. Fig. 8(a) shows that in some benchmarks (e.g., *susan smoothing*) PPC configuration results in lower failure rate than the Safe cache configuration and in some benchmarks (e.g., *susan corners*) the Safe cache configuration is better. However, in most benchmarks PPC configurations provide failure rates close to those of Safe cache configurations, and much lower than those of Unsafe cache configurations.

Fig. 8(b) shows that PPC configuration achieves the performance close to that of the Unsafe configuration, and incurs only less than 1% performance overhead than the Unsafe configuration on an average while the Safe configuration incurs about 19% performance overhead mainly because of cache access time penalty. Fig. 8(b) plots the runtime for the three cache configurations. The runtimes are normalized to the runtime of the Unsafe cache configuration. The plot shows that as compared to the previously proposed Safe cache configuration, the PPC configuration has on an average 16% performance improvement.



(a) Failure Rate Comparison

(b) Performance Comparison

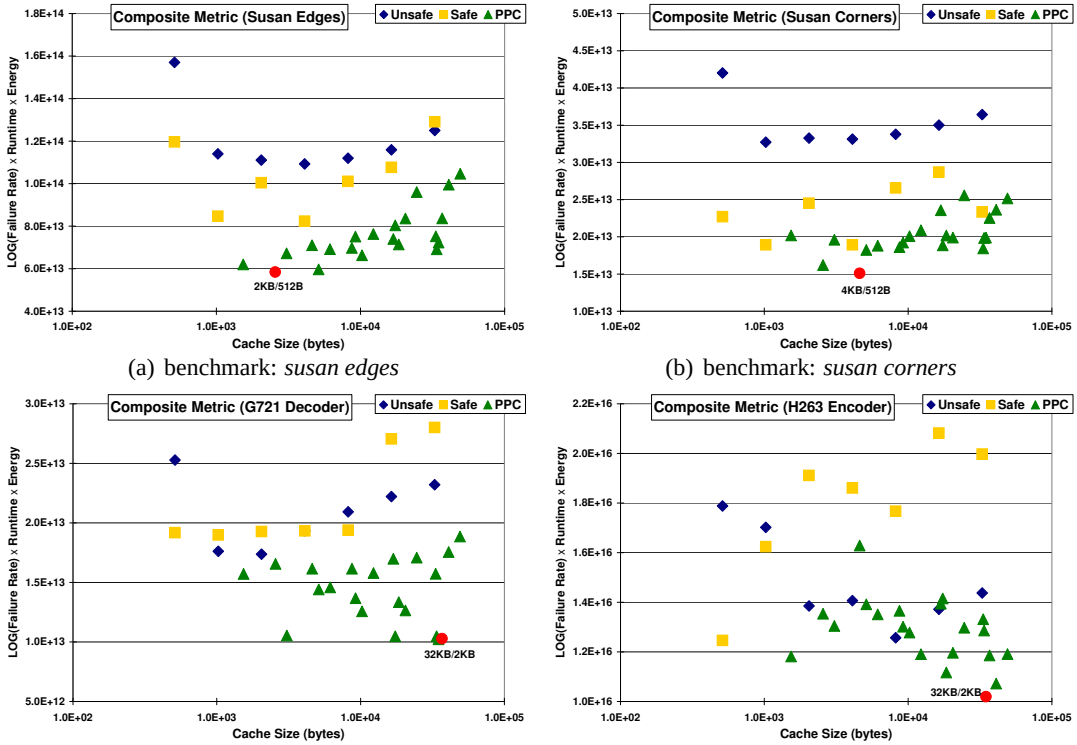
(c) Energy Consumption Comparison

(d) Comparison of Quality of Service

Figure 10: Robustness of Our Approach over Varying Cache Sizee (benchmark - *susan edges*)

Fig. 8(c) shows that PPC configuration consumes less system energy than the Safe configuration. Fig. 8(c) plots the system energy consumption of the three cache configurations normalized to that of the Unsafe cache configuration. The plots show that as compared to the Safe cache configuration, the PPC configuration consumes 8% less energy on an average. As compared to the Unsafe cache configuration, the PPC configuration consumes about 3% more energy while the Safe configuration consumes 13% more energy on an average mainly because of unnecessary protection for multimedia data, which is saved by PPC approach. Note that the energy consumption indicates the energy consumed by the whole system including processor, data cache, memory, and off-chip buses. If we consider only energy consumption of the memory subsystem, PPC configuration is more effective since the processor energy consumptions for each cache configuration are the same. For example, PPC cache configuration consumes the memory subsystem energy 20% less than the Safe configuration.

To simplify the multi-dimensional comparison of various metrics (failure rate, energy, runtime), we define a composite quality metric (CM_{cfg}) for each cache configuration, cfg, as $CM_{cfg} = F_{cfg} \times R_{cfg} \times E_{cfg}$, where F_{cfg} is the logarithmic failure rate, R_{cfg} is the runtime, and E_{cfg} is the energy consumption for a cache configuration cfg. The lower the composite metric, the better the configuration. Fig. 8(d) shows the composite metric for all the three cache configurations for each benchmark. The plot clearly shows that the PPC configuration is a superior design choice. The PPC



(a) benchmark: *susan edges* (b) benchmark: *susan corners*

(c) benchmark: *g721 decoder* (d) benchmark: *h263 encoder*

Figure 11: Comparison of composite metric among *Safe, Unsafe and PPC configurations*

configuration is 40× better than the Unsafe cache configuration, and 2× better than the Safe cache configuration in terms of a composite metric.

Fig. 9(a) shows that PPC configuration results in better QoS than the Unsafe configuration and worse QoS than the Safe configuration. Fig. 9(a) plots the QoS of the three cache configurations normalized to the QoS of the Unsafe cache configuration. The plots show that as compared to the Safe cache configuration, our PPC configuration incurs a QoS penalty of 32%, while as compared to the Unsafe cache configuration our PPC configuration improves the QoS by 54% on an average. Note that QoS values are exaggerated due to accelerated SER in order to observe the failure rates in a reasonable amount of simulation time. Table 1 presents the video quality in PSNR using a benchmark *H263 encoder* for the three configurations according to SER, and clearly shows that the quality degradation of PPC configurations is negligible (less than 5% other than SER = 10^{-9}) compared to that of Safe configuration.

Fig. 9(b) compares the area among the Unsafe, Safe, and PPC cache configurations along the total cache sizes. The plot shows that the area overhead for the PPC cache configuration is smaller than that for the Safe cache configuration since the PPC configuration just implements the SEC-DED algorithm for the only small size of mini cache, which remains the same size for the coding and decoding blocks but reduces the storage size for the control bits. In one specific configuration as 32 KB data cache for the Unsafe, Safe and PPC, which has extra 2 KB protected cache, the area

overhead for the Safe cache configuration compared to the Unsafe cache configuration shows about 22% but the PPC cache configuration can be implemented with just 7% area overhead, which means that we can reduce around 12% area when we build the PPC configuration cache instead of the Safe one.

To summarize, our results demonstrate that as compared to the traditional Unsafe cache configuration, our proposed PPC cache configuration can reduce failure rates by 47×, while incurring only 1% runtime, 3% energy, and 7% area overhead with improved QoS. As compared to the previously proposed Safe cache configuration, our proposed PPC cache configuration can achieve almost the same failure rates while improving both the runtime by 16%, energy by 8%, and area by 12% at the cost of QoS. Thus, PPC architectures allow designers to explore configurations with minimal failure rate by trading-off QoS at minimal power, performance, and area overheads.

6.2 Robustness of Our Approach

In this set of experiments, we demonstrate the effect of varying cache sizes on the effectiveness of PPC architectures with one benchmark *susan edges*. For the Safe and the Unsafe configurations, we use protected and unprotected cache sizes ranging from 512 bytes to 32 KB in exponents of 2. Thus, there are 7 cache configurations in the Safe cache configuration or Unsafe cache configuration. In the PPC cache configuration, we vary the unprotected cache size from 1 KB to 32 KB, while varying the protected cache size from 512 bytes to less than the unprotected cache size. Thus, there are 21 PPC cache configurations. Although we performed these experiments for all possible cache associativities, from direct mapped to fully associative, in exponents of 2, in this article we present results only for cache associativity equal to 4.

6.2.1 Failure Rate Comparison

Fig. 10(a) plots the failure rates of benchmark *susan edges* for Safe (squares), Unsafe (diamonds), and PPC (triangles) cache configurations. The first observation that we make in this graph is that the failure rate for the Unsafe cache configurations initially increases with the cache size, but eventually becomes constant. The initial increase in failure rate is because with increasing cache size, the amount of data in the cache increases, and more data is now vulnerable to soft errors. Once all the application data are cache-resident, increasing the cache size any further does not increase the failure rate. For benchmark *susan edges* the memory footprint (size of unique memory addresses) is 83 KB. Therefore, from 100 KB the failure rate saturates. The second important observation is

Table 1: **Video Quality in PSNR (*dB*) according to SER**

SER	Unsafe	Safe	PPC
10^{-9}	16.39	31.79	19.67
10^{-10}	26.22	33.35	31.89
10^{-11}	32.40	33.46	33.40
10^{-12}	33.39	33.46	33.45

that the failure rate in the PPC cache lies "in-between" the failure rates of the Safe configuration and the Unsafe configuration, and is consistently closer to the Safe configuration. On an average, the failure rate for PPC configuration is 38× better than those for the Unsafe cache configurations. As compared to the Safe cache configurations, PPC configuration is 2.5× higher in the failure rate on an average. The dark circle represents the PPC configuration that was found to be best by our composite metric. It corresponds to a 2 KB unprotected cache and 512 bytes protected cache.

6.2.2 Performance Comparison

Fig. 10(b) plots the runtime (in cycles) for the Safe, Unsafe, and the PPC configurations. The most important observation we make from this graph is that the runtime of PPC cache configurations lies "in-between" the runtimes of the Safe configuration and the Unsafe configuration. The Safe cache configurations suffer from high runtime primarily because of increased cache access time due to ECC implementation. The runtime overhead of PPC cache configurations is on an average 23% less than that of the Safe configurations. As compared to the Unsafe cache configurations, PPC configuration have 13% performance degradation on an average.

6.2.3 Energy Comparison

Fig. 10(c) plots the system energy consumption for the three cache configurations and shows that the energy consumption of the Safe cache configuration is very high, primarily due to the SEC-DED overhead. The most important observation from this plot is that the energy consumption for PPC configurations lies "in-between" the energy consumptions of the Safe configuration and the Unsafe configuration. On an average, PPC configurations consume 23% less energy than Safe cache configurations, and 17% more energy than Unsafe cache configurations.

6.2.4 Quality of Service Comparison

Fig. 10(d) plots the QoS of the Safe, Unsafe and PPC configurations. The graph again shows that the QoS in the PPC configuration is "in-between" the Safe and the Unsafe configurations. On an average, the QoS of the PPC configurations is 15% better than the QoS of Unsafe cache configurations. As compared to the Safe cache configurations, the QoS of the PPC cache configurations is worse by 33% on average. Note that these QoS evaluations result from accelerated SERs. In reality, SER is much less than this accelerated SER, and results in much less quality degradation.

6.2.5 Composite Metric

Fig. 11(a) plots the composite quality metric for all the three cache configurations. The plot shows that Safe cache configurations and the PPC configurations are the contenders for the best design points. The best design point, however, corresponds to the PPC configuration with 2 KB unprotected cache and 512 bytes protected cache. Plots for the other benchmarks, e.g., *susan corners* as in Fig. 11(b), *G721 decoder* as in Fig. 11(c), and *H263 encoder* as in Fig. 11(d), also support our claim of the effectiveness of PPC architectures over both the traditional Unsafe, and the previously proposed Safe cache configurations.

7 Design Space Exploration

Till now, we have demonstrated the superiority of *PPC cache configurations* over the traditional Unsafe cache configurations and the previously proposed Safe cache configurations. There is a definite need to choose the best configuration of a PPC to satisfy the multiple constraints such as the failure rate, power, and performance. However, those constraints are sensitive to cache parameters such as the size and set-associativity, and further the design space exploration of the PPC cache configurations has very high computational requirements.

7.1 Sensitivity of Cache Parameters in PPC

Under the multi-dimensional constraints, considering only one constraint can lead the 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 from Fig. 10(b). However, this configuration is bad in terms of failure rate since it has about 5 times higher failure rate than the best one in the failure rate (512 bytes protected cache in Fig. 10(a)). And also other parameters such as set-associativity cause high variation with respect to performance and energy consumption in our experiments. Thus, 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 a benchmark *susan corners* as will be shown in Fig. 13(a) and Fig. 13(b). Thus, there is a definite need to explore design space and to find out a PPC configuration satisfying multiple constraints.

7.2 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 associativities from 1 to 32-way set associative. Thus, there are only $5 \times 6 = 30$ configurations.

An exhaustive search of these cache configurations requires evaluating failure rate for all the 30 configurations. However, evaluating the failure rate of each configuration requires about 1,000 simulations based on failure rate estimation in Section 5.4. The failure rate can then be estimated as the number of failures that occur in thousand simulations. 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 parallelize-able, it is highly complex in computation. Please note that this is the case, 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.

If a single run of a benchmark takes an hour, finding out the failure rate of the benchmark with 95% confidence and 1% error will take a month. Clearly, the simulation time is far too much for design space exploration. In order to estimate the failure rate, we need to run simulations for each configuration, which takes unreasonable amount of time. So we cannot explore the whole design space with this approach. 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 find out the best protected cache configurations in minimal evaluations.

In this section, we will develop DSE algorithms, for 2 interesting problems.

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

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

7.3 BFExplore - DSE algorithm for Problem 1

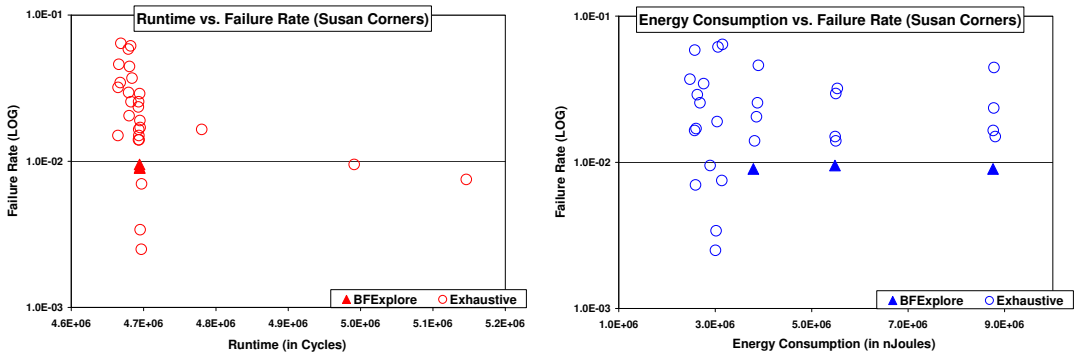
```
BFExplore(S, A, R_constraint, E_constraint)
01: S_min = 1, S_max = 16
02: A_min = 1, A_max = 32
03: for (s = S_min; s ≤ S_max; s = 2 × s)
04:     for (a = A_max; a ≥ A_min; a = a / 2)
05:         R_{s,a} = evalRuntime(s, a)
06:         if (R_{s,a} ≤ R_constraint)
07:             E_{s,a} = evalEnergy(s, a)
08:             if (E_{s,a} ≤ 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()
```

Figure 12: BFExplore Algorithm

For Problem 1, we develop a heuristic algorithm, BFExplore, described in Fig. 12. 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 the smaller size, the lower failure rate. S is a set of sizes of the protected caches 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 the cache configuration with minimal protected cache size with the largest set-associativity (Lines 3-4). If this configuration satisfies the given 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, it repeats the runtime and energy consumption evaluations (Line 10). 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, it means that there is no configuration satisfying the given constraints (Line 16).



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

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

Figure 13: 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)

When we explore the design space for the best configuration in terms of the failure rate with the given constraints such as energy consumption less than 4.0E+6 nJoules and runtime less than 4.7E+6 cycles, the exhaustive approach without any observations require 30 explorations for each configuration as marked with circles and triangles as in Fig. 13(a) and Fig. 13(b), which is about 30,000 evaluations. However, our BFExplore does not evaluate the failure rates for each configuration, which is the huge reduction. In this example, we only evaluate 3 configurations as marked with triangles as in Fig. 13(a) and Fig. 13(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.

7.4 BRExplore - DSE algorithm for Problem 2

For the second problem, how to find the best configuration in terms of runtime satisfying the given failure rate and energy consumption, we present a heuristic algorithm BRExplore as described in Fig. 14. First, our algorithm finds the largest protected cache to satisfy the given failure rate $F_{constraint}$ (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 the larger cache with the 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 9, 17).

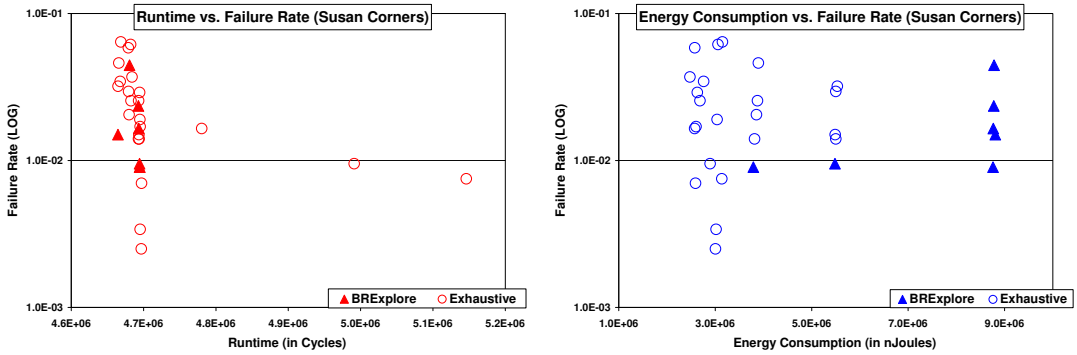
The number of evaluations for our BRExplore is 5,006 ($5 \times 1{,}000$ for failure rate evaluation and 6×1 for energy consumption evaluation) in the worst case where the minimal protected cache

```
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 ≥ S_min; s = s / 2)
04:     F_s = evalFailure(s)
05:     if (F_s ≤ 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 ≥ A_min; a = a / 2)
11:     E_s,a = evalEnergy(s, a)
12:     if (E_s,a ≤ E_constraint)
13:         return {s, a}
14:     endif
15: endfor
16: if (s ≤ S_min)
17:     return NULL
18: else
19:     s = s / 2
20:     return AssocExplore (s, A, E_constraint)
21: endif
endAssocExplore ()
```

Figure 14: BRExplore Algorithm

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 1×1 for energy evaluation). Thus, our proposed BRExplore algorithm can reduce the number of evaluations by up to 30 times as compared to the exhaustive exploration since the exhaustive search requires 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 nJoules and failure rate less than 0.01 (near to the worst case), the exhaustive approach explores the whole 30 points as marked with circles and triangles as in Fig. 15(a) and Fig. 15(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 marked with triangles in Fig. 15(a) and Fig. 15(b). This is approximately 6 times reduction in terms of the number of evaluations.



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

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

Figure 15: 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)

8 Summary

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. Our experimental results showed that our technique reduces runtime by 16% and power consumption by 8% and maintains the same or the better failure rate in comparison with soft error resilient SEC-DED cache at the cost of QoS degradation. As compared to the traditional *unsafe cache configuration*, our proposed *PPC cache configuration* can reduce failure rates by 47×, while incurring only 1% runtime and 3% energy overheads. These results on representative multimedia benchmarks clearly demonstrated the utility of our approach for mitigating soft errors that can affect the safe execution of these applications at a fraction of the cost and energy of a fully secure cache based system.

We also propose heuristic algorithms to find the best configuration among the huge design spaces considering multiple constraints such as failure rate, runtime, and energy consumption. Our experiments demonstrated that our heuristic algorithms, BFExplore and BRExplore, efficiently explore the huge design spaces for our proposed PPC architectures, and reduce the number of evaluations by up to 10,000 times as compared to the exhaustive explorations, for example, in finding the best configuration in terms of failure rate satisfying the given performance and energy consumption constraints.

Our future work includes finer data partitioning, which can increase the effects of our approach, and compiler techniques to intelligently differentiate, partition and map failure non-critical data from failure critical data. Especially, data partitioning algorithms in conjunction with vulnerability

metrics are interesting future works. In addition to data cache, other cache components such as instruction cache and tags will be studied for applying PPC architectures. We are also planning to extend the applicability of our technique to other normal applications concerning reliability as well as power and performance. Further, future work may explore the use of DSE algorithms to adjust PPC configurations at runtime.

References

[1] International technology roadmap for semiconductors 2005 executive summary.

[2] *Cisco 12000 Single Event Upset Failures Overview and Work Around Summary*. http://www.cisco.com/en/US/products/hw/routers/ps167/products_field_notice09186a00801b3df8.shtml, April 2003.

[3] Ghazanfar-Hossein Asadi, Vilas Sridharan, Mehdi B. Tahoori, and David Kaeli. Balancing performance and reliability in the memory hierarchy. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2005.

[4] Hossein Asadi, Vilas Sridharan, Mehdi B. Tahoori, and David Kaeli. Reliability tradeoffs in design of cache memories. In *Workshop on Architectural Reliability in conjuction with MICRO*, 2005.

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

[6] Mark P. Baze, Steven P. Buchner, and Dale McMorrow. A digital CMOS design technique for SEU hardening. *IEEE Trans. on Nuclear Science*, 47(6):2603–2608, Dec 2000.

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

[8] Y. Cai, M. T. Schmitz, A. Ejlali, B. M. Al-Hashimi, and S. M. Reddy. Cache size selection for performance, energy and reliability of time-constrained systems. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2006.

[9] G. Chen, M. Kandemir, M. J. Irwin, and G. Memik. Compiler-directed selective data protection against soft errors. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2005.

[10] J. Gaisler. Evaluation of a 32-bit microprocessor with builtin concurrent error-detection. In *IEEE International Symposium on Fault-Tolerant Computing (FTCS)*, 1997.

[11] 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 (ICS)*, pages 338–347, July 1995.

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

[13] 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.

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

[15] Intel Corporation, http://www.intel.com/design/intelxscale/273473.htm. *Intel XScale(R) Core: Developer's Manual*.

[16] John Bradley, http://www.trilon.com/xv/. *XV*.

[17] Seongwoo Kim and Arun K. Somani. Area efficient architectures for information integrity checking in cache memories. In *International Symposium on Computer Architecture (ISCA)*, pages 246–256, May 1999.

[18] Soontae Kim. Area-efficient error protection for caches. In *IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1282–1287, Mar 2006.

[19] Soontae Kim, N. Vijaykrishnan, Mahmut Kandemir, Anand Sivasubramaniam, and Mary Jane Irwin. Partitioned instruction cache architecture for energy efficiency. *ACM Trans. on Embedded Computing Systems (TECS)*, 2(2):163–185, 2003.

[20] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture (MICRO)*, pages 330–335, 1997.

[21] Jin-Fu Li and Yu-Jane Huang. An error detection and correction scheme for RAMs with partial-write function. In *IEEE International Workshop on Memory Technology, Design and Testing (MTDT)*, pages 115–120, 2005.

[22] Lin Li, Vijay Degalahal, N. Vijaykrishnan, Mahmut Kandemir, and Mary Jane Irwin. Soft error and energy consumption interactions: A data cache perspective. In *International Symposium on Low Power Electronics and Design (ISLPED)*, Aug 2004.

[23] 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 (FTCS)*, 1994.

[24] Dominic Lucchetti, Steven K. Reinhardt, and Peter M. Chen. ExtraVirt: detecting and recovering from transient processor faults. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–8, New York, NY, USA, 2005. ACM Press.

[25] Daniel Lyons. *Sun Screen*. http://www.forbes.com/global/2000/1113/ 0323026a.html, Nov 2000.

[26] Ritesh Mastipuram and Edwin C. Wee. *Soft Errors' Impact on System Reliability*. http://www.edn.com/article/CA454636, Sep 2004.

[27] Subhasish Mitra, Norbert Seifert, Ming Zhang, Quan Shi, and Kee Sup Kim. Robust system design with built-in soft-error resilience. *IEEE Computer*, 38(2):43–52, Feb 2005.

[28] K. Mohr and L. Clark. Delay and area efficient first-level cache soft error detection and correction. In *IEEE International Conference on Computer Design (ICCD)*, pages 88–92, Oct 2007.

[29] Shubhendu S. Mukherjee, Joel Emer, Tryggve Fossum, and Steven K. Reinhardt. Cache scrubbing in microprocessors: Myth or necessity? In *IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 37–42, Mar 2004.

[30] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *International Symposium on Microarchitecture (MICRO)*, Dec 2003.

[31] O. Musseau. Single-event effects in SOI technologies and devices. *IEEE Trans. on Nuclear Science*, 43(2):603–613, Apr 1996.

[32] Gustavo Neuberger, Fernanda De Lima, Luigi Carro, and Ricardo Reis. A multiple bit upset tolerant SRAM memory. *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, 8(4):577–590, Oct 2003.

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

[34] D. K. Pradhan. *Fault-Tolerant Computer System Design*. Prentice Hall, 1996. ISBN 0-1305-7887-8.

[35] Nhon Quach. High availability and reliability in the Itanium processor. *International Symposium on Microarchitecture (MICRO)*, pages 61–69, Sep–Oct 2000.

[36] George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, and David I. August. SWIFT: Software implemented fault tolerance. In *International Smposium on Code Generation and Optimization (CGO)*, pages 243–254, Washington, DC, USA, 2005. IEEE Computer Society.

[37] Jude A. Rivers, Edward S. Tam, Gary S. Tyson, Edward S. Davidson, and Matt Farrens. Utilizing reuse information in data cache management. In *Proceedings of the 12th international conference on Supercomputing (ICS)*, pages 449–456, New York, NY, USA, 1998.

[38] P. Roche, G. Gasiot, K. Forbes, Oapos, V. Sullivan, and V. Ferlet. Comparisons of soft error rate for SRAMs in commercial SOI and bulk below the 130-nm technology node. *IEEE Trans. on Nuclear Science*, 50(6), Dec 2003.

[39] P. Shivakumar and N. Jouppi. CACTI 3.0: An integrated cache timing, power, and area model. In *WRL Technical Report 2001/2*, 2001.

[40] Aviral Shrivastava, Ilya Issenin, and Nikil Dutt. Compilation techniques for energy reduction in horizontally partitioned cache architectures. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, 2005.

[41] Synopsys Inc., Mountain View, CA, USA. *Design Compiler Reference Manual*, 2001.

[42] Seoul National University. PeaCE: Ptolemy extension as Codesign Environment. Technical report, SNU, Nov 2003.

[43] Shuai Wang, Jie Hu, and Sotirios G. Ziavras. On the characterization of data cache vulnerability in high-performance embedded microprocessors. In *IEEE International Conference on Embedded Computer Systems, Architecture, Modeling, and Simulation (SAMOS)*, 2006.

[44] F. Wrobel, J. M. Palau, M. C. Calvet, O. Bersillon, and H. Duarte. Simulation of nucleon-induced nuclear reactions in a simplified SRAM structure: Scaling effects on SEU and MBU cross sections. *IEEE Trans. on Nuclear Science*, 48(6):1946–1952, 2001.

[45] Wei Zhang. Computing cache vulnerability to transient errors and its implication. In *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2005.

[46] Wei Zhang. Replication Cache: A small fully associative cache to improve data cache reliability. *IEEE Computers*, 54(12):1547–1555, Dec 2005.

[47] Wei Zhang, Sudhanva Gurumurthi, Mahmut Kandemir, and Anand Sivasubramaniam. ICR: In-cache replication for enhancing data cache reliability. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 291–300, June 2003.