Appears in the 10th International Symposium on Pacific Rim Dependable Computing (PRDC), March 3-5, 2004

Cache Scrubbing in Microprocessors: Myth or Necessity?

Shubhendu S. Mukherjee[1], Joel Emer[1], Tryggve Fossum[1], and Steven K. Reinhardt[1,2]

[1]Massachusetts Microprocessor Design Center, Intel Corporation
334 South Street, Shrewsbury 01545, Massachusetts

[2] Advanced Computer Architecture Lab
EECS Department, University of Michigan
1301 Beal Avenue, Ann Arbor, MI 48109

ABSTRACT

Transient faults from neutron and alpha particle strikes in large SRAM caches have become a major problem for microprocessor designers. To protect these caches, designers often use error correcting codes (ECC), which typically provide single-bit error correction and double-bit error detection (SECDED). Unfortunately, two separate strikes could still flip two different bits in the same ECC-protected word. This we call a temporal double-bit error. SECDED ECC can only detect—not correct—such errors.

This paper shows how to compute the mean time to failure for temporal double-bit errors. Additionally, we show how fixed-interval scrubbing—in which error checkers periodically access cache blocks and remove single-bit errors—can mitigate such errors in processor caches. Our analysis using current soft error rates shows that only very large caches (e.g., hundreds of megabytes to gigabytes) need scrubbing to reduce the temporal double-bit error rate to a tolerable range.

1. INTRODUCTION

Transient faults have emerged as one of the key challenges in microprocessor design today. These faults arise from energetic particles—such as neutrons from cosmic rays and alpha particles from packaging material—generating electron-hole pairs as they pass through a semiconductor device. Transistor source and diffusion nodes can collect these charges. A sufficient amount of accumulated charge may invert the state of a logic device—such as an SRAM cell, a latch, or a gate—thereby introducing a logical fault into the circuit's operation. Because this type of fault does not reflect a permanent failure of the device, it is termed *soft* or *transient*.

SRAM caches (Figure 1) are some of the largest structures in today's microprocessors and, hence, are most vulnerable to such transient faults. Caches are used heavily in microprocessors because they keep data closer to computation units and thereby improve a microprocessor's performance. Caches are typically divided up into numerous fixed-sized cache blocks containing the data, with typical sizes ranging between 16 and 128 bytes for each block. Additionally, each cache block has an associated address and state information (e.g., whether the block has been modified or is in read-only state).

Microprocessors typically have a hierarchy of caches, with smaller and faster caches closer to the computation units. Figure 1 shows a cache hierarchy with two levels of caches. Aided by an exponential increase in on-die transistor count, on-chip cache hierarchies have grown dramatically in size, with up to three levels, the largest being several megabytes in capacity [4]. The advent of large on-chip multiprocessors and increasing levels of on-chip multithreading [8] will make greater demands on the memory system. Consequently, cache sizes will continue to grow and approach several tens of megabytes in the near future.

To protect caches from transient faults, designers typically use a parity bit or SECDED (single-error correct, double-error detect) ECC code [11] to protect a group of data bits. Parity can detect single-bit errors and, depending on the cache policy (e.g., write-through and inclusion with higher levels), can even help recover the data transparently. In contrast, SECDED ECC codes detect all double-bit errors and always recover data transparently for single-bit errors.

The highest level of cache (e.g., the level 2 cache in Figure 1) typically uses ECC because it may hold modified data for a finite time before writing it back to main memory. Fortunately, most transient faults result in single-bit upsets for which SECDED ECC codes provide adequate protection. Nevertheless, a single neutron or alpha strike could still flip two bits located in close proximity. We call this a *spatial double-bit error*. SECDED ECC would be able to detect, but not correct, such a spatial double-bit error. To recover from spatial double-bit errors, designers typically interleave the ECC codes. That is, consecutive data bits would be protected by different ECC codes and the

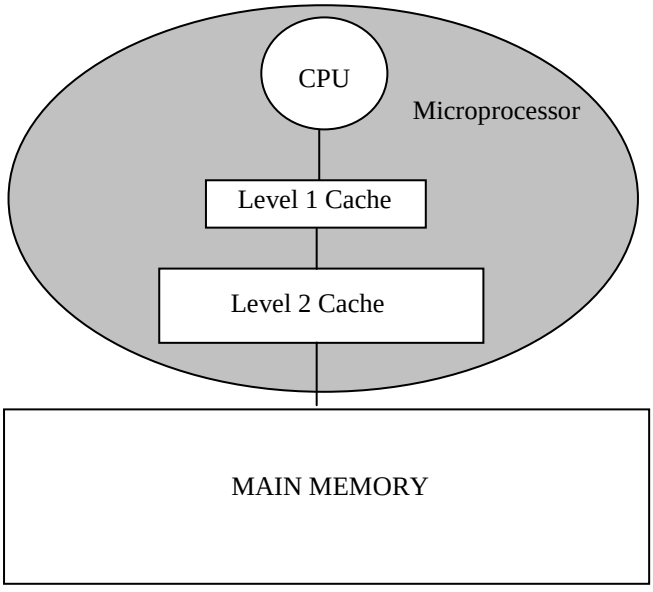

Figure 1. Memory hierarchy in a processor. The CPU accesses memory via multiple levels of caches. If it cannot find the data from the caches, then it sends a request to main memory to fetch the memory location it needs.

double-bit error would appear as individual single-bit errors for the respective ECC codes, thereby allowing transparent hardware recovery.

Unfortunately, SECDED ECC may not help a cache recover from a *temporal double-bit error*, which is the topic of this paper. A temporal double-bit error may occur when two different bits of the same ECC-protected data or code word are affected by single-bit errors at different times. Contrary to expectation, in most current microprocessors, the first single-bit error may not be corrected by the error checkers, unless the microprocessor reads the corresponding bits out of the cache. Thus, single-bit errors may accumulate in the cache and eventually develop into a temporal double-bit error when a second error occurs in the same set of data or code bits.

One solution to solving this problem is to use an ECC code that can correct double-bit errors. Unfortunately, this comes with a significant increase in the number of code bits. For a 64-bit data word, a minimum of 7 additional bits are required to correct all single-bit errors. SECDED ECC schemes add an eighth bit to guarantee detection of all double-bit errors. Correction of all double-bit errors requires 12 code bits. Thus, the memory overhead increases from 13% to 19%.

An alternate solution is to scrub the caches periodically [1]. Scrubbing has been widely used in the past in large main memories [12], which are typically protected with SECDED ECC. Scrubbing involves reading the bits from the cache, correcting any latent single-bit errors, recomputing the ECC, and writing the bits back. The read operation will provide the correct data even if a single-bit error is present; the writeback with the recomputed ECC will overwrite any existing single-bit error. If the scrubbing interval is short enough, the opportunity for a temporal double-bit error to arise is practically eliminated.

Scrubbing does, however, create extra overhead in terms of software and/or hardware. A pure software implementation may incur significant overhead in scrubbing because the software will have to access every block in a large cache. In contrast, current processors with very high memory bandwidths could potentially scrub in hardware entire multi-megabyte caches in a few milliseconds or less. Alternatively, if this is too high an overhead, then the processors could scrub in a *stealth* mode, where they incrementally scrub data blocks over time.

Interestingly, however, the size of a cache has a significant impact on the scrubbing interval. The bigger a cache is, the smaller the mean time to get a temporal double-bit error, and hence the shorter the necessary scrubbing interval. Conversely, smaller caches imply larger scrubbing intervals. In fact, if the mean time to get a double-bit error is large enough, then a microprocessor designer may choose to avoid scrubbing altogether.

This paper describes a method to determine how large a cache may be before scrubbing becomes a necessity, and, for a given cache size, what may constitute a reasonable scrubbing interval. Our analysis using current soft error rates for single-bit upsets shows that only systems with a total cache size in the range of hundreds of megabytes to gigabytes need to be scrubbed periodically. Current and near-future single processor cache sizes range up to tens of megabytes, so we would need a multiprocessor system comprising hundreds of processors to attain such a large total system cache size. Alternatively, in a few technology generations, a single processor may be able to have between 100 and 200 megabytes of L2 cache, which may then require scrubbing. However, scrubbing may not be justified for today's uniprocessors and small-scale multiprocessors.

The rest of the paper is organized as follows. Section 2 provides background on how we compute the soft error rate of a microprocessor. Section 3 shows how to compute the mean time to failure for a temporal double-bit error. Section 4 shows how this mean time to failure can be reduced using periodic scrubbing. Section 5 discusses related work. And, finally, Section 6 presents our conclusions.

2. BACKGROUND

Section 2.1 describes the failure metrics MTBF and FIT. Section 2.2 describes vulnerability factors and their impact on error detection and correction requirements.

2.1 MTTF and FIT

Vendors express a failure budget at a reference altitude in terms of Meant Time to Failure (MTTF) or Mean Time Between Failures (MTBF). MTBF equals the sum of MTTF and the repair time, which is usually very small compared to MTTF, if the repair is done in hardware. We will use MTTF in this paper as our metric because it is more appropriate for chip vendors, such as AMD or Intel®.

Failures are also often further classified as undetected or detected. The former are typically referred to as *silent data corruption* (SDC); we call the latter *detected unrecoverable errors* (DUE), which are the focus of this paper. Note that detected recoverable errors are not failures. Adding error detection (but not correction) to a structure eliminates SDC failures, converting those faults to DUE failures. Full error correction is required to reduce DUE errors. This paper focuses on DUE caused by double-bit memory errors.

Companies usually have target SDC and DUE rates for their microprocessors. For example, for its Power4 processor-based systems, IBM targets 1000 years system MTBF for SDC failures, 25 years system MTBF for DUE failures that result in a system crash, and 10 years system MTBF for DUE failures that result in an application crash [3]. Note that the processor MTBF must be significantly higher than the system MTBF, particularly for large multiprocessor systems.

Another commonly used unit for failure rates is FIT (Failure in Time), which is inversely related to MTBF. One FIT specifies one failure in a billion hours. Thus,

1000 years MTBF equals 114 FIT (10^9 / (24*365*1000)). A zero error rate corresponds to zero FIT and infinite MTBF. Designers usually work with FIT because FIT is additive, unlike MTBF.

To evaluate whether a chip meets its soft error budget—possibly via the use of error protection and mitigation techniques—microprocessor designers use sophisticated computer models to compute the FIT rate for every device—RAM cells, latches, and logic gates—on the chip. The result of this analysis is used to find the FIT rate of the chip using the following equation:

FIT rate of a chip = $\sum_{\text{i over all devices in the chip}}$ (raw FIT rate of i * vulnerability factor of i)

Current predictions show that typical FIT rate numbers for latches and SRAM cells at sea level vary between 0.001 – 0.01 FIT/bit ([15],[10],[6],[5]). This FIT/bit also increases with elevation. At 1.5km—altitude of Denver, Colorado—the FIT/bit is about 3.5x higher than at sea level. At 10km—typical altitudes for airplanes—the FIT/bit is approximately 100x higher [17]. Interestingly, however, the FIT/bit is projected to remain in this range or decrease slightly for next several technology generations [2][13], unless microprocessors aggressively lower the supply voltage to reduce the overall power dissipation of chip, thereby increasing FIT/bit, or move to fully-depleted silicon-on-insulator technology, which can dramatically decrease the FIT/bit of SRAM cells.

2.2 Vulnerability Factors

The effective FIT rate per bit is influenced by several *vulnerability factors* (also known as *derating factors* or *soft error sensitivity factors*). In general, a vulnerability factor indicates that, given conditions sufficient to cause a fault, there is a certain probability that an error will occur. The architectural vulnerability factor (AVF) expresses the probability that a visible system error will occur given a bit flip in a storage cell. As another example, when a level-sensitive latch is accepting data rather than holding data—typically 50% of the time—a strike on its stored bit may not result in an error, as the stored value will be overridden by the (correct) input value. We call this the *timing vulnerability factor*. The total vulnerability factor for a bit is the product of the timing and architectural vulnerability factors. For simplicity, we assume the timing vulnerability factor is already incorporated in the raw device fault rate. The computation of the device fault rate also includes some circuit-level vulnerability factors, which are beyond the scope of this paper.

The AVF can have a significant impact on the effective failure rate of a processor. Prior studies with statistical fault injection into RTL models have demonstrated AVFs of 1%-10% for latches [1] and 0% - 100% across a range of architectural and microarchitectural state bits [9].

For temporal double-bit errors in caches, there are two phenomena that reduce the vulnerability factor. First, if a block with a double-bit error is not in a modified state, then the correct data can be re-fetched from a higher level cache or from main memory. Note that the error must be in the data portion of the block, not in the cache tag or state, and that the cache control logic must support this refetch operation. Second, a block with a double-bit error may never be read by the processor. In this case, the block will be overwritten before the double-bit error is detected. In this latter case, cache scrubbing can be a detriment, as it may expose a latent double-bit error before the block is overwritten.

3. DUE RATE FROM TEMPORAL DOUBLE-BIT ERRORS WITH NO SCRUBBING

In this section we compute the FIT rate from temporal double-bit errors in the absence of any scrubbing. We assume that an 8-bit ECC code protects 64 bits of data, which we call a quadword. A temporal double-bit error occurs when two bits of this 72-bit protected quadword are flipped by two separate neutron or alpha particle strikes. We only concern ourselves with strikes in the data portion of the cache blocks. First, we compute FIT rate from temporal double-bit errors without the use of AVFs. Then, we show the impact of AVFs on these numbers.

To compute the FIT contribution of temporal double-bit errors, let us define the following terms:

- Q = number of quadwords in the cache memory. Each quad-word consists of 64 bits of data and 8 bits of ECC. Thus, we have a total of 72 bits per quadword.
- E = number of random single-bit errors that occur in the population of Q quadwords.

Given E single-bit errors in Q different quadwords, the probability that error E+1 will cause a double-bit error is E/Q. Let $P_d[n]$ = the probability that a sequence of n strikes causes n-1 single-bit errors (but no double-bit errors) followed by a double-bit error on the nth strike. Clearly $P_d[1] = 0$. $P_d[2]$ is the probability that the second strike hits the same quadword as the first strike, or 1/Q. $P_d[3]$ is the probability that the first two strikes hit different quadwords (i.e., $1-P_d[2]$) times the probability that the third strike hits either of the first two quadwords that got struck (i.e., 2/Q). Following this formulation, we get

- $P_d[2] = 1 / Q$
- $P_d[3] = [(Q-1)/Q] * [2/Q]$
- $P_d[4] = [(Q-1)/Q] * [(Q-2)/Q] * [3/Q]$
- …
- $P_d[E] = [(Q-1)/Q] * [(Q-2)/Q] * [(Q-3)/Q] * \ldots * [(Q-E+2)/Q] * [(E-1)/Q]$

Then, the probability of a double-bit error after a time period T = $\sum P_d[N]$ * P[N strikes in time T] for all N. Using this equation, we can solve for the expected value of T to derive the MTTF to a temporal double-bit error.

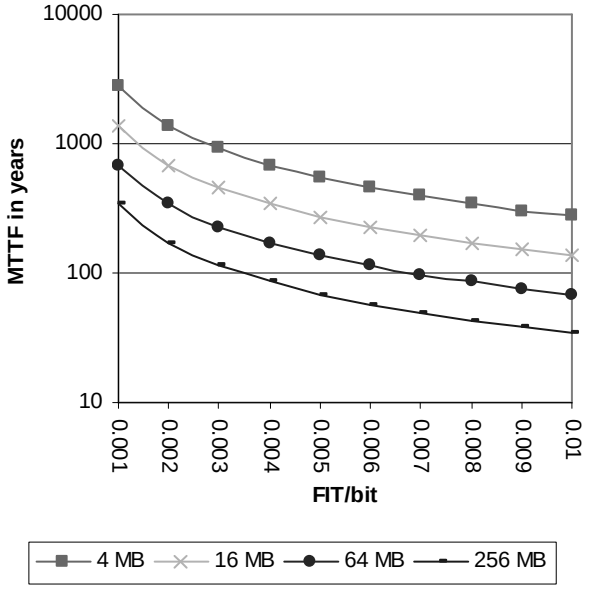

Figure 2. MTTF to a temporal double-bit error for different cache sizes (expressed in MB or megabytes).

Fortunately, there is an easier way to calculate this MTTF. Assume that M is the mean number of single-bit errors needed to get a double-bit error. Then, the MTTF for a temporal double-bit error = M * MTTF of a single-bit error. (Similarly, the FIT rate for a double-bit error = 1/M * FIT rate for a single-bit error.) A simple computer program can calculate M very easily as the expected value of $P_d[\cdot]$.

As an example, consider a 32 megabyte data cache. This cache has 2^{22} quadwords. Let us assume that an SRAM cell has an average FIT rate of 0.001. The single-bit FIT rate for the entire cache is $0.001 * 2^{22} * 72 = 3.02 * 10^5$, i.e. the MTTF is $10^9 / (3.02 * 10^5) = 3311$ hours. Using a computer program, we find that M = 2567. Then, the MTTF to a double-bit error = 3311 * 2567 hours = 970 years.

Using a Poisson distribution, Saleh, et al. [12] came up with a different method of computing such MTTFs for main memory systems (not on-chip caches), but their calculations match ours for large multi-megabyte memories. Saleh, et al.'s derivation shows that the MTTF for such temporal double-bit errors is equal to [1 / (72 * f)] * sqrt(pi / 2Q), where f = FIT rate of a single bit. Thus, using Saleh, et al.'s method, we find that MTTF for double-bit errors for a 32 MByte cache and FIT/bit of 0.001 (f) is $0.0085 * 10^9$ hours or 970 years.

Figure 2 shows the MTTF (in years) for such double-bit errors from two separate neutron or alpha particle strikes to the same quadword in a cache. For small caches—between 4 and 16 MB—the MTTF for temporal double-bit errors ranges from 137 years to 2746 years, which is significantly greater than the 10 years MTBF required for

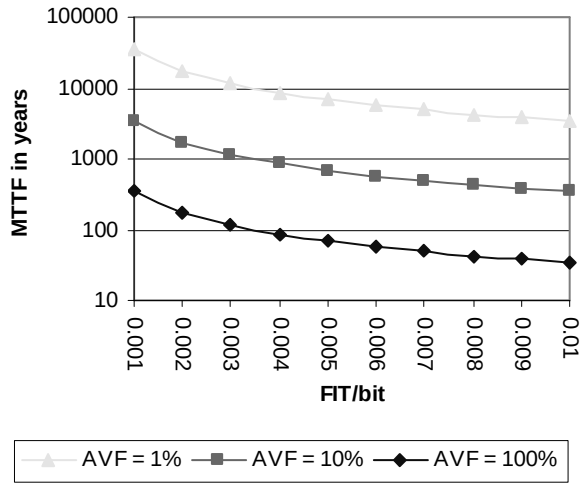

Figure 3. MTTF to a temporal double-bit error for a 256 MB cache for different AVF assumptions.

IBM's DUE rate specification. Thus, for caches less than 16 MB, the contribution of temporal double-bit error towards the whole DUE rate is minimal. Similarly, for FIT/bit less than 0.003, the contribution of large caches in the range of 64 to 256 MB is a small fraction of the total DUE target rate of 10 years MTBF. However, when the FIT/bit is greater than 0.003, the DUE contribution of temporal double-bit errors could become significant. For example, for a processor with 256 MB cache and a FIT/bit of 0.01, the total contribution DUE contribution from temporal double-bit errors is about 29% of a target DUE rate of 10 years.

The above calculation does not factor in the reduced error rates because of the architectural vulnerability factor (Section 2.2). Prior studies [9][1] have shown that AVFs of many structures are in the 10-30% range. Other studies (e.g., [14], [7]) with small caches—holding tens of kilobytes of data—with CPU95 benchmarks (see http://www.spec.org) and other numerical applications suggests that AVFs could range between 15 and 60% . For larger multi-megabyte caches with lower average utilization of cache blocks we expect the AVF to be in the 10-20% range. Nevertheless, the AVF is highly dependent on specific applications and further work needs to be done to validate the AVF numbers for large caches.

Figure 3 shows that for a 256 MB cache, 0.01 FIT/bit, and AVF of 10%, the MTTF for a temporal double-bit error is 343 years, which makes it a very small fraction of a possible target DUE rate of 10 years. Thus, for uniprocessor caches ranging from 4 MB to 256 MB, FIT/bit in the range of 0.001 - 0.01 FIT/bit, and AVF of around 10%, our DUE rate for temporal double-bit errors is sufficiently small. Hence, the designer may choose not to scrub these caches.

Nevertheless, large multiprocessor systems composed of multiple uniprocessors—each with tens of megabytes of

caches—may require scrubbing to reduce the DUE rate. For example, a multiprocessor or cluster composed of 64 processors, each with 256 MB of cache, a FIT/bit of 0.01, and AVF of 10%, would have a DUE rate of 4.3 years from temporal double-bit errors. Designers may opt to support cache scrubbing in such a system.

Interestingly, the MTTF contribution from double-bit errors for a system with multiple chips cannot be computed in the same way as we do for single-bit errors. If chip failure rates are independent, then a system composed of two chips, each with an MTTF of 100 years, has an overall MTTF of 100 / 2 = 50 years. Unfortunately, double-bit error rates are not independent, because the MTTF for a double-bit error is not a linear function of the number of bits. This is also evident in Saleh, et al.'s equation [12], which shows the rate of such double-bit errors is inversely proportional to the square root of the size of the cache. Thus, quadrupling the cache size halves the MTTF for double-bit errors, but does not reduce it by a factor of four.

4. DUE RATE FROM TEMPORAL DOUBLE-BIT ERRORS WITH SCRUBBING

Fixed-interval scrubbing can dramatically improve the MTTF of the cache subsystem. By scrubbing a cache block we mean that for each quadword of the block, we read it, compute its ECC, and compare the computed code with the existing ECC code. For a single-bit error, we correct the error and rewrite the correct ECC code into the cache. By fixed-interval scrubbing, we mean that all cache blocks in the system are scrubbed at a fixed interval rate, such as every year or every month. Scrubbing can help improve the MTTF because it removes single-bit errors from the cache system (protected with SECDED ECC), thereby reducing the probability of a future temporal double-bit error.

Even in systems without active scrubbing, single-bit errors are effectively scrubbed whenever a quadword's ECC is recalculated and rewritten. This occurs when new data is written to the cache, either because the cached location is updated by the processor, or because the cached block is replaced and overwritten with data from a different memory location. In some systems, a single-bit error detected on a read will also cause ECC to be recalculated and rewritten. The key difference between these passive updates and active scrubbing is that the former provides no upper bound on the interval between ECC updates.

To compute the MTTF with scrubbing, let us define the following terms:

- I = scrubbing interval
- N = number of scrubbing intervals to reach MTTF (with scrubbing active at the end of each interval I)
- pf = probability of a double-bit error from temporally separate neutron or alpha strikes in the interval I

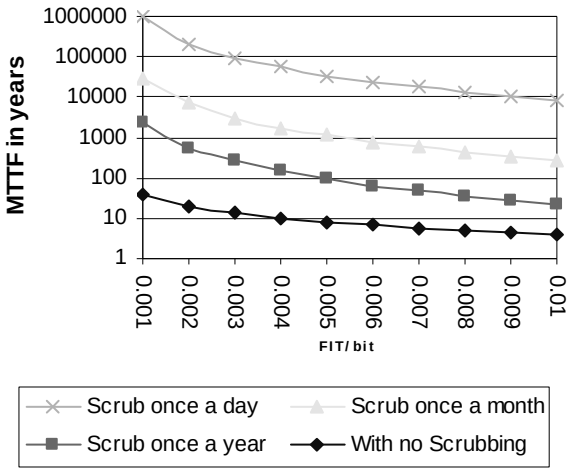

Figure 4. Impact on fixed-interval scrubbing (once a year, once a week, and once a day) on the MTTF of temporal double-bit errors for a system with 16 Gigabytes of on-chip cache.

Then, by definition, MTTF for a temporal double-bit error = N * I. Assuming each such scrubbing interval is independent, the probability that we have no double-bit error in the first N intervals followed by a double-bit error in the N+1th interval is $(1\text{-pf})^N$ * pf. Thus, N is the expected value of a random variable with probability distribution function $(1\text{-pf})^N$ * pf. So, given an interval I, we compute the number of single-bit errors (say S) that can occur in that interval. pf is equal to the sum of the probabilities of a double-bit error, given 2, 3, 4, …, S errors. This probability can be computed the same way as described earlier for a system with no scrubbing (Section 3). Thus, given pf and I, we can easily compute N using a simple computer program.

Figure 4 shows how scrubbing once a year, month, and day can improve the MTTF numbers for a system configured with 16 GBbyte of on-chip cache and assuming an AVF of 100%. This could, for example, arise from 64 processor multiprocessor or cluster with each processor having 256 MB of on-chip cache. Clearly, fixed interval scrubbing can significantly improve the MTTF of a processor and system's cache subsystem. Thus, for example, for a FIT/bit of 0.001, the MTTF for temporal double-bit errors are 40 years, 2281 years, 28172 years, and 959267 years, respectively, for a system with no scrubbing, scrubbing once a year, scrubbing once a month, and scrubbing once a day.

Our numbers come close to Saleh, et al.'s [12] prediction of MTTF for temporal double-bit errors with fixed-interval scrubbing. Saleh, et al.,'s closed form for this MTTF is 2 / [Q * I * (f * 72)^2]. With a FIT/bit (i.e., f) of 0.001, Saleh et al., would predict MTTFs of 2341 years, 28484 years, and 854515 years, respectively, for a system with scrubbing once a year, once a month, and once a day.

5. RELATED WORK

As discussed in Sections 3 and 4, Saleh, et al. [12] have proposed closed form equations for the MTTF for a temporal double-bit error. Our results validate Saleh, et al.'s equations. However, unlike Saleh, et al.'s work, our formulation is based on discrete probability distributions and perhaps somewhat easier to understand and validate.

Further, we improve upon Saleh, et al.'s work by using realistic FIT rates and AVFs for SRAM caches and comparing the temporal double-bit error rate with target DUE rate set by companies, such as IBM. Our results show that only large caches—in the range of hundreds of megabytes or gigabytes—would need scrubbing. In contrast, Saleh, et al. evaluated their equations for DRAM memories with a FIT/bit of 42, which is significantly higher than current FIT/bit for SRAM caches. Clearly, using such high FIT/bit numbers for SRAM caches would lead to a conclusion different from ours.

6. CONCLUSIONS

Transient faults from neutron and alpha particle strikes have become one of the key challenges in the microprocessor industry today. Because SRAM caches are some of the largest structures in today's microprocessors, they are most prone to such strikes. Designers protect these caches with SECDED (single error detect, double error correct) ECC (error correcting codes). These codes comprise of 8 bits of code to protect every 64 bits of data. Unfortunately, for very large caches, there is still a finite probability that two separate neutron or alpha particles could strike the same data word protected by the same ECC code. We called this phenomenon a temporal double-bit error.

Currently, microprocessor vendors (e.g., AMD) scrub caches to reduce the temporal double-bit error rate [1]. Scrubbing involves reading bits from the cache, computing the ECC, checking the generated ECC against the stored ECC, and correcting any existing single-bit errors. If the scrubbing interval is shorter than the average duration between two single-bit errors in the same set of data and code bits, then this would potentially eliminate almost all temporal double-bit errors.

In this paper, we developed a methodology to compute the MTTF to such a temporal double-bit error. Our analysis showed that small caches—less than tens of megabytes—have a very low temporal double-bit error rate and, hence, do not need to be scrubbed. In contrast, large caches—in the range of hundreds of megabytes to gigabytes—may need to be scrubbed to reduce the temporal double-bit error rate to a tolerable range.

ACKNOWLEDGMENTS

We would like to thank Stephen Felix, who helped us develop some of the equations in this paper. We would also like to thank Geoff Lowney for feedback on initial drafts of this paper.

REFERENCES

[1] AMD, "BIOS and Kernel Developer's Guide for AMD Athlon™64 and AMD Opteron™ Processors," Publication # 26094, Revision 3.04, Issue date July 2003, http://www.amd.com/usen/assets/content_type/white_papers_and_tech_docs/26094.PDF.

[2] Robert Baumann, "Soft Errors in Commercial Semiconductor Technology: Overview and Scaling Trends," *IEEE 2002 Reliability Physics Tutorial Notes, Reliability Fundamentals*, pp. 121_01.1 – 121_01.14, April 7, 2002.

[3] D.C.Bossen, "CMOS Soft Errors and Server Design," *IEEE 2002 Reliability Physics Tutorial Notes, Reliability Fundamentals*, pp. 121_07.1 – 121_07.6, April 7, 2002.

[4] L.Gwennap, "Alpha 21364 to Ease Memory Bottleneck," *Microprocessor Report*, 12(14): 12-15, October 26, 1998.

[5] S.Hareland, J. Maiz, M.Alavi, K.Mistry, S.Walstra, and C.Dai, "Impact of CMOS Scaling and SOI on soft error rates of logic processes," Symposium *on VLSI Technology Digest of Technical Papers*, 2001.

[6] T.Karnik, B.Bloechel, K.Soumyanath, V.De, and S.Borkar, "Scaling trends of Cosmic Rays induced Soft Errors in static latches beyond 0.18μ," *Symposium on VLSI Circuits Digest of Technical Papers*, 2001.

[7] S.Kim and A.K.Somani, "Area Efficient Architectures for Information Integrity in Cache Memories," *Proceedings of the 26th Annual International Symposium on Computer Architecture (ISCA)*, pages 246 – 255, May, 1999.

[8] Kevin Krewell, "Sun Weaves Multithreaded Future," *Microprocessor Report*, Volume 17, Archive 4, April 2003.

[9] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin, "A Systematic Methodology to Computer the Architectural Vulnerability Factors for a High-Performance Microprocessor," *36th Annual International Symposium on Microarchitecture (MICRO)*, December 2003.

[10] Eugene Normand, "Single Event Upset at Ground Level," *IEEE Trans. on Nuclear Science*, Vol. 43, No. 6, Dec. 1996.

[11] Dhiraj K. Pradhan, "Fault-Tolerant Computer System Design," Second print 2003, Computer Science Press.

[12] A.M.Saleh, J.J.Serrano, and J.H.Patel, "Reliability of Scrubbing Recovery Techniques for Memory Systems," IEEE Transactions on Reliability, Vol.39, NO.1, April 1990.

[13] P.Shivakumar, M.Kistler, S.W.Keckler, D.Burger, and L.Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinatorial Logic," *Dependable Systems and Networks*, 2002.

[14] A.K.Somani and K. Trivedi, "A cache error propagation model," *Proceedings of the International Symposium on Pacific Rim Fault Tolerant Computing*, pages 15 – 21, Dec. 1997.

[15] Y.Tosaka, S.Satoh, K.Suzuki, T.Suguii, H.Ehara, G.A.Woffinden, and S.A.Wender, "Impact of Cosmic Ray Neutron Induced Soft Errors, on Advanced Submicron CMOS circuits," *VLSI Symposium on VLSI Technology Digest of Technical Papers*, 1996.

[16] Nicholas Wang and Sanjay Patel, "Modeling the Effect of Transient Errors on High Performance Microprocessors," *Center for Circuits, Systems, and Software (C2S2), 2nd Annual Review, Berkeley*, March 18-19, 2003.

[17] J.F.Ziegler, "Terrestrial cosmic rays," *IBM J. of Research and Development*, pp. 19 – 39, Vol. 40, No. 1, Jan. 1996.