Joint Performance Improvement and Error Tolerance for Memory Design Based on Soft Indexing

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

***Abstract*— Memory design is facing the dual challenges of performance improvement and error tolerance due to a combination of technology scaling and higher levels of integration. To address these challenges, we propose a new memory microarchitecture referred to as the *soft indexing*. The proposed technique allocates memory resources in a self-adaptive manner in accordance with runtime program variations, thereby achieving efficient memory access and effective error protection in a coherent manner. Statistical analysis shows $10X$ improvement in error detection capability over the existing error-control techniques. The benefits of the proposed technique are also experimentally demonstrated using the SPEC CPU2000 benchmarks. Simulation results show 94.9% average error-control coverage on the 23 benchmarks, with average of 23.2% reduction in memory miss rates as compared to the conventional techniques.**

I. Introduction

The advances in semiconductor technology allow integrated circuits to maintain an incredible pace of performance improvement. The state-of-the-art processor design benefits greatly from higher levels of integration enabled by the relentless scaling of semiconductor process. With semiconductor process being scaled into sub-65nm nodes, processor design has entered the multibillion-transistor architecture era. At the same time, nanometer devices are approaching their physical limits. Reliable implementation of integrated systems is becoming increasingly difficult. Consequently, processor design is experiencing a wide range of unmanageable performance spread caused by the non-idealities in design, fabrication and operation environment [1]– [5]. These problems not only make it difficult to achieve affordable scaling but also induce severe reliability issues.

These emerging issues are most pronounced in on-chip memory systems, where minimum-geometry devices are utilized to build bulk of memory for high-density integration. Memory circuits are subject to a larger percentage of process variations and their performance is very sensitive to variations in process, supply voltage and temperature. Furthermore, memory circuits are vulnerable to soft errors caused by particle strikes and timing-related transient errors introduced by clock skew coupled with signal delay variations. As memory performance has become a limiting factor in high-performance processors, the composite effect makes memory design exposed to the dual challenges of performance improvement and error tolerance.

Existing reliability enhancing techniques include radiation/upset-hardened memory structures [6], [7], double or triple memory redundancy [8], [9], and code checking algorithms [10], [11]. Integrating these techniques into high-performance on-chip memory presents a significant challenge due to the severe constraints on area and timing margins. On the other hand, research in memory microarchitecture focuses primarily on reducing misses and memory traffic. Sub-blocked caches [12], [13] reduce memory traffic by transferring only a single sub-block on a cache miss instead of fetching the whole cache line. Alternatives to sub-blocked caches use adaptive techniques [14]– [16] to further improve cache utilization. Some works also focus on statically distributing cache accesses on the granularity of cache block. Skewed associative caches [18]– [20] use skew functions for mapping the desired address to the cache line address in order to achieve balanced cache access distribution. Further improvement is achieved in [21]– [23] by employing new hash functions to obtain uniform cache accesses without performance slowdown.

However, none of the above techniques target both memory robustness and access performance, which are now closely connected due to a combination of technology scaling and higher levels of integration. In this paper, we propose a new approach referred to as the *soft indexing* that achieves high-performance memory utilization and effective error control in a coherent manner. Our idea is based on the observation that memory accesses are non-uniform, which results in many idle memory spaces and high conflict misses. The idle memory spaces create transient memory redundancy that can be exploited for both performance improvement and error control. The proposed soft indexing technique allocates memory resources in a self-adaptive manner in accordance with runtime memory behaviors, thereby achieving efficient memory access and effective error protection jointly. Statistical analysis shows 10X improvement in error detection over the existing error-control techniques. The benefits of the proposed technique are also demonstrated by the SPEC CPU2000 benchmarks [26]. Simulation results show 94.9% average error-control coverage on the 23 benchmarks, with average of 23.2% reduction in memory miss rate as compared to the existing techniques.

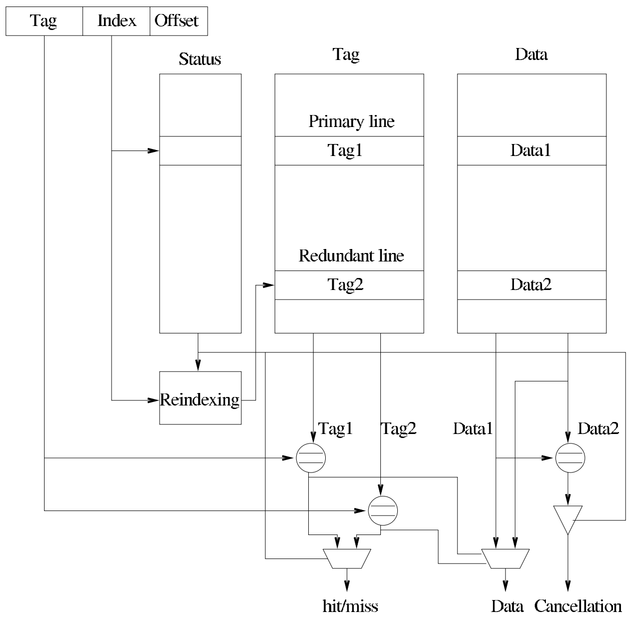

Fig. 1. Microarchitecture of soft indexing memory.

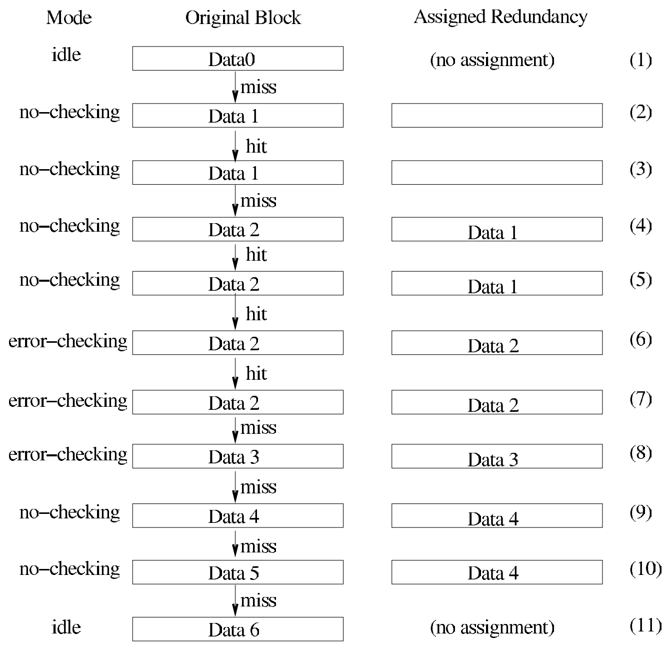

Fig. 2. An example of soft indexing.

In section II, we develop the soft indexing memory microarchitecture for joint error protection and performance improvement. In section III, we present a statistical analysis on error tolerance and provide the comparison to the existing techniques. In section IV, we evaluate the performance of the proposed technique. Section V concludes the paper.

II. Soft Indexing Memory Microarchitecture

The size of cache memory has a direct impact on the overall processor performance. In general, increasing cache size can reduce the miss rate by providing more memory resources for the workloads. However, runtime program statistics reveal some non-uniform memory access patterns, where many cache lines are accessed less frequently and could even remain idle or unused over time. This creates transient redundancy that can be exploited to jointly improve memory access performance and error tolerance. Since the distribution of idle cache lines varies during runtime, we need an adaptive (soft) indexing mechanism for dynamic allocation of memory resources.

In this section, we present the soft indexing memory microarchitecture that exploits the transient redundancy generated by non-uniform memory accesses for joint performance improvement and error tolerance.

A. Soft Indexing

Fig. 1 shows the proposed memory microarchitecture. A re-indexing function is introduced to assign an idle cache line to the currently accessed cache line for different purposes that will be explained later. Each cache line is extended with some status bits that keep track of the history of access patterns. These status bits are stored in a status table for the control of memory allocation. As shown in the following discussion, memory resources are dynamically allocated according to the availability of transient redundancy and the statistics of memory access patterns.

An example of memory access sequence using soft indexing is shown in Fig. 2. Each cache line can be operated in one of the three allocation modes: *idle*, *no-checking*, and *error-checking*, which represent different memory access patterns. The *idle* mode indicates that a cache line is currently idle or unused. Thus, this cache line can be used for transient redundancy. The *no-checking* mode represents the situation where a cache line is accessed frequently but the access pattern is less predictable. The program lacks the confidence in the outcome (hit or miss) of subsequent accesses to this cache line. Since the performance of this cache line is unstable, we can assign a redundant cache line (i.e., one of those in the *idle* mode) to this cache line to improve the access performance. On the other hand, the *error-checking* mode indicates the confidence in hit occurrence for a frequently accessed cache line. Since this cache line is experiencing access hit in a stable pattern, we can assign a redundant cache line to this cache line to improve the error tolerance.

The mode switching is based on the access history stored in the status table. The detailed operation of memory access is explained below in reference to the steps in Fig. 2.

Initially, all cache lines are set to the *idle* mode by default, and the re-indexing function is disabled. In the subsequent operations, assume that a miss occurs in a cache line. The status of this cache line is thus changed from the *idle* mode to the *no-checking* mode (step 2 in Fig. 2). Meanwhile, a redundant cache line will be assigned to this cache line by re-indexing the requested address. Ideally, the redundant cache line should be the one currently in the idle or unused status. In the proposed technique, we locate the redundant cache line using a re-indexing function as discussed later.

A single, non-consecutive hit in the cache line that is in the *no-checking* mode will not cause any data replacement (steps 3 and 5), whereas a single, non-consecutive miss will result in a replacement of both the primary and the redundant cache lines (step 4). The current data in the primary cache line will be transferred to the redundant cache line, and the new data will be filled into the primary cache line. Note that in the *no-checking* mode, the redundant cache line keeps the previously accessed data (data 1), which in general is different from the new data (data 2) in the primary cache line. This improves memory access performance by saving the previously accessed data nearby for possible future use.

Two consecutive hits in the *no-checking* mode establish the confidence in memory hit occurrence in this cache line. Thus, this cache line is switched to the *error-checking* mode (step 6). The redundant cache line is thereafter updated as a redundant copy of the primary cache line. Note that the switch condition (e.g., two consecutive hits or misses in this example) can be configured for different requirements on memory performance and error control. In the *error-checking* mode, a hit will get both copies in the primary cache line and the redundant cache line. The two copies are then compared to detect any possible data errors. If an error is detected, the hit is canceled and a miss is generated instead (step 7, also see Fig. 1). On the other hand, a single miss will replace both cache lines with the new data (step 8).

Two consecutive misses in the *error-checking* mode will change the status of the primary cache line to the *no-checking* mode (step 9) due to the loss of confidence in hit occurrence. Furthermore, two consecutive misses in the *no-checking* mode will send the cache line back to the *idle* mode (step 11). When the status returns to *idle* mode, this cache line is no longer associated with any redundant cache line. Subsequently, this cache line can either return to the *no-checking* mode if an access occurs, or remain in the *idle* mode if no access occurs. For the latter case, this cache line can be used as a transient redundancy for other cache lines.

In the proposed soft indexing microarchitecture, the allocation mode switches when enough confidence in hit or miss occurrence has been established. In the above example, this confidence is measured by two consecutive hits or misses. The switch control diagram is shown in Fig. 3. In total, five states are needed to control the mode switch, which requires only three status bits each cache line. This incurs very small hardware overheads.

Ideally, we would like to find the idle or unused cache lines for redundancy. However, this is a really difficult task due to the complexity of memory runtime behaviors. Here, we exploit memory spatial locality to locate redundant cache lines. Specifically, we utilize a XOR-based re-indexing function, where the primary cache index are masked by an XOR code to generate the address of the redundant cache line that is guaranteed to be far away from the primary cache line. Due to memory spatial locality, it is unlikely that the program will access these two memory locations simultaneously. Simulation results in section IV demonstrate this scheme is quite effective.

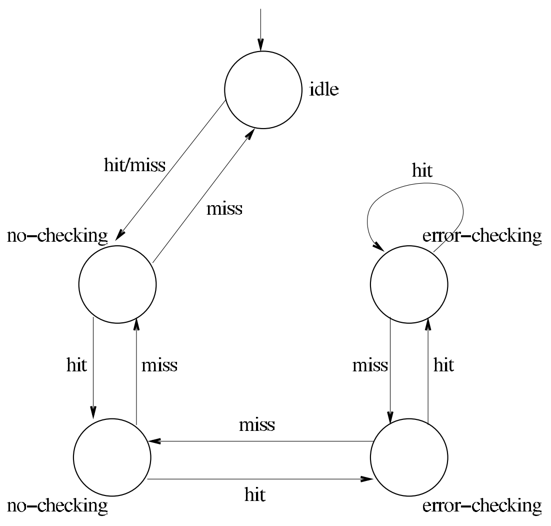

Fig. 3. Allocation mode switch control.

B. Joint Performance Improvement and Error Tolerance

The proposed soft indexing technique exploits runtime memory behaviors to improve access performance and error tolerance. For cache lines in the *no-checking* mode, the program does not have enough confidence in hit or miss occurrence. This implies that the access performance is unstable for these cache lines. Instead of trashing the previously accessed data during a miss, we save these data in the redundant cache lines for possible future use (see steps 4, 5 and 10 in Fig. 2). In response to the subsequent memory accesses, if the redundant cache lines have the requested data, the data can be retrieved instead of fetching from the lower level memory. Indeed, by holding the previously accessed data in the redundant cache lines, conflict misses are reduced due to an equivalent increase in set associativity.

For cache lines in the *error-checking* mode, the program provides enough confidence in hit occurrence. These cache lines are hit frequently and presumably contain important data. Therefore, the redundant cache lines are assigned to these cache lines for error control (see steps 6−8 in Fig. 2). During data read-out, there is an additional comparison between the two data copies in the cache line pair. This comparison can be performed in parallel with the tag address comparison. Mismatches between the two copies indicate errors and hence call for cancelation of the data read-out. A memory miss is generated as a result and the new data will be fetched from the lower level memory. The redundant cache lines are generated dynamically and provide an effective means for error tolerance.

C. Design Considerations

The overheads of the proposed microarchitecture are manageable. Our technique does not require extra ports for the cache. Actually, only a small write buffer is needed. Other hardware overheads include a few bits per cache line for the status table and the XOR re-indexing function. Also, the latency of the additional data comparison is masked by the tag address comparison, thereby not involving any timing penalty on the critical paths.

The operations in the *idle* mode remain the same as those in traditional cache. On the other hand, if the accessed cache line is in the *no-checking* mode, the original tag and data in the primary cache line are forwarded to the write buffer, while the tags from both the primary cache line and the redundant cache line are compared at the same time. If either cache line contains the requested address, a hit is generated and the requested data is delivered to the execution unit (for a read) or written to the primary cache line (for a write). Otherwise, a miss is generated and the requested data are fetched from the lower level memory (for a read) or written to the primary cache line (for a write). Thereafter, the previous data in the primary cache line stored in the write buffer will be written to the redundant cache line. This write operation is performed off-line and thus does not affect other memory operations that might be timing-critical.

Similarly, if the accessed cache line is in the *error-checking* mode, an additional data comparison between the cache line pair is needed during a read access for error detection. As mentioned before, this comparison can be conducted in parallel with tag comparison, thereby not affecting memory timing. For a write access, the write buffer is able to hide the write latency by scheduling the access to the redundant cache line at a later time.

III. Statistical Analysis of Error Tolerance

In this section, we perform a statistical analysis to quantify the error tolerance achieved by the proposed soft indexing technique.

In traditional memory systems, soft errors are typically 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 corrupt multiple memory bits, resulting in multiple-bit upsets (MBU). In addition, timing noise tends to cause MBU as well. Among the existing solutions, parity checking code is considered as the most effective for detecting SBU, whereas Hamming code provides error detection for up to two bits of errors. Error-control codes for more than two bits of errors are quite complicated and thus are seldom used in memory systems.

Consider a cache line with n bits each entry, i.e., each access can obtain n bits in total. A single soft error corruption may lead to either a SBU or a MBU. In [24], the rates of SBU and MBU are different from one order to three orders of magnitude based on the operating conditions such as supply voltages. According to this observation, we use two orders of magnitude of difference between the SBU and MBU rates, i.e.,

$$P_s = \beta, \tag{1}$$

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

where β is the soft error rate (SER). The soft errors are assumed to be independently and identically distributed (i.i.d.) events.

TABLE I
Probability of Undetected SBU (PUS).

SER	PUS parity checking	PUS proposed technique
10^{-4}	1.20×10^{-6}	1.60×10^{-7}
10^{-5}	1.20×10^{-8}	1.60×10^{-9}
10^{-6}	1.20×10^{-10}	1.60×10^{-11}
10^{-7}	1.20×10^{-12}	1.60×10^{-13}
10^{-8}	1.20×10^{-14}	1.60×10^{-15}

For SBU dominant cases, we denote $P_{ue\_s\_sr}$ and $P_{ue\_s\_par}$ as the probabilities of undetected errors (PUE) in the proposed technique and that in the parity checking code, respectively. We can derive

$$P_{ue\_s\_sr} = \sum_{i=1}^{n} C_n^i P_{s\_r}^i (1 - P_{s\_r})^{n-i}, \tag{3}$$

$$P_{ue\_s\_par} = \sum_{i=1}^{n/2} C_n^{2i} P_s^{2i} (1 - P_s)^{n-2i}, \tag{4}$$

where $C_n^i = \frac{n!}{(n-i)!i!}$, and $P_{s\_r} = P_s^2$ is the probability of a single-bit error that cannot be detected by the proposed technique. 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 (3) and (4) demonstrate $10X$ improvement in error detection capability over the parity checking code, as shown in Table I where the number of bits is $n = 16$, i.e., each memory access fetches a 16-bit word.

The proposed technique is also able to detect multiple errors occurred in any bits. This is a significant improvement over the existing error-control techniques such as parity checking code and single-error-correction double-error-detection Hamming code. The probability of undetected errors (PUE) in the MBU cases can be derived as

$$P_{ue\_m\_sr} = \sum_{i=1}^{n} C_n^i P_{m\_r}^i (1 - P_{m\_r})^{n-i}, \tag{5}$$

$$P_{ue\_m\_ham} = \sum_{i=2}^{n} C_n^i P_m^i (1 - P_m)^{n-i}, \tag{6}$$

where $P_{m\_r} = P_m^2$ denotes the probability of a double-bit error (the dominant MBU) that cannot be detected by the proposed technique. 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 occur when more than two memory bits are corrupted. Again, numerical results from (5) and (6) demonstrate $10X$ improvement in error detection capability over the Hamming code, as shown in Table II where the number of bits is $n = 16$.

Unlike parity checking and Hamming code that provide static error-control coverage to all the cache lines, the proposed

TABLE II
PROBABILITY OF UNDETECTED MBU (PUM).

SER	**PUM Hamming code**	**PUM proposed technique**
10^{-4}	1.20×10^{-10}	1.60×10^{-11}
10^{-5}	1.20×10^{-12}	1.60×10^{-13}
10^{-6}	1.20×10^{-14}	1.60×10^{-15}
10^{-7}	1.20×10^{-16}	1.60×10^{-17}
10^{-8}	1.20×10^{-18}	1.60×10^{-19}

TABLE III
CONFIGURATION OF SIMULATION ENVIRONMENT.

Parameter	**Value**
Cache Size	32KB
Line Size	32B
Associativity	Direct Mapped
States number of initial mode	1
States number of no-checking mode	2
States number of error-checking mode	2
Reindexing XOR code	10'b1000010001

technique relies upon a dynamic mapping strategy that enables error protection only when necessary while releasing the unused memory resources for other critical tasks such as improving access performance. In fact, this approach leads to a joint optimization for efficient memory access and effective error tolerance. Since memory operation mode is dynamically switching in accordance with runtime memory requirements, the cache lines are not always under the error protection. Specifically, when a cache line is in the *no-checking* mode, the data is not protected as this cache line undergoes an unstable access pattern. The error-control coverage ratio, denoted as R_p, can be calculated by

$$R_p = \frac{MA_{error-checking}}{MA_{total}}, \tag{7}$$

where $MA_{error-checking}$ and MA_{total} are the number of memory accesses when the cache line is in the *error-checking* mode and the total number of memory accesses, respectively. The error-control coverage ratio R_p reflects the effectiveness of the proposed technique in dealing with soft errors. Obviously, the switching frequency of memory allocation modes affects the error-control coverage ratio.

IV. SIMULATION RESULTS

In this section, we study the performance of memory access and error tolerance achieved by the proposed technique.

Our simulation results were obtained from a trace-driven simulator based on Dinero IV [25], which is a uniprocessor cache simulator for memory reference. The cache model in this simulator is modified to support the proposed soft indexing microarchitecture. Table III shows the configuration of the simulation environment. All the simulations were running on the SPEC CPU2000 [26] trace files collected from the Stream-Based Trace Compression (SBC) [27], where traces of 23 benchmarks are available. In these simulations, we use direct mapped cache for the purpose of demonstration. We expect to extend the proposed memory microarchitecture to set-associative cache in our future work.

As shown in Fig. 4, our technique achieves an average of 23.2% reduction in miss rate for the 23 benchmarks as compared to the conventional cache design. These results demonstrate that our technique is very effective in improving memory access performance. Note that the XOR code for the re-indexing function is a pre-determined value for all the 23 benchmarks. This XOR code introduces three-bit inversion

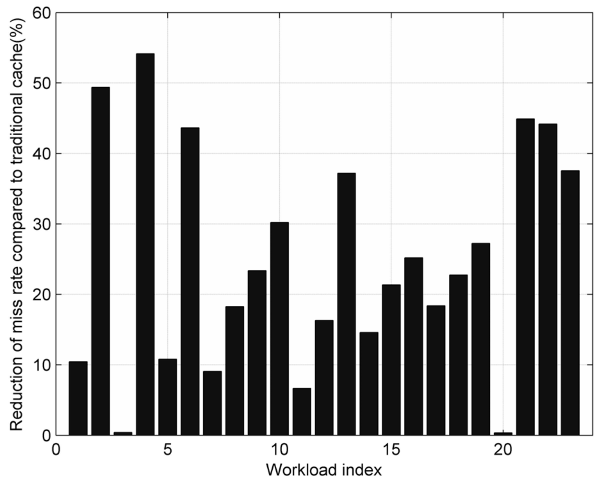

Fig. 4. Reduction of miss rate compared to conventional direct mapped cache.

TABLE IV
ERROR-CONTROL COVERAGE RATIO.

Index	**Workloads**	**Error-Control Coverage Ratio**
1	ammp	97.6%
2	applu	99.8%
3	apsi	91.9%
4	art	84.2%
5	crafty	97.2%
6	eon	99.9%
7	equake	99.9%
8	fma3d	99.8%
9	galgel	99.6%
10	gap	99.9%
11	gcc	99.9%
12	gzip	97.7%
13	lucas	99.8%
14	mcf	26.6%
15	mesa	99.9%
16	mgrid	93.4%
17	parser	98.7%
18	perlbmk	99.7%
19	sixtrack	99.7%
20	swim	99.8%
21	twolf	98.9%
22	vortex	99.8%
23	wupwise	99.7%
	Average	94.9%

from the index of the primary cache line to that of the redundant cache line. Future work needs to exploit dynamic code generation for re-indexing function to further improve the adaptability.

Evaluating error tolerance using architecture simulators requires realistic error models and error injection mechanisms. Many existing works [28]– [32] on soft errors usually assume certain conditions or target specific architectures. Instead of simulating errors directly, we evaluate the error-control coverage ratio as defined in (7). A higher error protection coverage ratio along with the improved error detection capability implies better tolerance to memory errors. Table IV shows the results of error-control coverage ratio of the 23 workloads. These results are obtained from (7) using statistical results reported by the simulator. The average error-control coverage ratio of all the 23 benchmarks is measured at 94.9%. These results along with the theoretical analysis on error detection capability in section III demonstrate the significant advantage of the proposed technique in error tolerance. Moreover, the proposed technique induces very small design overheads as described in section II-C. Future work to improve the proposed error-control technique could be a combination of the soft indexing and error checking codes, thereby providing error checking to cover all the data and meanwhile improve error detection.

V. Conclusions

This paper presents a new memory microarchitecture, referred to as the *soft indexing*, for joint performance improvement and error protection. Statistical analysis shows about $10X$ improvement in error detection capability over the existing error-control techniques for both single-bit upsets and multiple-bit upsets. Trace-based simulation results demonstrate 94.9% average error-control coverage on the 23 benchmarks, with average of 23.2% reduction in memory miss rates as compared to the conventional techniques. Future work is being directed towards exploiting transient redundancy with thread information for multithreaded computing.

Acknowledgment

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

References

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

[2] K. Bowman, S. Duvall, and J. Meindl, "Impact of die-to-die and withindie parameter fluctuations on the maximum clock frequency distribution for gigascale integration," *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 183C190, 2002.

[3] P. Zuchowski, P. Habitz, J. Hayes, and J. Oppold, "Process and environmental variation impacts on ASIC timing," *Proc. ICCAD Tech. Dig.*, pp. 336-342, 2004.

[4] S. Samaan, "The impact of device parameter variations on the frequency and performance of VLSI chips," *Proc. ICCAD Tech. Dig.*, pp. 343-346, 2004.

[5] X. Qi, S. C. Lo, Y. Luo, A. Gyure, M. Shahram, and K. Singhal, "Simulation and analysis of inductive impact on VLSI interconnects in the presence of process variations," *Proc. IEEE Custom Integr. Circuits Conf.*, pp. 309-312, 2005.

[6] 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.

[7] T. Calin, M. Nicolaidis, R. Velazco, "Upset hardened memory design for submicron CMOS technology," *IEEE Trans. on Nuclear Science*, vol. 43, pp. 2874-2878, 1996.

[8] K. Chakraborty, S. Kulkami, M. Bhattacharya, P. Mazumder, and A. Gupta, "A physical design tool for built-in self-repairable RAMs," *IEEE Trans. on VLSI*, vol. 9, pp. 352-364, 2001.

[9] F. L. Kastensmidt, L. Sterpone, L. Carro, M. S. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," *Design, Automation and Test in Europe*, pp. 1290-1295, 2005.

[10] W. Peterson, "Error-correcting codes," 2nd ed., *Cambridge : The MIT Press*, 1980.

[11] 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, 2000.

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

[13] M. Kadiyala, L. N. Bhuyan, "A dynamic cache sub-block design to reduce false sharing," *IEEE Intl. Conf. on Computer Design*, pp. 313-318, 1995.

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

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

[16] 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.

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

[18] A. Seznec, F. Bodin, "Skewed-associative caches," *Proceedings of PARLE*, pp 305-316, 1993.

[19] F. Bodin, A. Seznec, "Skewed associativity improves program performance and enhances predictability," *IEEE Trans. on computers*, vol. 46, pp. 530-544, 1997.

[20] A. Djordjalian, "Minimally-skewed-associative caches," *Symp. on Computer Architecture and High Performance Computing*, pp. 100-107, 2002.

[21] M. Kharbutli, K. Irwin, Y. Solihin, J. Lee, "Using prime numbers for cache indexing to eliminate conflict misses," *Proc. HPCA*, pp. 573-586, 2005.

[22] A. Gonzalez, M. Valero, N. Topham, J. M. Parcerisa, "Eliminating cache conflict misses through XOR-based placement functions," *Intl. Conf. on Supercomputing*, pp. 76-83, 1997.

[23] N. Topham, A. Gonzalez, "Randomized cache placement for eliminating conflicts," *IEEE Trans. on Computers*, vol. 48, pp. 185-192, 1999.

[24] 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*, pp. 48(6):1946-1952, 2001.

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

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

[27] A. Milenkovic and M. Milenkovic, "Exploiting Streams in Instruction and Data Address Trace Compression," *Proceedings of the IEEE 6th Annual Workshop on Workload Characterization*, pp. 99-107, 2003.

[28] 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.

[29] 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.

[30] 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.

[31] M. Sugihara, T. Ishihara, M. Muroyama, K. Hashimoto, "A Simulation-Based Soft Error Estimation Methodology for Computer Systems," *Intl. Symp. On Quality Electronic Design*, pp. 196-203,2006.

[32] S. Dolev, Y. A. Haviv, "Self-stabilizing microprocessor: analyzing and overcoming soft errors," *IEEE Trans. on Computers*, vol. 55, pp. 385 - 399, 2006.