Exploiting Soft Redundancy for Error-Resilient On-Chip Memory Design

Shuo Wang and Lei Wang
Department of Electrical and Computer Engineering
University of Connecticut
371 Fairfield Road, U-2157
Storrs, CT 06269
{shuo.wang, leiwang}@engr.uconn.edu

ABSTRACT

Memory design is facing the upcoming challenges due to a combination of technology scaling and higher levels of integration and system complexity. In particular, memory circuits become vulnerable to transient (soft) errors caused by particle strikes and process spread. In this paper, we propose a new error-tolerance technique referred to as the *soft redundancy* for on-chip memory design. Program runtime variations in memory spatial locality cause wasted memory spaces occupied by the irrelevant data. The proposed soft-redundancy allocated memory exploits these wasted memory spaces to achieve efficient memory access and effective error protection in a coherent manner. Simulation results on the SPEC CPU2000 benchmarks demonstrate 73.7% average error protection coverage ratio on the 23 benchmarks, with average of 52% and 48.3% reduction in memory miss rate and bandwidth requirement, respectively, as compared to the existing techniques.

Categories and Subject Descriptors: B.3.4 [Memory Structures]: Reliability, Testing, and Fault-Tolerance; C.4 [Performance of Systems]: Fault tolerance

General Terms: Design, Reliability

Keywords: Memory System, Error Tolerance, Cache Space Utilization

1. INTRODUCTION

Technological scaling has driven the design of integrated circuits with exploding system complexity and performance improvement. With the continuing trend towards nanoscale integration, nanometer devices are approaching their physical limits, and precise control over feature size and design uniformity becomes extremely difficult [1]. Design of high-performance integrated systems is thus compelled to contend with a wide range of unmanageable performance spread and reliability degradation introduced by the underlying technology challenges.

ICCAD'06, November 5-9, 2006, San Jose, CA.

These obstacles to the semiconductor roadmap are particularly noticeable in memory circuits such as on-chip caches. Technology scaling reduces the capacitance, supply voltages, and hence the information-bearing charges at storage nodes. Consequently, memory circuits become vulnerable to transient (soft) errors caused by particle strikes. Timing complexity is also a problem that further complicates memory design. Memory access relies on closely coupled clock waveforms to perform latching, gating, dynamic timing, and sophisticated cycle borrowing. However, clock skew coupled with signal delay variations introduces timing-related transient errors that affect memory robustness. In addition, minimum-geometry devices that build bulk of memory are very sensitive to variations in process, supply voltages and temperature.

The frustrating design complexity and reliability degradation press for new capabilities of error tolerance for on-chip memory design. Traditional approaches include radiation-hardened memory structures [2], double or triple memory redundancy [3], and code checking algorithms [4]. Integrating these techniques into high-performance on-chip memory presents a significant challenge due to the severe constraints on area and timing margins. On a parallel path, research in cache microarchitecture has been focusing on efficiently utilizing memory resources, reducing the bandwidth requirement of data fetching, while not inducing too much overheads or lowering spatial locality. Sub-blocked caches [5] reduce memory traffic by transferring only a single sub-block on a cache miss. Some alternatives [6], [7] dynamically adapt the block size to the spatial locality of the program. Other techniques [8], [9] adopt history based fetching on the granularity of sub-blocks or words. It should be pointed out that none of the above techniques address both memory robustness and access efficiency.

In this paper, we propose a new error-tolerance technique, referred to as the *soft redundancy*, for on-chip memory design. We observe that program runtime variations in memory spatial locality cause many irrelevant data being fetched into the memory and thus waste memory spaces. By freeing these memory spaces, the program creates transient (soft) memory redundancy that can be exploited for tolerance of transient errors. Through adaptively balancing cache resources among different computing tasks, the proposed technique is capable of achieving efficient memory access and effective error protection in a coherent manner. To assist the implementation of our technique, we develop a design methodology that enables optimal tradeoffs between hard-

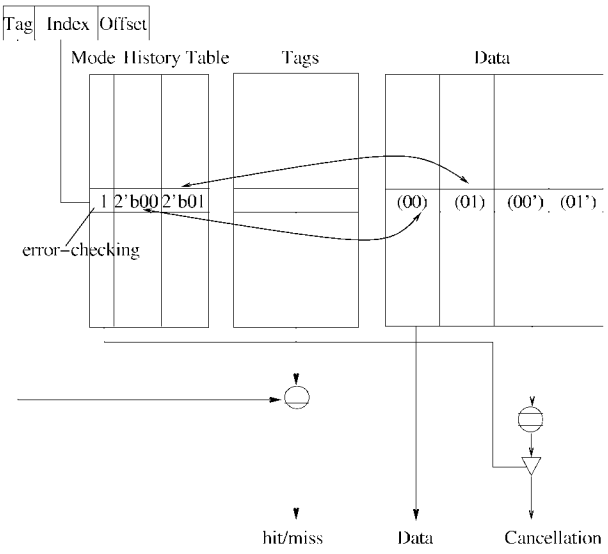

Figure 1: Microarchitecture of soft-redundancy allocated memory.

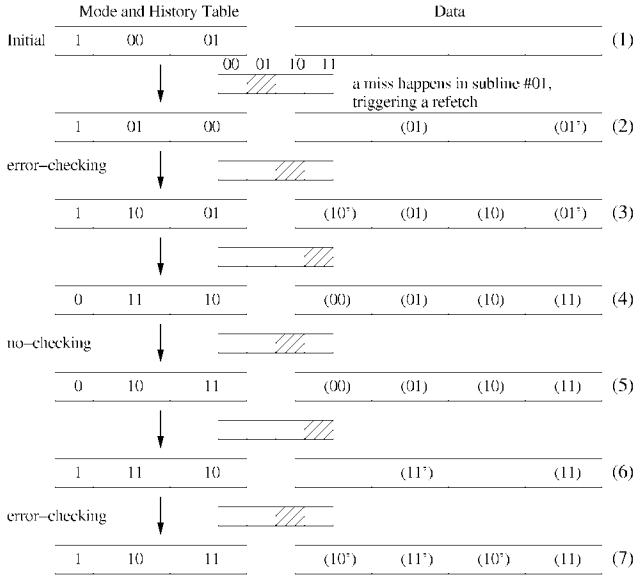

Figure 2: An example of soft redundancy allocation.

ware overheads and error tolerance. Simulation results on the SPEC CPU2000 benchmarks [12] demonstrate 73.7% average error protection coverage ratio on the 23 benchmarks, with average of 52% and 48.3% reduction in memory miss rate and bandwidth requirement, respectively, as compared to the existing techniques.

In section 2, we develop the soft-redundancy allocated memory microarchitecture for effective error protection. In section 3, we formulate a design methodology for optimizing the performance of soft redundancy allocation. Section 4 presents a statistical analysis on the error tolerance and comparison to the existing techniques. In section 5, we evaluate the performance of the proposed technique.

2. SOFT-REDUNDANCY ALLOCATED MEMORY

In most processor architectures, the size of cache lines is fixed, while the spatial locality varies during runtime under different programs. There are always some irrelevant data being fetched in the cache lines, wasting memory spaces.

In this section, we discuss the soft-redundancy allocated memory that achieves efficient memory utilization and effective error protection. The proposed technique aims at freeing the cache line spaces taken by the irrelevant data. The released spaces provide soft (transient) redundancy that is hidden in the cache lines due to variations in memory spatial locality. Thus, instead of fetching the entire cache lines containing many irrelevant data, the proposed technique selectively fetches useful data to the sublines within a cache line. The unused sublines are utilized to store a redundant copy of the useful data, thereby providing effective error protection.

2.1 Soft Redundancy Allocation

Figure 1 shows an example of memory microarchitecture employing the proposed soft redundancy allocation. Each cache line is divided into multiple sublines (e.g., four sublines in Figure 1 for the purpose of demonstration). A history table is introduced to record subline access status. Note that the granularity, i.e., the subline size, is a key parameter that affects several performance metrics, such as the distribution of soft redundancy, the coverage of error protection, and the miss rate of memory access. A general solution on determining the subline size and the corresponding history table is given in section 3.

In the example shown in Figure 1, the history table uses four bits per cache line to keep track of the two most recently replaced sublines. For example, the value $4'b00\_01$ shown in the history table indicates that the two sublines $2'b00$ and $2'b01$ are fetched most recently due to cache misses. In the following memory accesses, a miss occurring in the subline $2'b00$ will not change the history table; whereas a miss in the subline $2'b01$ will update the history table to $4'b01\_00$. If a miss occurs in any other sublines, e.g., subline $2'b10$ or $2'b11$, the history table will be updated to $4'b10\_00$ or $4'b11\_00$, respectively.

An example of memory access sequence using the soft redundancy allocation is shown in Figure 2. Each cache line can be operated in one of the two allocation modes, *error-checking* and *no-checking*. The mode switch is controlled by the information stored in the history table. The detailed operation and soft redundancy allocation is explained below in reference to the steps in Figure 2.

Initially, the cache line is in the error-checking mode. The flag bit is set to "1" indicating this mode. The history table stores "00-01" for two default sublines "00" and "01". Note that the default sublines can be chosen arbitrarily. Assume that the first miss in this cache line triggers the fetch of data to replace those in subline "01" (step 2 in Fig 2). Since the subline "01" is already in the history table, the allocation mode will remain at error-checking, but the content of history table will be updated to "01-00", indicating that subline "01" is the most recently replaced subline. The cache will only fetch data to subline "01", not the entire cache line. Meanwhile, an unused space, e.g., subline "11", is assigned to save a redundant copy of the new data in subline "01".

Assume the next miss causes the replacement of subline

Table 1: Algorithm of history based soft redundancy allocation.

```
DOWHILE initiation
    subline division
    history_table initiation
    allocation_mode initiation
    confidence_counter initiation
ENDDOWHILE

DOWHILE a miss occurs
    IF the to-be-refetched subline hit history_table
        IF confidence_counter!=CONFIDENCE_MAX
            confidence_counter++
        ENDIF
        ELSE
            IF confidence_counter! = 0
                confidence_counter--
            ENDIF
        ENDIF

    update history_table

    IF confidence_counter==CONFIDENCE_THRESHOLD
        allocation_mode switches
    ENDIF
ENDDOWHILE

DOWHILE refetch
    CASE(allocation_mode)
        error_checking:
            assign redundancy pairs
        no_checking:
            cancel redundancy pairs
    ENDCASE
ENDDOWHILE
```

"10" (step 3). Since this subline is not listed in the history table, the history table will be updated to "10-01". Since this is the first time of a miss whose entry is not in the history table, the allocation mode will remain at error-checking mode. New data is fetched into subline "10" and a redundant copy is stored at an unused subline determined by Table 2.

Next, assume the subline "11" is to be replaced because of a new miss (step 4). The history table is thus changed to "11-10". Since this is the second time of a miss whose entry is not in the history table, statistically we can conclude that the program is becoming less predictable with respect to the memory spatial locality during this period. Thus, the allocation mode will be switched to no-checking. The entire cache line will be fetched and no subline will have a redundant copy.

If the subsequent miss occurs in subline "10" (step 5), the program will still fetch the entire cache line and update the history table to "10-11". But because subline "10" is already in the history table, the hit in the history table establishes some confidence in the location where the program is most likely to access.

The allocation mode will switch back to error-checking provided the next miss is also a hit in the history table. For example, a miss in subline "11" leads to update of history table to "11-10" (step 6), and this is the second consecutive hit in the history table. Considering this event as an evidence of sufficient memory spatial locality, we can assume the subsequent memory accesses are likely to be directed to those sublines recorded in the history table. Thus, new data

Table 2: An example of predefined subline pairs for redundancy.

Subline numbers in history table	Subline pairs
"00" and "01"	("00", "10"), ("01", "11")
"00" and "10"	("00", "01"), ("10", "11")
"00" and "11"	("00", "01"), ("11", "10")
"01" and "10"	("01", "00"), ("10", "11")
"01" and "11"	("01", "00"), ("11", "10")
"10" and "11"	("10", "00"), ("11", "01")

will be fetched to subline "11" only and a copy of the new data is stored according to Table 2.

In a similar way, a replacement of subline "10" does not change the allocation mode (step 7). It will update the history table to "10-11" and save a redundant copy of subline "10".

The above example demonstrates the main idea of the proposed soft redundancy allocation. By monitoring real-time access patterns recorded in the history table, we can keep track of the locations (sublines) in a cache line that are accessed more frequently by a program. Since these sublines are accessed very often, we assign redundant sublines to them for the purpose of error tolerance. On the other hands, the sublines that are accessed less frequently are most likely the ones storing irrelevant data due to the non-idealities of spatial locality. We can then free these sublines and use them as soft redundancy for the frequently accessed sublines. Hence, the proposed technique is self-adaptive to runtime varying spatial locality and is able to exploit transient (soft) redundancy for error protection.

Table 1 shows the general procedure for updating the history table and directing the mode transition. The allocation mode decides whether to allocate redundant space for error protection or to fetch the entire cache line for performance. If the cache line is in the error-checking mode, each subline listed in the history table will have a redundant copy in the same cache line. On the other hand, if the cache line is in the no-checking mode, the entire cache line will be fetched without making any redundant copy. The allocation mode switches when enough confidence has been established.

2.2 Error Protection

When the program reads data in the cache lines currently in the error-checking mode, errors could be detected by comparing the data with the redundant copy.

In traditional cache microarchitecture, memory hit/miss is generated by comparing the desired tag address with the tag address of the cache line. A hit is generated if the two tag addresses are matched. What is different in the proposed technique is an additional comparison between the two data copies in the same cache line. Mismatches between the two copies indicate errors hence result in cancelation of memory access. A memory miss is generated as a result. There is a possible situation that when redundant copy is corrupted while the original data is correct, a misjudgement on the correctness of the data would occur, thereby introducing an additional miss with performance penalty. However, since the probability is relatively low, the performance penalty is negligible.

The proposed technique could detect multiple errors that occur in any bits. This is a significant improvement over the

existing error-control techniques that are only effective for single-bit or double-bit errors. Statistical analysis in section 4 demonstrates about $10X$ improvement in error detection capability over the existing techniques.

3. DESIGN OPTIMIZATION

In this section, we present an algorithm to determine the optimal configuration of the proposed soft-redundancy allocated memory microarchitecture.

Consider a general case where each cache line contains m words divided into n sublines. Here we assume the minimum size of a subline is a single word. Each cache line also integrates one flag bit for the allocation mode and a history table with $\frac{n}{2}\log_2 n$ bits to record the most recently replaced sublines. The capacity of the history table should be large enough to store the IDs of at most half of the total sublines in each cache line.

If the subline listed in the history table is hit or missed consecutively for certain times, the allocation mode will switch between the no-checking mode and error-checking mode. Assume that the mode will switch after c times. Thus, the parameter c defines the confidence level in mode switch. In the no-checking mode, the entire cache line will be fetched and no redundant copy will be made. In the error-checking mode, each subline listed in the history table has a redundant copy in the same cache line. When the program accesses a word in a subline listed in the history table, the second copy is compared with in order to detect possible errors in the original data.

The subline number n is a key parameters in the proposed technique. The value of n needs to satisfy the following condition

$$n = 2^t, \tag{1}$$

where $t \in [1, 2, ..., \log_2 m]$, and m is the number of words in each cache line. Thus, $n_{min} = 2$ and $n_{max} = m$.

The confidence level c also affects the performance of the proposed technique. It can be chosen from a range given by

$$c \in [1, 2, \cdots, \frac{n_{max}}{2}]. \tag{2}$$

It is necessary to determine the optimal values of these two parameters by considering the tradeoffs between error protection and design overheads. The timing overheads of the proposed technique can be minimized by executing the required operations (e.g., data comparison and redundant data copying) in parallel with non-critical operations, thereby keeping the operations off the critical timing path. In the following, we will focus on the hardware overheads in this optimization analysis.

A large subline number n provides more flexibility in exploiting the soft redundancy and hence leads to better error protection. But on the other hand, it increases the hardware complexity, especially the size of history table.

The confidence level c affects the mode switch frequency and thus the coverage of error protection. In the proposed technique, the sublines are not under the error protection all the time. When a cache line is in the no-checking mode, the data in that cache line is not protected. The error protection coverage ratio, denoted as R_p, can be calculated by

$$R_p = \frac{MA_{error-checking}}{MA_{error-checking} + MA_{no-checking}}, \tag{3}$$

Table 3: Algorithm for determining the optimal parameters - (n_{opt}, c_{opt}).

parameter_selection(α, β, *line_size*, *cache_size*, *assoc*, n_{opt}, c_{opt})

input:
- α (weight of error protection coverage ratio consideration)
- β (weight of hardware cost consideration)
- *line_size* (cache line size in words)
- *cache_size* (cache size in words)
- *assoc* (associativity)

output:
- n_{opt} (optimized subline number)
- c_{opt} (optimized switch confidence)

Do, using *line_size*, *cache_size*, *assoc*
- calculate possible values of (n,c)
- **FOR** each possible (n,c)
  - $R_p = \frac{MA_{error-checking}}{MA_{error-checking} + MA_{no-checking}}$
  - $C_h(n,c) = k\left(\frac{n\log_2 n}{2} + \frac{8m}{n} + \log_2 c + 2\right)$
  - find R_{max} of $R_p(n,c)$
  - find C_{max} of $C_h(n,c)$
  - $\widehat{R}_p(n,c) = \frac{R_p(n,c)}{R_{max}}$
  - $\widehat{C}_h(n,c) = \frac{C_h(n,c)}{C_{max}}$
- **ENDFOR**
- find (n_{opt}, c_{opt}) to maximize $\left[\frac{(\widehat{R}_p(n,c))^\alpha}{(\widehat{C}_h(n,c)^\beta}\right]$

End

where $MA_{no-checking}$ and $MA_{error-checking}$ are the numbers of memory accesses when the cache line is in the no-checking mode and error-checking mode, respectively. Obviously, the switch frequency of allocation mode determines the error protection coverage ratio. As R_p is determined by both parameters n and c, we can express it as an explicit function of n and c as $R_p(n,c)$. It is very difficult to represent the $R_p(n,c)$ in a closed-form expression. However, numerical values can be obtained by averaging over different programs subject to various memory access patterns.

The hardware overheads of the proposed technique include history tables, additional comparators and control bits. Each cache line employs $(\frac{n}{2}\log_2 n)$ bits to maintain the access history. At the switch confidence level c, each cache line needs $(\log_2 c + 1)$ bits for mode transition control and one bit for allocation mode indication. Each comparator for error detection is a $\frac{16m}{n}$ bit comparator, where m is the number of words in each cache line. Since each comparator is shared by two arbitrarily assigned sublines and each cache line has n sublines, the number of comparator bits is $\frac{16m}{n} \times \frac{n}{2} = 8m$. For a cache with k cache lines where each line contains m words and is divided into n sublines with switch confidence level of c, the hardware overhead $C_h(n,c)$ is the total size of the history tables, the comparators, and the control bits, expressed as

$$C_h(n,c) = k\left(\frac{n\log_2 n}{2} + 8m + \log_2 c + 2\right). \tag{4}$$

To derive a general solution applicable to different programs with different memory access patterns, we normalize the hardware overhead by its maximum value C_{max}, and the

error protection coverage ratio by R_{max}, i.e.,

$$\widehat{C}_h(n,c) = \frac{C_h(n,c)}{C_{max}}, \tag{5}$$

$$\widehat{R}_p(n,c) = \frac{R_p(n,c)}{R_{max}}. \tag{6}$$

The optimal soft redundancy allocation is the one that achieves the best tradeoff between error protection coverage ratio and hardware overheads. This can be expressed as the following optimization problem

$$\exists(n_{opt}, c_{opt}) \in (n,c), s.t., \tag{7}$$

$$\frac{(\widehat{R}_p(n_{opt}, c_{opt}))^\alpha}{(\widehat{C}_h(n_{opt}, c_{opt}))^\beta} = \max\left[\frac{(\widehat{R}_p(n,c))^\alpha}{(\widehat{C}_h(n,c)^\beta}\right], \tag{8}$$

where α and β are the weight factors for error protection coverage ratio and hardware overheads, respectively. By changing the values of α and β, we can adjust the design priority between error protection and hardware overheads.

Table 3 summarizes the proposed algorithm in determining the optimal configuration of soft redundancy allocation.

4. STATISTICAL ANALYSIS OF ERROR TOLERANCE

In this section, we perform a statistical analysis to quantify the error tolerance of the proposed soft-redundancy allocated memory.

Traditionally, soft errors in memory systems are modeled as single-bit upsets (SBU). As the feature size of semiconductor process being scaled into the nanometer domain, a single partial strike may potentially destroy the states of multiple memory bits, resulting in multiple-bit upsets (MBU). Thus, we need to evaluate the error protection for both SBU and MBU.

We compare our technique with the parity checking code and single-error-correction double-error-detection Hamming code. Parity checking code is considered as the most effective technique for detecting SBU, whereas Hamming code provides error detection for up to two bits of errors. In our estimation model, each soft error can cause a single-bit error or a double-bit error with the probabilities of P_s and P_m, respectively. The corresponding values are estimated as

$$P_s = \beta, \tag{9}$$

$$P_m = \beta \cdot 10^{-2}, \tag{10}$$

where β is the soft error rate (SER). Based on the observation in [10], the probability of SBU is selected as two orders of magnitude larger than that of MBU. In addition, we model the soft errors as independently and identically distributed events, i.e., a memory entry may have multiple errors caused by multiple soft error events.

For SBU dominant scenarios, the probabilities of undetected errors (PUE) are denoted by $P_{ue\_s\_sr}$ and $P_{ue\_s\_par}$ for the proposed technique and the parity checking code, respectively. The PUE can be derived as

$$P_{ue\_s\_sr} = \sum_{i=1}^{n} C_n^i P_{s\_r}^i (1 - P_{s\_r})^{n-i}, \tag{11}$$

$$P_{ue\_s\_par} = \sum_{i=1}^{n/2} C_n^{2i} P_s^{2i} (1 - P_s)^{n-2i}, \tag{12}$$

where $C_n^i = \frac{n!}{(n-i)!i!}$, and $P_{s\_r} = P_s^2$ denotes the probability of a single-bit error that the proposed technique cannot detect. This occurs rarely only when the same bits of the original data and the redundant copy are both corrupted. On the other hand, the undetectable errors in parity checking schemes occur when the number of corrupted bits are even. Numerical results from (11) and (12) demonstrate $10X$ improvement in detection of SBU errors over the parity checking.

For errors due to MBU, the probabilities of undetected errors (PUE) are denoted as $P_{ue\_m\_sr}$ and $P_{ue\_m\_ham}$ for the proposed technique and the Hamming code, respectively. The PUE can be derived as

$$P_{ue\_m\_sr} = \sum_{i-1}^{n} C_n^i P_{m\_r}^i (1 - P_{m\_r})^{n-i}, \tag{13}$$

$$P_{ue\_m\_ham} = \sum_{i=2}^{n} C_n^i P_m^i (1 - P_m)^{n-i}, \tag{14}$$

where $P_{m\_r} = P_m^2$ denotes the probability of a double-bit error (the dominant MBU) that the proposed technique cannot detect. Similar to the SBU cases, this happens rarely only when the same two bits are corrupted in both the original data and the redundant copy. On the other hand, the undetectable errors in Hamming code checking schemes occur when more than one double-bits are corrupted. Numerical results from (13) and (14) demonstrate $10X$ improvement in detection of MBU errors over the Hamming code.

In addition to the improvement in error detection, a high error protection coverage ratio R_p (defined in (3)) as shown in the next section makes the proposed technique a superior solution for error tolerance.

5. EVALUATION AND DISCUSSION

In this section, we evaluate the error tolerance of the proposed soft-redundancy allocated memory microarchitecture. We also compare the memory access performance and bandwidth requirement with the traditional cache systems.

Our simulation results were obtained from a simulator based on the trace-driven simulator Dinero IV [11]. The cache model in this simulator is modified to support the proposed soft redundancy microarchitecture. The proposed technique is evaluated in a direct mapped cache whose total size is $32KB$ and each line is $32B$. The number of sublines is selected to be 8, and switch confidence value is set to 2. All the simulations were running on the SPEC CPU2000 [12] trace files collected from the Stream-Based Trace Compression (SBC) [13], where trace files of 23 benchmarks are available.

In section 4, we have shown that the proposed technique achieves an improved error detection capability. Many existing works [14]–[16] on soft errors usually assume certain conditions or target specific architectures. Instead of simulating soft errors directly, we evaluate the error protection ratio R_p as defined in (3). Table 4 shows the results of error protection ratio of the 23 workloads selected for simulation. These results are obtained from (3) using statistical results reported by the simulator. The average error protection ratio of all the 23 benchmarks is 73.7%. In some benchmarks, e.g., art, mcf, and applu, nearly all the memory accesses are protected by our mechanism. In addition, the proposed technique induces very small hardware overheads. The ex-

Table 4: Error protection coverage ratio.

Workloads	Coverage	Workloads	Coverage
1.ammp	84.8%	13.lucas	99.8%
2.applu	99.9%	14.mcf	100.0%
3.apsi	97.4%	15.mesa	21.4%
4.art	100.0%	16.mgrid	99.7%
5.crafty	54.1%	17.parser	58.1%
6.eon	33.5%	18.perlbmk	31.4%
7.equake	78.3%	19.sixtrack	75.8%
8.fma3d	84.9%	20.swim	99.7%
9.galgel	99.8%	21.twolf	56.0%
10.gap	45.2%	22.vortex	31.3%
11.gcc	91.2%	23.wupwise	94.7%
12.gzip	58.3%	**Average**	73.7%

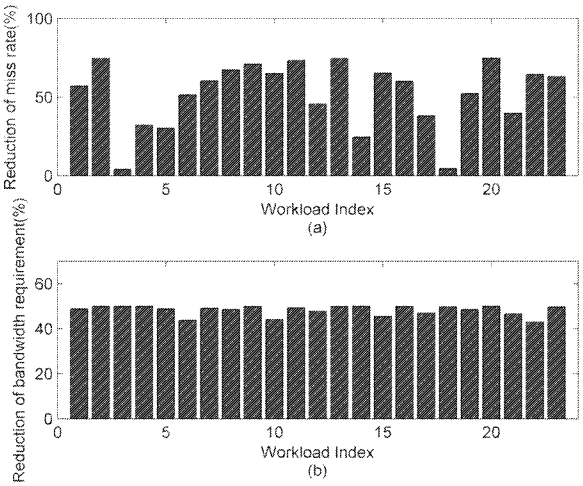

Figure 3: Simulation results. (a) Reduction of miss rate as compared to the sub-blocked cache. (b) Reduction of bandwidth requirement as compared to the traditional cache.

tra hardware includes just 15 bits per cache line and a 32-bit comparator for each redundancy subline pair.

It is expected that the proposed technique will possibly introduce higher miss rates caused by the wrong prediction of soft redundancy. But on the other hand, the proposed technique only fetches useful data when possible, thereby reducing the memory bandwidth requirement. As proved in many sub-blocked cache designs [5] which feature a similar cache structure, the reduction in bandwidth requirement can offset the increase in miss rate, leading to an overall performance improvement. In comparison with the sub-blocked cache design with the same sub-block number, our technique achieves an average of 52% reduction in miss rate (see Fig. 3(a)). The bandwidth requirement in term of total fetched words is also significantly reduced, averaging at 48.3% as compared to the tradition cache (see Fig. 3(b)).

Additional power consumption results from the error control operations such as duplicating data and error checking. Our future work will be directed to the tradeoffs between error tolerance and power consumption. Also note that the efficiency of the history-based redundancy allocation determines the error tolerance, performance, and bandwidth requirement. Future work will study on how to efficiently exploit soft redundancy in applications that have different memory access spatial locality along with caches that have different cache line sizes. Further improvement of the proposed error-control technique could be a combination of the soft redundancy and error checking codes, thereby providing error checking to cover all the data and meanwhile improving error detection.

6. CONCLUSIONS

This paper presents a new error-control technique referred to as the *soft redundancy* for efficient memory access and effective error protection. Statistical analysis reveals about $10X$ improvement in error detection over the existing error-control techniques. Simulation results demonstrate 73.7% average error protection ratio on the 23 benchmarks, with average of 52% and 48.3% reduction in memory miss rate and bandwidth requirement, respectively, as compared to the existing techniques. Future work is being directed towards exploiting soft redundancy with thread information for multithreaded computing.

Acknowledgment

This research was supported in part by the University of Connecticut Faculty Research Grant 446751.

7. REFERENCES

[1] The International Technology Roadmap for Semiconductors, 2003 at *http://public.itrs.net/Files/2003ITRS/Home2003.htm.*

[2] H. L. Hughes and J. M. Benedetto, "Radiation effects and hardening of MOS technology: devices and circuits," *IEEE Trans. on Nuclear Science*, vol. 50, pp. 500-521, 2003.

[3] K. Chakraborty, et. al., "A physical design tool for built-in self-repairable RAMs," *IEEE Trans. on VLSI*, vol. 9, pp. 352-364, 2001.

[4] M. Biberstein and T. Etzion "Optimal codes for single-error correction, double-adjacent-error detection," *IEEE Trans. on Information Theory*, vol. 46 , pp. 2188-2193, Sept. 2000.

[5] R. Fellman, et. al., "Design and evaluation of an architecture for a digital signalprocessor for instrumentation applications," *IEEE Transaction on Acoustics, Speech, and Signal Processing*, vol. 38, pp. 537-546, 1990.

[6] A. Veidenbaum, et. al., "Adapting Cache Line Size to Application Behavior," *Proc. ICS*, 1999.

[7] C. Zhang, et. al., "Energy Benefits of a Configurable Line Size Cache for Embedded Systems," *Proc. International Symp. on VLSI*, pp. 87-91, 2003.

[8] K. Inoue, et. al., "Dynamically Variable Line-Size Cache Exploiting High On-Chip Memory Bandwidth of Merged DRAM/Logic LSIs," Proc. HPCA, pp. 218-222, 1999.

[9] P. Pujara, A. Aggarwal, "Increasing the cache efficiency by eliminating noise," IEEE High-Performance Computer Architecture, pp. 145-154, 2006.

[10] F. Wrobel, et. al., "Simulation of nucleon-induced nuclear reactions in a simplified SRAM structure: scaling effects on SEU and MBU cross sections," *IEEE Transactions on Nuclear Science*, pp. 48(6):1946-1952, 2001.

[11] J. Edler and M. D. Hill, "Dinero IV Trace-Driven Uniprocessor Cache Simulator," at *http://www.cs.wisc.edu/~markhill/DineroIV/.*

[12] SPEC CPU2000 at *http://www.spec.org/cpu/.*

[13] A. Milenkovic and M. Milenkovic, "Exploiting Streams in Instruction and Data Address Trace Compression," *Proc. IEEE Annual Workshop on Workload Characterization*, pp. 99-107, 2003.

[14] S. S. Mukherjee, J. Emer, S. K. Reinhardt, "The soft error problem: an architectural perspective," *Proc. Intl. Symp. High-Performance Computer Architecture*, pp. 243-247, 2005.

[15] N. Seifert, X. Zhu, and L. W. Massengill, "Impact of scaling on soft-error rates in commercial microprocessors," *IEEE Trans. Nuclear Science*, vol 49, pp. 3100-3106, 2002.

[16] P. Hazucha, T. Karnik, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Trans. on Dependable and Secure Computing*, vol 1, pp. 128-143, 2004.