Mitigating Soft Error Failures for Multimedia Applications by Selective Data Protection *

Kyoungwoo Lee[1], Aviral Shrivastava[2], Ilya Issenin[1], Nikil Dutt[1], and Nalini Venkatasubramanian[1]
[1]Department of Computer Science
School of Information and Computer Sciences
University of California, Irvine, CA 92697, USA
{kyoungwl, isse, dutt, nalini}@ics.uci.edu
[2]Department of Computer Science and Engineering
School of Computing and Informatics
Arizona State University, Tempe, AZ 85281, USA
Aviral.Shrivastava@asu.edu

ABSTRACT

With advances in process technology, soft errors (SE) are becoming an increasingly critical design concern. Due to their large area and high density, caches are worst hit by soft errors. Although Error Correction Code based mechanisms protect the data in caches, they have high performance and power overheads. Since multimedia applications are increasingly being used in mission-critical embedded systems where both reliability and energy are a major concern, there is a definite need to improve reliability in embedded systems, without too much energy overhead. We observe that while a soft error in multimedia data may only result in a minor loss in QoS, a soft error in a variable that controls the execution flow of the program may be fatal. Consequently, we propose to partition the data space into failure critical and failure non-critical data, and provide a high-degree of soft error protection only to the failure critical data in Horizontally Partitioned Caches. Experimental results demonstrate that our selective data protection can achieve the failure rate close to that of a soft error protected cache system, while retaining the performance and energy consumption similar to those of a traditional cache system, with some degradation in QoS. For example, for conventional configuration as in Intel XScale, our approach achieves the same failure rate, while improving performance by 28% and reducing energy consumption by 29% in comparison with a soft error protected cache.

Categories and Subject Descriptors: B.8.1 [Hardware]: Performance and Reliability–Reliability, Testing, and Fault-Tolerance; C.3 [Computer Systems Organization] Special-Purpose and Application-Based Systems – Real-time and embedded systems

GeneralTerms: Design, Experimentation, Performance, Reliability

Keywords: Soft Errors, Horizontally Partitioned Caches, Selective Data Protection, Multimedia Embedded Systems

*This work was partially supported by SRC Contract 1111 and NSF Grant CNS-0615438

CASES'06 October 23–25, 2006, Seoul, Korea.

1. INTRODUCTION

Soft errors, or radiation-induced transient faults, are becoming an overarching challenge in computer system design. A soft error is the phenomenon of temporary change of the state of a logic gate in an integrated circuit under the influence of radiation coming from packaging material or cosmic rays that strike on the silicon device. The occurrence of soft errors may have catastrophic consequences for the system: the application may generate incorrect results, try to access protected memory regions, crash, or go into an infinite loop.

Although the phenomenon of soft errors has been known for a long time, rapidly shrinking feature sizes and lower supply voltages have only now resulted in a significant increase in soft errors and make the presence of soft errors a real design concern. Soft errors can be critical for multimedia embedded systems, especially for mission critical applications such as remote sensing in space, surveillance in hostile territory, or monitoring in highly radioactive environments. It should be noted that these embedded systems may be difficult to reach if it is necessary to recover from a crash. Thus, system failures due to soft errors in such systems may be very difficult, if not impossible, to fix motivating the need for strategies that enhance resilience to soft errors at minimal cost and energy overheads.

The key attribute capturing soft errors is the *Soft Error Rate* (SER), typically measured in *Failures In Time* (FIT), which denotes the number of failures in one billion hours of device operation. Another popular and more intuitive measure of SER is *Mean Time To Failure* (MTTF). A device with 1,000 FIT will have MTTF equal to approximately 114 years. With technology scaling and the trend of increasing capability and complexity of embedded systems, it is clear that the SER in multimedia embedded systems constitute a reliability issue. For example, with a SER of 1,000 FIT per megabit of SRAM at 0.18um technology, the SER in a typical handheld device already has MTTF of approximately 30 years (considered acceptable), but is expected to be less than 10 years (considered high) in the next generation devices [1]. In fact the SER is expected to increase exponentially in semiconductor devices beyond nanotechnology [1, 14, 25, 7].

The occurrence of soft errors is directly proportional to the exposed area of the logic [4]. Since caches are among the largest area contributors in processors, they are most vulnerable to soft errors. Previous research has therefore focused on protecting the caches against soft errors. The use of Error Detection Codes (EDCs) such as a parity check, and Error Correction Codes (ECCs) such as Hamming Codes has

been suggested to protect the data in each line of the cache. While a single bit EDC can be used to detect single-bit errors, *Single-Error Correction and Double-Error Detection* (SECDED) can correct single-bit errors and also can detect double-bit errors transparently. Since most of the soft errors are single-bit errors (the frequency of double-bit errors is about 2-3 orders of magnitude less than that of single-bit errors) SECDED are a very effective architectural technique to protect caches from soft errors.

However, protecting the caches using SECDED has significant power, performance, and area overheads. Every time data from a cache line is read, the ECC check needs to be performed to see if an error occurred in this line. As a result, the error correction logic becomes a part of the time-critical path in cache lookup. Previous research indicates that SECDED implementations can increase the cache access time by up to 95% [15] and power consumption by up to 22% [21]. Embedded systems, which have very stringent power and performance requirements, may not be able to afford such high overheads. Thus, there is a critical need for effective, yet low-overhead architectural techniques to combat soft errors.

Our approach to reducing such overheads is based on the observation 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 a segmentation fault. In such a case, we say that the image pixel is part of *failure non-critical* (FNC) data, while the loop variable is part of *failure critical* (FC) data. While it is important to keep all the failure critical data in a soft error protected cache (that incurs energy and cost overheads for error protection), the failure non-critical data can be kept in a cheaper and simpler cache that is not protected from soft errors.

In this paper, we exploit this property of multimedia applications by employing a modified architectural feature called *Horizontally Partitioned Caches* (HPCs). HPC is a technique in which the processor has multiple (typically two) caches at the same level of memory hierarchy. Each memory address is mapped exclusively to one of the caches at the same level of hierarchy. HPCs are popular in embedded systems, because judiciously partitioning of the application data between the two caches can improve both the performance and the energy consumption [5, 27] of the system.

We extend the concept of HPCs by protecting one of the caches against soft errors by SECDED, while leaving the other one unprotected against soft errors. We partition the application data into failure critical data and failure non-critical data. By mapping the failure critical data into the soft error protected cache, and the failure non-critical data into the unprotected cache, we can keep the failure rates of multimedia applications the same as in an architecture with a single SECDED protected cache while effectively minimizing power and performance overheads at a price of some degradation in QoS. What makes our technique practical and useful is the fact that multimedia applications contain a lot of failure non-critical data that can be easily identified and mapped to a cheaper, unprotected cache. The main contributions of our work are:

- We propose an approach for mitigating failures due to soft errors using the notion of partially protected Horizontally Partitioned Caches.
- We show that as compared to traditional caches, which are not protected from soft errors, our technique can *reduce the failure rate of multimedia applications by two orders of magnitude*, at only 7% increase in runtime and 10% increase in energy consumption.
- Our experiments show that as compared to the previously proposed cache architectures with SECDED protection, our technique reduces the runtime and energy consumption by 30%, while maintaining approximately the same failure rate (due to double-bit errors) with a slight reduction in QoS.

2. RELATED WORK

2.1 Soft Errors

Soft errors, i.e., transient faults, or single-event upsets (SEU) [1], are caused primarily by external radiations in microelectronic circuits, and have been investigated extensively since the late 1970's.

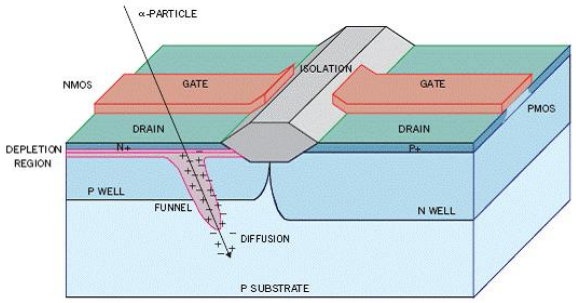


Figure 1: External radiation may induce soft errors

Figure 1 [17] 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 on 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 or soft error. The SER (in FITs) in Figure 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. 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 [7, 31]. In fact, Baumann [1] predicts that the SER in next generation SRAMs will be

[1]These terms are used interchangeably in this paper

up to two orders of magnitude higher. Multiplied by the trend of increasing size of SRAMs in multimedia embedded systems, the SER is becoming an important design concern. As a result solutions to combat the challenge of soft errors have been proposed at various levels of design abstraction:

2.1.1 Packaging and Process Technology Solutions

Radioactive substances present in the packaging material are one of the major sources of radiations that cause soft errors. Therefore techniques like purifying the packaging material, and hardening the semiconductors against radiations [2], have been proposed to reduce the occurrence of soft errors. But interactions between high energy of external radiations and materials cannot be avoided completely.

Process technology solutions, e.g., Silicon-On-Insulator (SOI) [19, 24], to reduce soft errors by raising $Q_{critical}$ by extending the depletion region or increasing the capacitance while maintaining or reducing $Q_{collected}$ have been proposed. However, technology engineering may require the expense of additional process complexity, yield loss, and substrate cost [1].

2.1.2 Processor Architecture Solutions

At the processor architecture level, error detection and correction codes have been widely investigated and implemented as the most effective schemes in order to detect and correct soft errors in memory systems like cache as well as main memory. 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 parity values. Thus, ECC consumes extra energy and incurs performance delay as well as additional area cost [22]. A cache scrubbing technique [18] has been proposed, that reads cache blocks periodically and fixes all single-bit errors, and which can avoid potential double-bit errors. However, this technique may have high performance, power, and area overheads due to SECDED implementation. Lowering supply voltage increases the probability of soft errors. To address this, [16] evaluates the drowsy cache and the decay cache exploiting voltage scaling and shut-down schemes, respectively, in order to decrease the power leakage. [16] also proposes an adaptive error correcting scheme to different cache data blocks, which can save energy consumption by protecting clean data less than dirty blocks. [10] proposes the combined approach of parity and ECC codes to generate the reliable cache system in an area-efficient way. [20] presents an energy-efficient combined method with Hamming and Reed-Solomon codes in order to correct at least double-bit transient faults. However, they all exploit expensive error correcting codes in order to protect all the data. Recently, Cai et al. [4] profile the effects of cache size selection on reliability and power consumption as well as performance by extensively simulating several benchmarks on different cache size configurations. They explore cache parameters to increase reliability while considering power and performance.

The technique presented in this paper is different from all the previous techniques in two distinct ways: 1) we do not try to reduce soft errors, rather try to reduce the failures due to soft errors, and 2) our technique is specifically designed for multimedia applications and can be used in conjunction with these previously proposed architectural techniques.

2.2 Horizontally Partitioned Caches

Horizontally partitioned caches (HPC) [5, 29, 23, 11] have been investigated extensively with perspectives of improvements in performance and energy consumption. [27] presents compiler techniques with several approaches aimed at energy reduction for this architecture. However, there has been no previous work in exploiting this architectural feature to mitigate the impact of soft errors. In this paper we extend and employ the HPC architecture for the above purpose.

3. OUR APPROACH

3.1 Application Data Partitioning

Multimedia applications are typically characterized by Quality of Service (QoS) requirements that define the precision, accuracy and quality of the delivered multimedia streams. Thus, 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 a memory segment violation, or a failure. Other examples of a failure caused by soft errors include a system crash, or the application trapped into an infinite loop.

In the context of multimedia applications, we can define the concept of *failure criticality* of a variable. A variable is failure critical if a soft error in the variable can result in a failure. As per this definition, a loop control variable is failure critical, while the image pixel is failure non-critical.

Although a detailed analysis of all the variables can be performed and the variables can be divided into failure critical and failure non-critical, such fine grain analysis would be very complex. However, for multimedia applications, the assumption that all the multimedia data is failure non-critical, and all the rest of the data is failure critical, is a good approximation. We employ such a separation principle in our approach.

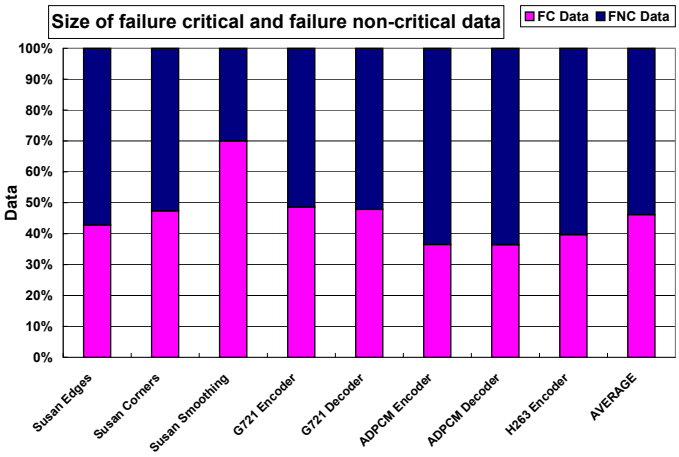


Figure 2: Size of failure-critical and failure-noncritical data in multimedia applications

Figure 2 plots the percentage of failure-critical and failure non-critical data in a number of common multimedia benchmarks, as found by our method. The plot shows that even this simple data partitioning 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. However, even this simple technique of finding the failure non-critical data is quite effective. Also note that it is very easy for the designer to identify the multimedia data, which is typically present as large arrays in the application specification. If the multimedia data is mixed with the control variables, they can be separated by programmers or designers in order to exploit our technique.

3.2 Architecture Model and Approach

We extend the concept of *Horizontally Partitioned Caches* (HPC) for mitigating the impact of soft errors. As shown in Figure 3, we protect one of the caches in the HPC from soft errors by SECDED, while keeping the other one unprotected. We call the cache protected from soft errors as *Soft Error Protected Cache* (SE protected cache), and the traditional unprotected cache as *Soft Error Prone Cache* (SE prone cache).

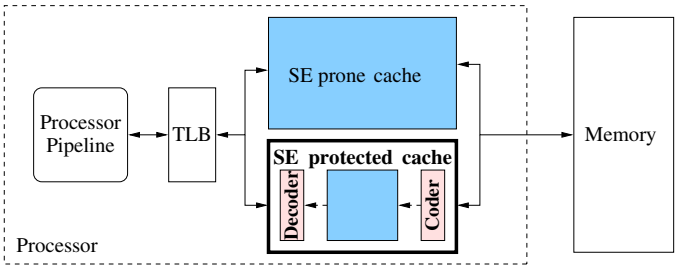


Figure 3: Horizontally Partitioned Cache Architecture for Selective Data Protection

Similar to existing HPC architectures, the memory is divided into pages, and each page has a *Cache Mapping Attribute* or CMA. The 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. The 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, in which one? Therefore, only one cache lookup is performed per access. In the case of the SE protected cache with SECDED, 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. Thus, the SECDED decoder becomes the part of timing critical path and has power and performance overheads.

In order to minimize the performance impact of the HPC architecture, we only consider SE protected cache sizes, such that the total penalty of the SE protected cache due to the SECDED implementation is less than that of the SE prone cache. Since the SE protected cache is smaller, this architecture may lead to performance penalties due to excessive cache misses. Fortunately, the failure critical data that will be mapped to this cache has very good cache behavior.

Figure 4 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 over cache sizes is less than that of the failure non-critical data. This observation implies that the size of the SE protected cache can be reduced without much performance penalty. The reason behind this idea is that the failure critical data

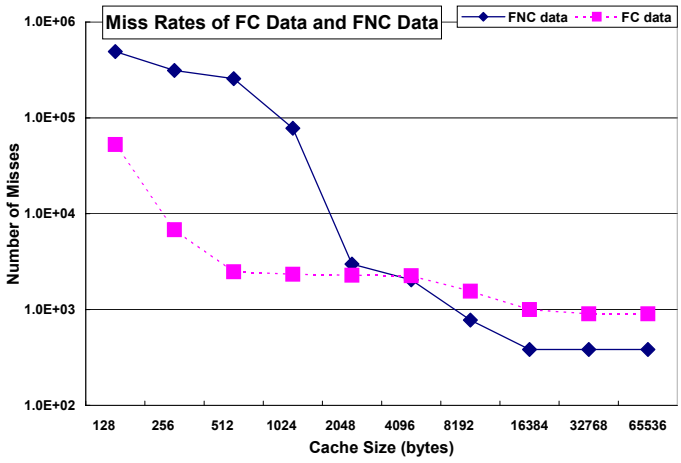


Figure 4: Cache miss rates of failure critical and failure non-critical data

that we have marked is comprised of the local variables, function stack, etc., which have much better cache behavior in locality than that of the multimedia data. As a result, we can achieve low failure rates without much power, performance and energy consumption overheads.

4. EXPERIMENTAL FRAMEWORK

In order to demonstrate the effectiveness of our approach, we have developed a compiler-simulator-analysis framework, as shown in Figure 5.

4.1 Applications

We use multimedia applications from the domain of image processing, audio encoding/decoding, and video processing as our benchmarks. In the domain of image processing applications, we use three image filters, namely *susan edges, susan smoothing,* and *susan corners*, from MiBench [6]. In the domain of audio applications, we use *adpcm encoder* and *adpcm decoder*, from MiBench, and *G721 encoder* and *G721 decoder* from MediaBench [12]. As a representative video application, we select *H263 encoder* from the PeaCE project [30]. These examples form a representative set of typical multimedia applications.

We mark all the arrays that contain the multimedia data with the "_FNCdata_" keyword. We would like to stress again that it is very easy for the application developer to identify the multimedia data. Multimedia data is typically present in large arrays. The compiler groups all the marked arrays and makes them global variables. Along with the executable, the compiler also generates a page mapping file, which lists all the pages containing the FNC data.

4.2 Simulation Platform and Soft Error Injection

We simulate a platform similar to the HP iPAQ h4300 [8]. We tuned the SimpleScalar sim-cache simulator [3] to model this architecture and further modified the simulator to support the HPC cache architecture in Figure 3. The simulator, along with the executable, takes a page mapping file as input, and maps the pages listed in the file to the SE prone cache (the cache which is not SE protected). Everything else is mapped to the SE protected cache.

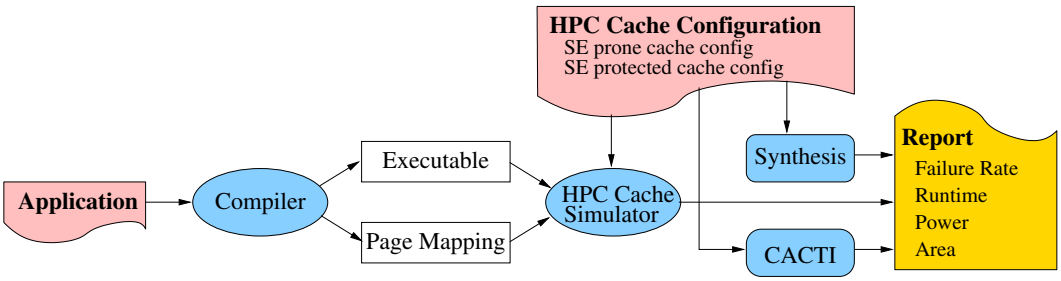


Figure 5: Experimental Framework

The simulator models errors by randomly injecting single-bit and double-bit errors in the SE prone cache. The SE protected cache implements SECDED, and therefore can automatically correct single-bit errors. Consequently, we insert only double-bit errors in the SE protected cache. To be able to accomplish the experiments in a reasonable amount of time, we perform accelerated injection of soft errors in the cache. Our injection module implemented in the SimpleScalar simulator generates single-bit errors at the rate of 10^{-9} single-bit errors per instruction per KB of cache. However, we maintain the accuracy in the relative rates of single-bit and double-bit errors, and therefore inject double-bit errors at the rate of 10^{-11} double-bit errors per instruction per KB of cache. Note that the acceleration of soft error injection does not affect the performance or the energy consumption.

4.3 Cache Configurations

To evaluate the effectiveness of our HPC architectures for reducing the failures due to soft errors, we define three cache configurations: 1) *unsafe*, 2) *safe*, and 3) *HPC*. The *unsafe configuration* denotes a traditional cache architecture (a single SE prone cache). The *safe configuration* represents the previously proposed cache architecture which consists of traditional cache architecture with ECC protection (a single SE protected cache). The *HPC configuration* represents a horizontally partitioned cache architecture with two caches consisting of a SE protected cache and a SE prone cache. We compare our proposed HPC architecture (*HPC cache configuration*) with the *unsafe cache configuration* and the *safe cache configuration* by extensively simulating our benchmarks and comparing the failure rate, runtime, energy consumption, and QoS.

4.4 Modeling

4.4.1 Failures

The occurrence of soft errors in failure critical data may have catastrophic results. In this context, we define the concept of *failure* in a simulation. We term a simulation as a failure if the output of the multimedia processing cannot be opened by the application meant to observe the media data. For example, in the context of image processing, the application *susan edges* generates ".pgm" files, which can be viewed by the graphical viewer *xv*. If the *xv* is unable to open the output file, then we consider the simulation as a failure. A simulation can be a failure due to various reasons, e.g. the output image is not produced, the name of the output image is incorrect, the header of the processed image is not correct, the size of the output image is incorrect.

If the application crashes or generates incorrect results, it is easy to detect a failure. However, to detect a failure when the application goes into an infinite loop, we restrict the number of simulation cycles to be less than 10 times of that of a normal simulation. If the application takes more time, the simulation is stopped and declared a failure. To compute the failure rate, the application is simulated until we achieve 10 failures. We have chosen such a soft error injection rate that we do not get more than a 10% failure rate.

4.4.2 Performance

For performance comparison we estimate the runtime of applications using the statistics generated by the cache simulator. We assume 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. Thus, the runtime for the *unsafe cache configuration*, R_u, is estimated as $R_u = I + A_{SEprone} \times 2 + M_{SEprone} \times 25$, where I is the number of instructions executed, and $A_{SEprone}$ and $M_{SEprone}$ are the number of accesses and misses to the traditional SE prone cache. For the *safe cache configuration*, there is one extra cycle penalty due to the SECDED check. Thus, the runtime for the *safe cache configuration*, R_s, is estimated as $R_s = I + A_{SEprotected} \times 3 + M_{SEprotected} \times 25$, where $A_{SEprotected}$ and $M_{SEprotected}$ are the number of accesses and misses in the SE protected cache. For the *HPC cache configuration*, we keep the SE protected cache size smaller than the SE prone cache size. This method ensures that the access times for both the caches are the same, and equal to 2. Thus, the runtime for *HPC configuration*, R_h, is estimated as $R_h = I + (A_{SEprotected} + A_{SEprone}) \times 2 + (M_{SEprotected} + M_{SEprone}) \times 25$.

4.4.3 Energy Consumption

We estimate the energy consumption of the whole system comprising the processor pipeline, caches, memory and the off-chip buses. We estimate the energy per access of the SE prone cache $E_{SEprone}$ using CACTI [26] cache energy models. The SE protected caches implement SECDED, and therefore consume additional energy for coding (decoding) and writing (reading) extra control bits (effectively increasing the linesize) for each access. To estimate the energy per access for the SE protected cache with the extra coding bits $E_{SEprotected}$, we use CACTI with an increased linesize value parameter. To estimate the energy overhead due to the SECDED coder and decoder, we synthesize them using the Synopsys Design Compiler [28] with lsi10k libraries of 0.5 um 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 $0.18um$ 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. As in [27], 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 [9]. Thus, $E_{proc} = 0.67\ nJ$.

Using the above-mentioned energy models, we estimate the system energy E as, $E = \{(A_{SEprone} \times E_{SEprone}) + A_{SEprotected} \times (E_{SEprotected} + E_{dec}) + (W_{SEprotected} \times E_{cod})\} + \{(M_{SEprone} + M_{SEprotected}) \times (E_{bus} + E_{mem})\} + \{(M_{SEprotected} \times E_{cod}) + (R_{SEprotected} \times E_{dec})\} + \{E_{proc} \times (A_{SEprotected} + A_{SEprone})\}$, where $W_{SEprotected}$ are the number of writes in the SE protected cache, and $R_{SEprotected}$ is the number of replacements in the SE protected cache.

4.4.4 Quality of Service

QoS is very application specific, and is different for different 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 average 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.

5. EXPERIMENTAL RESULTS

To demonstrate the effectiveness in mitigating failures due to soft errors for systems with our *partially protected HPC cache* as compared to systems with the traditional *unsafe cache* and the previously proposed *safe cache*, we divide our experiments into two parts.

The first set of experiments compares the failure rate, runtime, energy, and QoS for all the benchmarks for *safe*, *unsafe* and *HPC cache configurations*.

To explore the effectiveness of *HPC configuration* over various cache sizes, we perform design space exploration as our second set of experiments.

5.1 Effectiveness of HPC

In this subsection we evaluate the effectiveness of *HPC* on all the benchmarks for the fixed cache sizes. Similar to cache sizes in the Intel XScale architecture, the *unsafe cache configuration* consists of a 32 KB SE prone cache, the *safe cache configuration* consists of 32 KB SE protected cache, and the *HPC configuration* consists of a 32 KB SE prone cache and a 2 KB SE protected cache.

Figure 6 plots the failure rates achieved by the three cache configurations on a logarithmic scale and normalized to the failure rate of the *unsafe cache configuration* for comparison. The plot shows that the *HPC configuration* achieves failure rates close to that of the *safe cache configuration*. Both of these configurations achieve failure rates about 47× less than that of the *unsafe cache configuration*. In both the *safe* and *HPC configurations*, failures occur only due to double-bit soft errors, since all the single-bit errors are corrected by ECC in SE protected caches, or while they occur

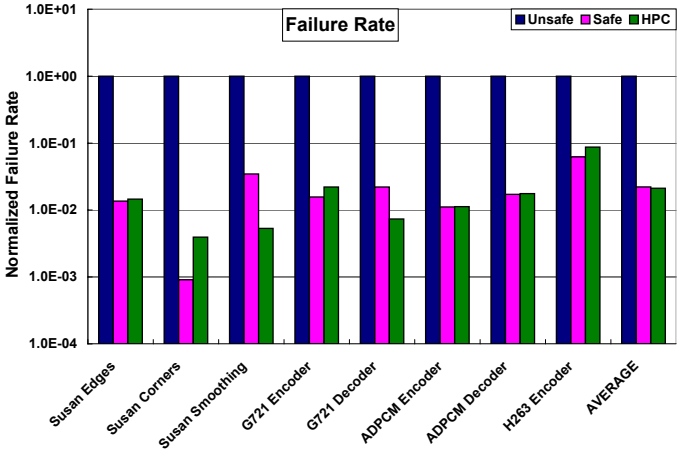


Figure 6: Failure Rate

in failure non-critical data in SE prone cache in the *HPC configuration*. On average, as shown in Figure 6, the *HPC configuration* shows a lower failure rate than that of the *safe cache configuration*. This is because the SE protected cache size is smaller in the *HPC configuration*. As a result, less failure critical data is exposed to soft errors in the *HPC configuration*. In summary, the *HPC configuration* provides failure rates better than or close to those of the *safe cache configuration*.

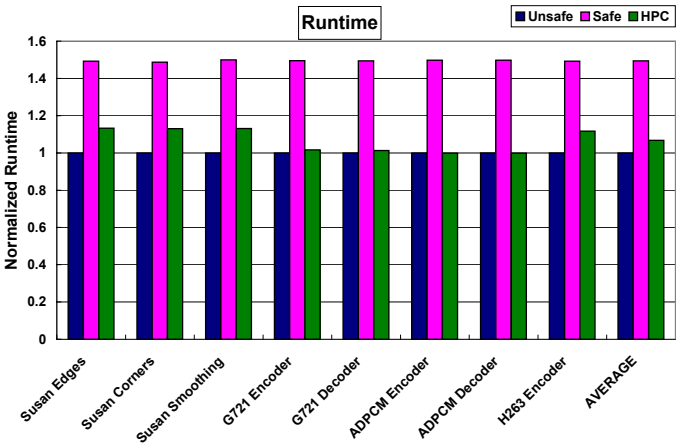


Figure 7: Runtime

Figure 7 plots the runtime for the three cache configurations, in which the runtimes are normalized to the runtime of the *unsafe cache configuration*. Figure 7 clearly shows that the performance overhead of the *HPC cache configuration* is consistently lower than that of the *safe cache configuration*. As compared to the previously proposed *safe cache configuration*, the graph shows that the runtime for the *HPC configuration* takes 28% less on average. This is not only because *safe configurations* incur a cache access time penalty but also because the smaller SE protected cache, the lower cache access time will be, and the separation of data results in less data conflict in *HPCs*. However, as compared to the *unsafe cache configuration*, *HPCs* incur about a 7% runtime penalty on average due to higher miss rate and ECC performance overhead.

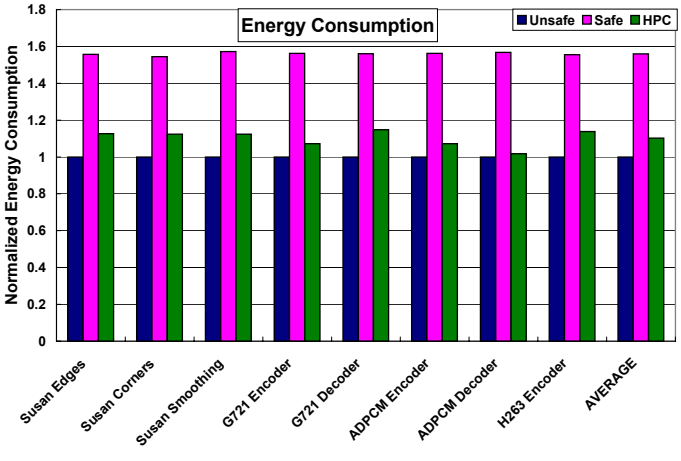


Figure 8: Energy Consumption

Figure 8 displays the energy consumption of the three cache configurations normalized to the energy consumption of the *unsafe cache configuration*. This graph clearly demonstrates that the *HPC cache configuration* consumes less energy due to the absence of the overhead in the energy spent in coding and decoding FNC data in the *safe cache configuration*. The plot shows that as compared to the *safe cache configuration*, the *HPC configuration* consumes 29% less energy on average. As compared to the *unsafe cache configuration*, the *HPC configuration* consumes 10% more energy while the *safe configuration* has more than 50% of energy overhead due to protection for all the data.

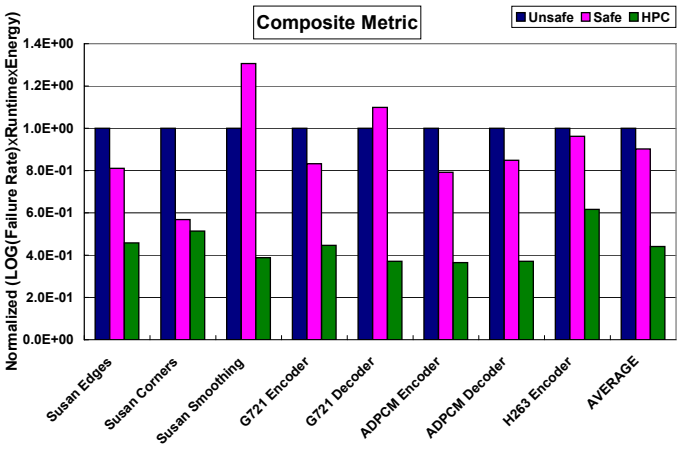


Figure 9: Composite Metric: LOG(Failure Rate) × Runtime × Energy

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 as $CM_{cfg} = LOG(F_{cfg}) \times R_{cfg} \times E_{cfg}$, where F_{cfg} is the failure rate, R_{cfg} is the runtime, and E_{cfg} is the energy consumption of cache configuration cfg. The lower the composite metric, the better the configuration. Note that the failure rate differences can be up to several hundred times while the differences of the energy consumption and the performance are within 50%, so we use the failure rate on the log scale ($LOG(F_{cfg})$) to give the fair weights to each metric. Figure 9 shows the composite metric for all the three cache configurations for each benchmark. The plot clearly shows that the *HPC configuration* is a superior design choice for all benchmarks. Approximately, the *HPC configuration* is up to 3× better than the *unsafe cache configuration* for *ADPCM encoder* benchmark, and up to 4× better than the *safe cache configuration* for *susan smoothing* in terms of a composite metric.

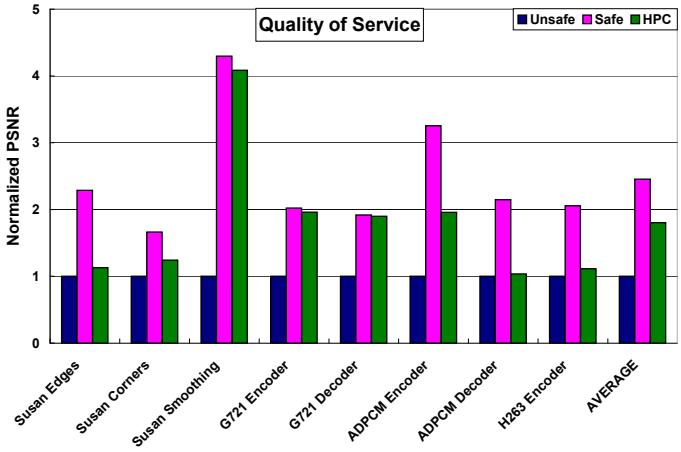


Figure 10: Quality of Service

Figure 10 plots the QoS comparison among the three cache configurations by normalizing each PSNR value to that of the *unsafe cache configuration*. The plot shows that as compared to the *safe cache configuration*, our *HPC cache configuration* incurs a QoS penalty of 26%, which is calculated as $(P_s - P_h)/P_s$, where P_s and P_h are the normalized PSNR values for the *safe cache configuration* and the *HPC cache configuration*, respectively. When compared to the *unsafe cache configuration*, our *HPC configuration* improves the QoS by an average 80%. Note that we perform experiments using accelerated SER. In reality the SER is much less. Consequently, the failure rate and QoS results of our experiments are not correct in the absolute sense; they just have relative accuracy. Thus, even though the QoS degrades by 26%, the actual error in the image will be very small in absolute terms. For example, if 5 pixels were getting modified in 100 images earlier, then after using our technique about 6 pixels will be erroneous in 100 images, which is still insignificant.

Although a similar argument is applicable to reduction in failure rate, due to the catastrophic nature of failures, even a slight increase is important. Our technique is able to reduce the failure rate by 45× at less than 1.3× degradation in QoS. We believe this result is extremely good.

To summarize, our results demonstrate that as compared to the traditional *unsafe cache configuration*, our proposed *HPC cache configuration* can reduce the failure rate by 45×, while incurring only 7% runtime, and 10% energy overhead. As compared to the previously proposed *safe cache configuration*, our proposed *HPC cache configuration* can achieve almost the same failure rates while improving both the runtime by 28% and energy by 29%, but trading off QoS by 26%.

5.2 Design Space Exploration

We have shown that *HPC cache configurations* are a su-

perior design choice over all benchmarks for a specific cache size. In this section, we investigate the impact of changing cache sizes on the three configurations, i.e., *unsafe*, *safe* and *HPC configurations*. There may be several cache sizes in each configuration. For the *safe* and the *unsafe configurations*, we use SE protected and SE prone cache sizes ranging from 512 B to 32 KB in exponents of 2. Thus, there are 7 cache configurations in the *safe cache configuration* and *unsafe cache configuration*. For the *HPC cache configuration*, we vary the SE prone cache size from 1KB to 32 KB, while varying the SE protected cache size from 512 B to less than the SE prone cache size [2]. Thus, there are 21 *HPC cache configurations* in all. Although we performed these experiments for all possible cache associativities, from direct mapped to fully associative, in exponents of 2, in this paper we present results only for 4-way set associative caches. More details are in our technical report [13].

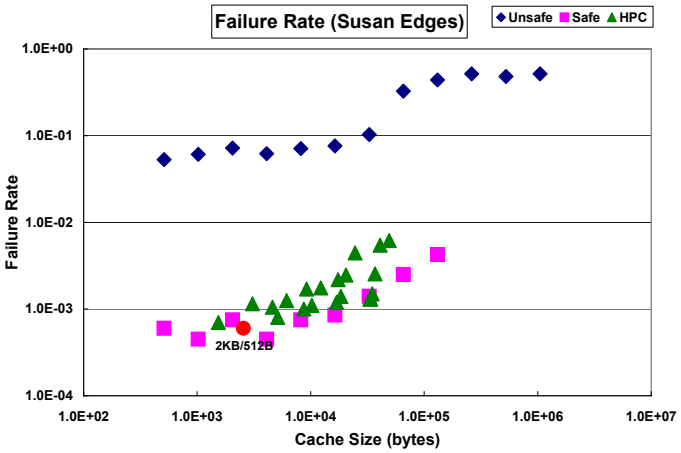


Figure 11: Failure Rate Comparison

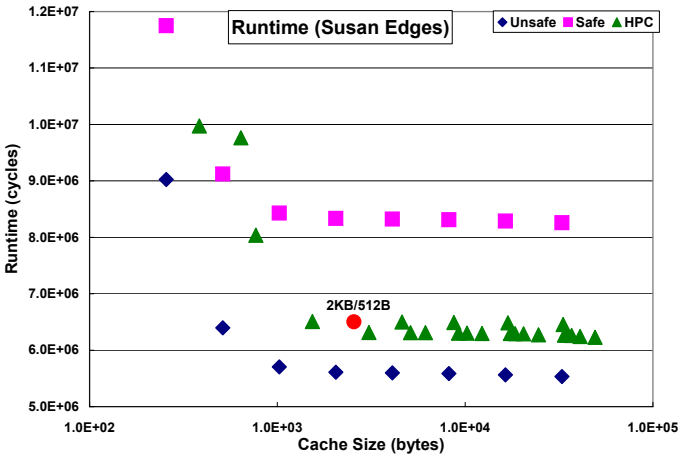


Figure 12: Performance Comparison

[2]Our experiments show that the optimal size for all benchmarks is no more than 32 KB

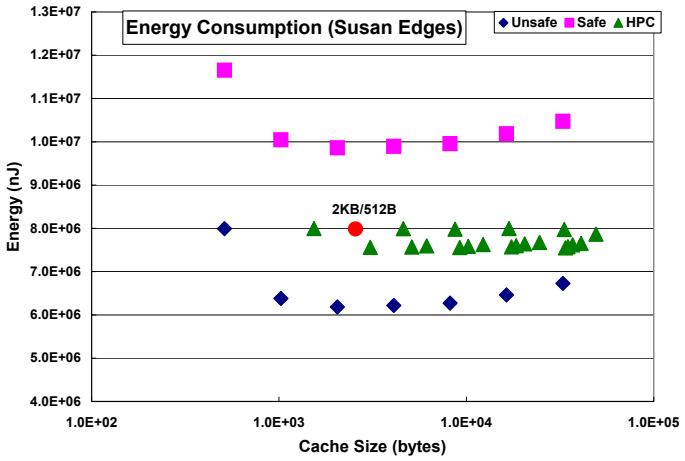


Figure 13: Energy Comparison

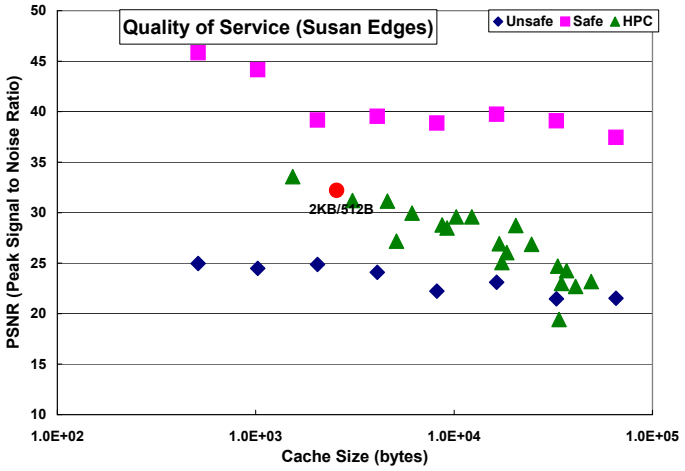


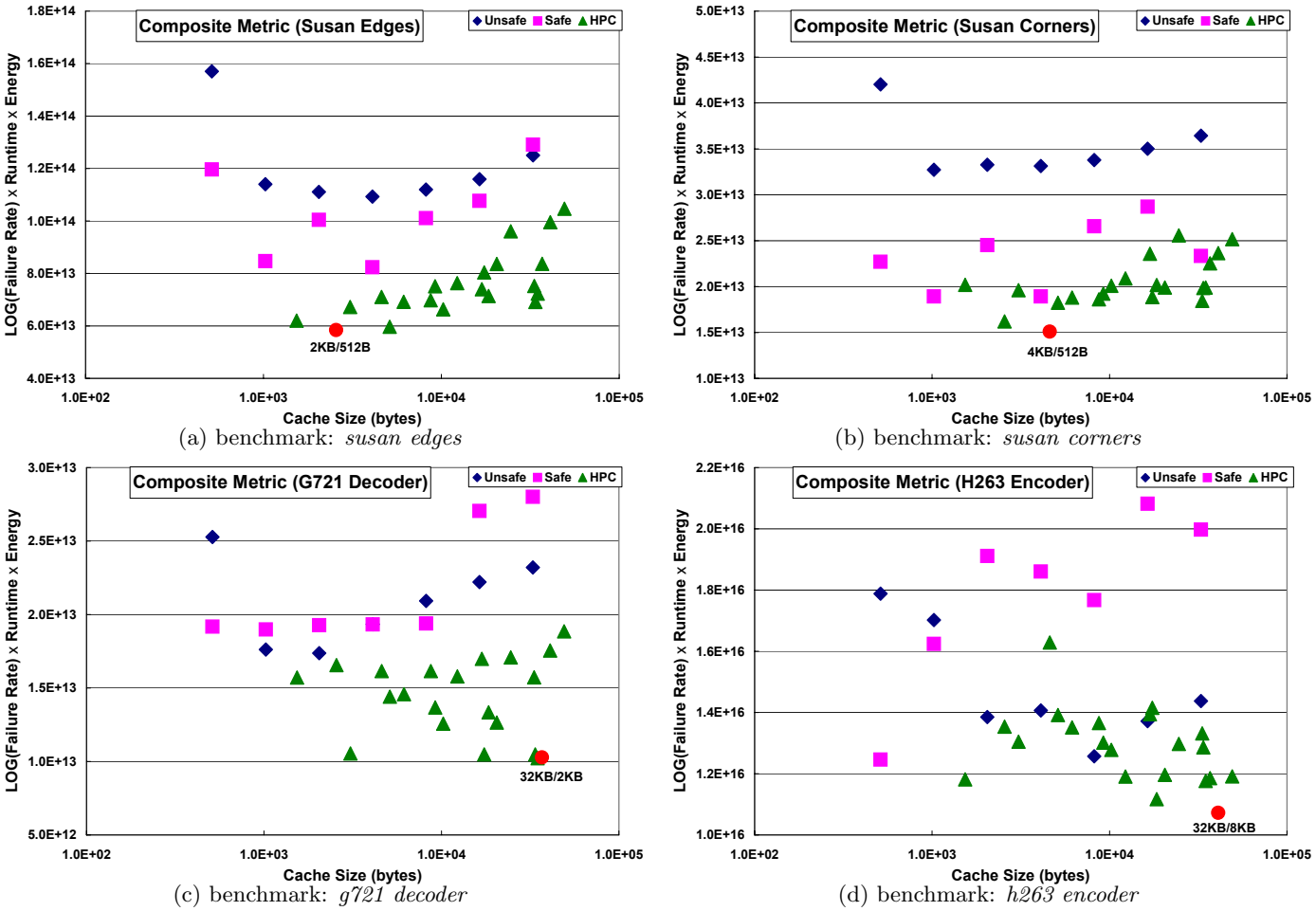
Figure 14: Comparison of quality of service

5.2.1 *Failure Rate Comparison*

Figure 11 plots the failure rates of benchmark *susan edges* for *safe (squares)*, *unsafe (diamonds)*, and *HPC (triangles) cache configurations* for different total cache sizes.

The first observation we make in this graph is that the failure rate for the *unsafe configuration* initially increases with the cache size, but eventually becomes constant (in order to make it clear, we present failure rates for up to 1 MB of cache size in the case of the *unsafe configuration* in Figure 11). 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 is 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 48 KB. Therefore after 64 KB the failure rate saturates as shown clearly in Figure 11.

The second important observation is that the failure rate in the *HPC configuration* is consistently close to the *safe configuration*. On average, the failure rate for *HPC configurations* is 38× better that for the *unsafe cache configurations*. As compared to the *safe cache configuration*, the *HPC configuration* presents 2.5× higher failure rate for *su-*



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

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

Figure 15: Comparison of composite metric among *safe, unsafe and HPC configurations*

san edges benchmark. The dark circle represents the *HPC configuration* that was found to be best by our composite metric. It corresponds to a 2KB SE prone cache and 512 B SE protected cache.

5.2.2 Performance Comparison

Figure 12 plots the runtime (in cycles), increasing with a decrease of cache size mainly due to higher cache misses, for the *safe*, *unsafe*, and *HPC configurations*. The *safe cache configurations* suffer from high runtime primarily because of increased cache access time. The runtime of *HPC cache configurations* is on average 24% less than that of the *safe configurations*. As compared to the *unsafe cache configurations*, *HPC configurations* have 13% runtime degradation but the *safe configurations* lose about 48% performance. Thus, the *HPC configuration* can provide the better performance than the *safe cache configuration* with comparable failure rates.

5.2.3 Energy Comparison

Figure 13 plots the energy consumption for the three cache configurations, which is decreasing and then increasing with an increase of cache sizes. The first decrease comes from lowering miss rates and the second increase results from high access energy overhead for larger cache sizes. The plot shows that the energy consumption of the *safe cache configuration* is high, primarily due to the SECDED decoder and coder overhead. On average, *HPC configurations* consume 24% less energy than *safe cache configurations*, and 17% more energy than *unsafe cache configurations*. Again, *HPC cache configurations* are better in terms of both performance and power than *safe configurations* with similar failure rate.

5.2.4 Quality of Service Comparison

Figure 14 plots the PSNR in dB as a QoS metric of the *safe*, *unsafe* and *HPC configurations*. Increasing the cache size reduces QoS since SER is proportional to the cache size. This effect explains the lower QoS, for example, in 32 KB SE prone *HPC cache* since it has additional soft errors (double-bit errors) in a SE protected cache ranging from 512 B to 16 KB. The graph again shows that the QoS in the *HPC configuration* is "in-between" the *safe* and the *unsafe configurations*. On an average, the QoS of the *HPC configurations* is 13% better than the QoS of *unsafe cache configurations*. As compared to the *safe cache configurations*, the QoS of the *HPC cache configurations* is smaller by 33% on average.

5.2.5 Composite Metric Comparison

Figure 15(a) plots the composite quality metric for all three cache configurations. The plot shows that the *safe cache configurations* and the *HPC configurations* are the

contenders for the best design points. The best design point, however, corresponds to the *HPC configuration* with 2 KB SE prone cache and 512 B SE protected cache. We get similar plots for the other benchmarks, e.g., *susan corners* in Figure 15(b), *G721 decoder* in Figure 15(c), and *H263 encoder* in Figure 15(d). They show that our proposed *HPC cache configurations* mark the best design points for all multimedia benchmarks.

6. SUMMARY

In this paper 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 introduce selective data protection using Horizontally Partitioned Caches and propose a simple technique for mapping data into such caches geared towards their use in multimedia applications. Our experimental results show that our technique reduces runtime and power consumption by around 30% and maintains the same or the comparable failure rate in comparison with soft error resilient SECDED cache while suffering only a slight degradation in QoS.

In the first set of experiments we demonstrated that as compared to the traditional *unsafe cache configuration*, our proposed *HPC cache configuration* can reduce failure rates by 45×, while incurring only 6.7% runtime and 10.2% energy overhead. As compared to the previously proposed *safe cache configuration*, our proposed *HPC cache configuration* can achieve almost the same failure rates while improving the runtime by 28.6% and energy by 29.3%, but trading off QoS by 26%. The second set of experiments showed that the best *HPC cache configuration*, in terms of the suggested composite metric, can reduce the failure rate by 45× compared to the best *unsafe cache configuration*, while incurring 16.1% runtime and 28.4% energy overhead. As compared to the best *safe cache configuration*, the best *HPC cache configuration* can achieve the lower failure rate while improving the runtime by 21.9% and energy by 19.3%, but trading off QoS by 18.5%. These results on representative multimedia benchmarks clearly demonstrate 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.

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. We are also working to extend the applicability of our technique to other general applications concerning reliability as well as power and performance.

7. REFERENCES

[1] R. Baumann. Soft Errors in Advanced Computer Systems. *IEEE Design and Test of Computers*, 22(3):258–266, May–June 2005.

[2] M. P. Baze, S. P. Buchner, and D. McMorrow. A Digital CMOS Design Technique for SEU Hardening. *IEEE Trans. on Nuclear Science*, 47(6):2603–2608, Dec 2000.

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

[4] 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 *ASP-DAC*, 2006.

[5] A. Gonz'alez, C. Aliagas, and M. Valero. A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality. In *ICS'95*, pages 338–347, July 1995.

[6] 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*, pages 10–22, Dec 2001.

[7] 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.

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

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

[10] S. Kim. Area-Efficient Error Protection for Caches. In *DATE'06*, pages 1282–1287, Mar 2006.

[11] S. Kim, N. Vijaykrishnan, M. Kandemir, A. Sivasubramaniam, and M. J. Irwin. Partitioned Instruction Cache Architecture for Energy Efficiency. *Trans. on Embedded Computing Sys.*, 2(2):163–185, 2003.

[12] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.

[13] K. Lee, A. Shrivastava, I. Issenin, N. Dutt, and N. Venkatasubramanian. Horizontally Partitioned Caches to Reduce Failures due to Soft Errors for Mission-Critical Multimedia Embedded Systems. Technical report, UCI, 2006.

[14] W. Leung, F.-C. Hsu, and M.-E. Jones. The Ideal SoC Memory: 1T-SRAM. In *13th IEEE SoC/ASIC Conference*, pages 32–36, Sep 2000.

[15] 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 (MTDT'05)*, pages 115–120, 2005.

[16] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Soft Error and Energy Consumption Interactions: A Data Cache Perspective. In *ISLPED*, pages 132–137, Aug 2004.

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

[18] S. S. Mukherjee, J. Emer, T. Fossum, and S. K. Reinhardt. Cache Scrubbing in Microprocessors: Myth or Necessity? In *PRDC'04*, 2004.

[19] O. Musseau. Single-Event Effects in SOI Technologies and Devices. *IEEE Trans. on Nuclear Science*, 43(2):603–613, Apr 1996.

[20] G. Neuberger, F. D. Lima, L. Carro, and R. Reis. A Multiple Bit Upset Tolerant SRAM Memory. *ACM Trans. on Design Automation of Electronic Systems*, 8(4):577–590, Oct 2003.

[21] R. Phelan. Addressing Soft Errors in ARM Core-based Designs. Technical report, ARM, 2003.

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

[23] J. A. Rivers, E. S. Tam, G. S. Tyson, E. S. Davidson, and M. Farrens. Utilizing Reuse Information in Data Cache Management. In *ICS'98*, pages 449–456, 1998.

[24] 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):2046–2054, Dec 2003.

[25] N. Seifert, D. Moyer, N. Leland, and R. Hokinson. Historical Trend in Alpha-Particle induced Soft Error Rates of the Alpha Microprocessor. In *IEEE 39th Annual International Reliability Physics Symposium*, 2001.

[26] P. Shivakumar and N. Jouppi. CACTI 3.0: An Integrated Cache Timing, Power, and Area Model. In *WRL Technical Report 2001/2*, 2001.

[27] A. Shrivastava, I. Issenin, and N. Dutt. Compilation Techniques for Energy Reduction in Horizontally Partitioned Cache Architectures. In *CASES'05*, 2005.

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

[29] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun. A Modified Approach to Data Cache Management. In *MICRO 28*, pages 93–103, Los Alamitos, CA, USA, 1995.

[30] S. N. University. PeaCE: Ptolemy extension as Codesign Environment. Technical report, SNU, Nov. 2003.

[31] 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.