

# Architectural Support for Mitigating DRAM Soft Errors in Large-Scale Supercomputers

Dennis Abts, John Thompson and Gerald Schwoerer

Cray Inc.

{dabts, johnt, gfs}@cray.com

**Abstract**—Modern DRAM devices are built from high-density, low-voltage integrated circuits that are becoming increasingly susceptible to influences from external factors such as electrical noise, process variation, and natural radiation (particle-induced upsets). Errors resulting from these effects are referred to as “soft errors” since although they corrupt the state of a storage element, they generally do not cause any permanent damage to its underlying circuitry. The rate at which these events occur is referred to as the soft error rate (SER) and has been steadily increasing as transistor geometries shrink. In particular, large-scale supercomputers, where high volumes of memory parts are utilized in a single system, must be designed to address and tolerate otherwise crippling SERs. The Cray BlackWidow large-scale multiprocessor is one such system where particular attention has been paid to mitigating the effects of soft errors in main memory. In this paper we describe several mechanisms used in the Cray BlackWidow memory system to combat soft errors and illustrate how, together, these mechanisms work to address a broad range of memory errors.

## I. INTRODUCTION

As transistor geometries continue to shrink, CMOS devices are becoming increasingly susceptible to electrical noise, radiation-induced particle upsets, negative-bias temperature instability (NBTI), and erratic bits [1]. Further exacerbating the problem, semiconductor process variation is widening as the gate-oxide thickness of these devices approaches only a handful of atoms – making some of the devices inherently more vulnerable to single-bit upsets, or “bit flips”, than others. These “soft errors” are defined as any error which effects the data stored by a device without doing any permanent damage to the device’s underlying circuitry. In other words, once the soft error is corrected, for example, by being overwritten with a new data value, the storage device is no more likely to fail in time than it was before the initial failure. The rate at which soft errors occur in a system is referred to as the soft error rate, or SER, a term first coined in the 1970s, when seemingly random and intermittent memory device failures were first observed. Today, SER is of ever increasing concern to system designers building large-scale multiprocessors, such as BlackWidow, consisting of tens-of-thousands of processing nodes and hundreds of terabytes of DRAM memory.

Typically, SERs are represented in units of failures-in-time, with a single FIT being defined as one failure in a billion device hours. Schroeder and Gibson [2] studied different classes of failure types and found a common trend in how various failure rates change as a function of system age. They described that as a system matures, fewer failures are

attributable to software and most of the remaining failures are caused by *hardware* or *unknown* causes.

### A. BlackWidow Overview

The Cray BlackWidow system [3] is a globally shared-memory multiprocessor designed to run demanding applications with high communication requirements. Memory is accessed using direct load/store instructions to any memory location in the system, and unlike conventional microprocessors, it can support thousands of outstanding global memory operations. The memory system is highly parallel with 64 independent memory controllers for each four-way node, providing a maximum memory capacity of 64 Gbytes<sup>1</sup> of DDR2 memory – 320 DDR2 devices per node. With an estimated SER of 10 FIT per DDR2 device, we are faced with the daunting task of building reliable *systems* [4] from increasingly unreliable *components* [5].

Each memory reference is tagged with a  $k$  bit which indicates whether the reference was generated by a user-level application ( $k=0$ ), or the operating system ( $k=1$ ). This information allows one to easily differentiate errors resulting at the user-level (application failure) from those taken at the system-level (system crash). The vast majority of memory references will be user-level references; however, if we cannot distinguish between the two types of references, then it becomes much more complicated or impossible for the exception handler to prevent a single memory error from cascading. In order to promote overall system reliability, faults must be contained as much as possible. In the best case, a fault may be recovered from gracefully, for example, via a retry mechanism. In other cases the effected application may be terminated and in the worst cases the result may be a system crash. In the case where an application takes a fault, we want to guarantee that the fault will not affect other applications running on the system; the  $k$  bit allows us to accomplish this goal.

### B. Memory System Organization

Each BlackWidow processor interfaces with 16 custom-designed memory controller chips, called “Weaver”. Each Weaver chip is implemented in 90nm ASIC technology and has a 512 Kbyte L3 cache, for a total of 8 Mbytes of L3 cache per node, and provides a directory-based cache coherence engine for the memory that it interfaces with. In addition, a

<sup>1</sup>The memory controller supports *stacked* memory parts for additional capacity.

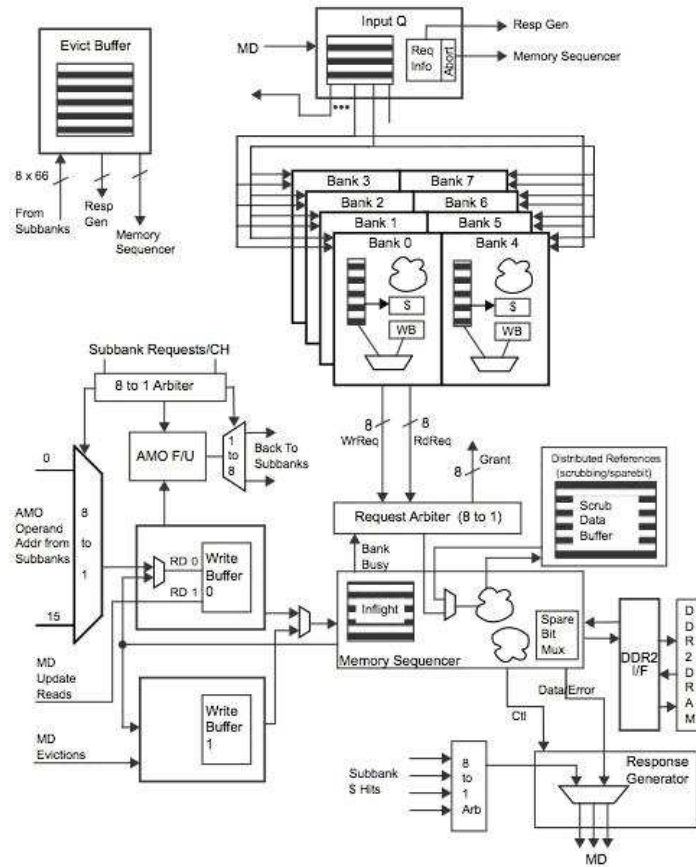


Fig. 1. BlackWidow memory manager block diagram.

single Weaver has four *memory directory* (MD) blocks, each with its own *memory manager* (MM) to service the memory backed by that node. Furthermore, each memory manager provides:

- arbitration and scheduling of the DDR2 memory devices according to bank, row, and column dimensions to maximize the effective pin bandwidth of the DDR2 devices,
- fine-grained atomic memory operations (AMOs),
- memory refresh and necessary housekeeping functionality to maintain the memory cells,
- automatic scrubbing of memory to repair single-bit upsets,
- data poisoning and deferred error handling,
- an auto-retry mechanism to cope with transient multi-bit errors,
- a comprehensive set of status collecting memory mapped registers, and
- spare-bit insertion to repair persistent memory errors.

Error mitigation techniques, such as ECC/SECDED codes and other modified Hamming codes designed to correct a single-bit error are well understood and have been successfully employed for increased memory reliability. Using schemes such as chip kill correct, it is possible to extend these techniques to allow multiple adjacent memory bits to be

corrected. However, this approach is generally more expensive since it requires additional check bits and associated logic. Furthermore, these schemes are only advantageous in the case where soft errors are clustered, effecting multiple adjacent bits within a memory word.

The fault-tolerant DRAM scheduling, and hardware support for scrubbing, auto-retry, and spare-bit insertion address the growing problem of soft errors in DRAM devices. Many of these techniques work transparently together to improve both *application* and *system* reliability. In the following sections we describe the micro-architecture of the BlackWidow memory manager, placing an emphasis on both the operation and the implementation of several key error-handling mechanisms that, together, allow for the detection, recovery, and logging of single and multi-bit memory errors.

## II. THE BLACKWIDOW MEMORY MANAGER

The memory manager (Figure 1) serves as the interface between the BW processor and the DDR2 memory devices. It is responsible for arbitration of new requests while observing various delays and scheduling constraints imposed by the memory devices. Moreover, it is principally responsible for the detection, recovery (when possible), and reporting of all encountered memory errors.

Each 32-bit data word in main memory is protected with a 7-bit error correcting code (ECC) that will correct any single-bit error and detect any double-bit error (SECEDED). The DRAM is organized as five 8-bit devices, for a total of 40 bits. Only 39 of the 40 bits are used by data and ECC, leaving a single bit to be used as a “spare” bit. A special ECC value, with a characteristic syndrome of 0x7F, is used to designate a data word that is *poisoned*; that is, the data contained in the associated location is known to be corrupted. An example of when memory-bound data is poisoned occurs when a hardware error in the cache results in corrupted dirty cache data. In this case, when the data is written back to main memory, the corrupted locations are poisoned by the memory manager to prevent any subsequent loads from using the corrupted data.

When a read response returns from the DRAM interface, the ECC is checked for integrity. If a single-bit error is detected, the data is corrected and used as normal; whereas, a double-bit error will result in an error response being returned to the requesting processor. An atomic memory operation (AMO) that returns a multi-bit error (MBE) on the read portion of the read-modify-write cycle will poison the write data, as will all errant data provided to the memory manager as part of a write-back from cache.

If an error is detected in data returned by the DRAM, the error is logged in an MBE error table. System software can interrogate this structure to inspect error signatures and diagnose faulty memory parts with error rates that warrant replacement. There is an analogous SBE table which can also be used by software to diagnose failures.

### III. REQUEST AUTO-RETRY

In addition to conventional ECC, Weaver implements a request *auto-retry* mechanism for read operations that experience non-poisoned multi-bit errors (MBE). The retry protocol allows multiple attempts to read the memory and disambiguate a persistent MBE soft error that caused multiple “bit flips” in the memory device from a soft error induced by electrical (simultaneous switching) noise, operating temperature variance, or marginal electrical signaling. Furthermore, it is possible for a combination of errors to exist simultaneously and give the appearance of a single event causing multiple-bit errors. For example, a single bit flip event in main memory, combined with electrical noise on the memory interface pins can result in a soft error that contains multiple bits in error. By *retrying* a faulty memory read operation, the hardware can distinguish between an intermittent error and a persistent error.

All memory references that return an error response that is non-poisoned are retried. When an MBE is detected, the memory sequencer will immediately (subject to the bank cycle time of the device) schedule the retry operation. Arbitration logic in the memory sequencer gives the retry request priority so that no other requests are allowed to be reordered in front of the retry operation; that is, the next reference to the bank where the MBE occurred is guaranteed to be the retry. This retry operation is repeated  $n$  times (where  $n$  is set by a configuration register in the memory manager) or until the checksum of the

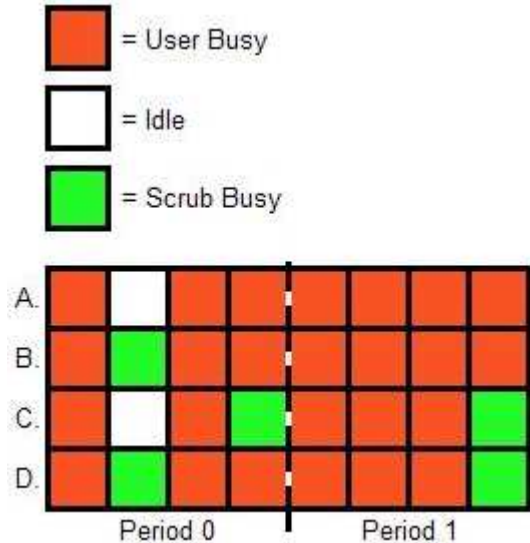


Fig. 2. Scrub scheduling example: (A) No scrubbing, (B) Completely non-intrusive scrubbing schedule without QoS guarantees, (C) Purely periodic scrubbing schedule, (D) Periodic schedule with idle preference.

read data is valid (no error). After  $n$  failed attempts to read the faulty memory location, the memory sequencer returns an error response and the error is logged as a persistent MBE in the error table. An exception is raised to the operating system so the application that observes the error can be terminated, and the memory page with the faulty location can be removed from the free-page list.

The memory sequencer maintains a list of requests that are currently in-flight for each DRAM memory bank and keeps track of memory manager transaction identifiers (MMTIDs) that are used to uniquely identify each request. When an MBE occurs on a request that will be retried, the memory reply is marked with a squash bit to indicate that this request will be retried. This prevents the memory directory from getting multiple read replies for a single request that was retried because of an MBE. With memory retry, the BlackWidow system is able to cope with intermittent soft errors with very little impact on performance (slightly more memory latency is incurred for the retry) and only a modicum of additional hardware complexity for the retry logic.

### IV. MEMORY SCRUBBING

Assuming that soft errors follow a uniform distribution in memory, the longer a word of data lives in DRAM, the more likely it will be to suffer the effects of any number of soft errors. In the worst case, a sufficient number of bits will be upset to result in silent data corruption. In an effort to prevent independent single-bit errors from compounding to form application crippling multi-bit errors or worse, the memory manager implements a *hardware-based memory scrubbing* engine. The scrubbing engine is capable of cycling through memory, reading and correcting any encountered single-bit errors by writing back corrected data. At the same time, non-poisoned multi-bit errors are retried  $n$  times by the scrubbing

engine, just as other requests are.

In order to make the scrubbing engine as non-intrusive as possible, it is desirable to perform scrub reads when the DRAM channel is otherwise idle. At the same time, certain quality of service (QoS) guarantees must be made, ensuring that the entire DRAM part is scrubbed with a specifiable frequency. To satisfy these requirements, Weaver uses a scheme in which a DRAM scrub cycle is broken up into fixed periods, each of which includes a single scrub read request. Furthermore, each scrub period is divided into two distinct time regions, the first of which will perform an early scrub read if no other traffic is present at the eight-to-one request arbiter. However, at some point the scrub request must be considered a priority, and in the second phase of each period, user requests will be blocked out allowing the DRAM to idle and make way for the pending scrub request. The benefit of such a scheme is illustrated in Figure 2.

As an alternative to the DDR2 device auto-refresh capability, Weaver implements a *distributed* refresh algorithm that avoids the bank quiescence necessary with auto-refresh, and consume less pin bandwidth than auto refresh. However, with higher-density parts (with more rows that need to be refreshed in a distributed manner) the benefit is more modest. Distributed refresh works by interleaving reads requests, whose purpose is to merely touch and refresh memory, into the normal request stream. When distributed refresh is enabled, scrubbing is piggy-backed on top of it, allowing all of the scrub reads to be performed at no cost. With memory scrubbing, the BlackWidow memory system is able to cope with uniformly distributed DRAM soft errors without sacrificing memory bandwidth.

## V. SPARE-BIT INSERTION

The DDR2 devices provide 40-bits of data where only 39-bits are needed for data and ECC. The left-over data bit can be multiplexed into the data path using a series of two-to-one multiplexers. The control of each mux is selected individually according to the bit position that is to be skipped so that the “spare” bit is used instead. This spare-bit insertion can be done with relatively little interaction with the normal request path.

The spare-bit logic is an adjunct to the refresh/scrubbing logic described earlier. If a spare-bit insertion sequence is run, the scrub logic is set to always execute a 32-byte read-modify-write sequence to insert the new spare-bit selection. If the MM is set to use auto-refresh instead of the distributed refresh/scrub sequence, the same scrubbing and spare-bit functions execute with the likely difference that the specified inter-request wait interval will be set to a higher value. However, it may be desirable to change the spare-bit location in a more timely fashion in an effort to avoid potentially compounding errors. In such a case, the inter-request counter may be set to a lower value just prior the spare-bit insertion pass, allowing it to complete in very little time.

Since, obviously, this spare-bit cannot be inserted instantaneously, there must be a mechanism to track which memory words use the newly inserted spare-bit location and which use

the previous location. To accomplish this while incurring a minimal cost, a single pointer is used to mark the ending boundary between words that use the new and words that use the old spare-bit locations. The insertion sequence is only started once the distributed refresh and scrubbing engine has rolled over, guaranteeing the starting bound to always be at address zero.

In order to determine which bit position make for good spare-bit candidates, a set of single-bit error *histograms* are provided – with a bin for each of the 40 bit positions. These histograms track the location of all single bit errors and can be periodically polled by system software. Although particle-induced soft errors in a memory system should be uniformly distributed within the memory space, the memory space is not necessarily uniformly accessed. Memory references will likely possess some locality. Therefore, commonly observed single-bit errors will be exposed by reading the histogram registers and examining the frequency of errors. Using the histogram data provides an informed decision about which failing bit to replace with spare-bit insertion. Through the use of spare-bit insertion, the BlackWidow system can overcome what appear as column-clustered soft errors that may, for example, be the result of a noisy memory channel or request locality.

## VI. CONCLUSIONS

This paper describes the micro-architecture of the Cray BlackWidow memory manager, and the features necessary to cope with increasingly unreliable memory devices. We describe a combination of architectural features: spare-bit insertion, memory auto-retry, data poisoning, and memory scrubbing that are necessary ingredients for building a large-scale multiprocessor with thousands of nodes and terabytes of system memory. Although the BlackWidow system is still in early prototype stages, we are able to observe direct benefits of these reliability features even for a small system size. For example, disabling the memory retry feature results in more multi-bit errors that are visible to the application. The error logging and histogramming of memory errors provides valuable observability during system bring-up of early prototype hardware.

## ACKNOWLEDGMENT

The authors would like to thank everyone involved on the BlackWidow team.

## REFERENCES

- [1] M. Agostinelli, J. Hicks, J. Xu, B. Woolery, K. Mistry, K. Zhang, S. Jacobs, J. Jopling, W. Yang, B. Lee, M. Mehalel, P. Kolar, Y. Wang, J. Sandford, D. Pivin, C. Peterson, M. DiBattista, S. Pae, M. Jones, S. Johnson, and G. Subramanian, “Erratic fluctuations of SRAM cache Vmin at the 90nm process technology node,” in *Electron Devices Meeting 2005*, Dec 2005, pp. 655–658.
- [2] B. Schroeder and G. Gibson, “A large-scale study of failures in high-performance computing,” in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, June 2006.
- [3] S. Scott, D. Abts, J. Kim, and W. J. Dally, “The BlackWidow High-radix Clos Network,” in *Proc. of the International Symposium on Computer Architecture (ISCA)*, Boston, MA, June 2006, pp. 16–28.

- [4] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The Soft Error Problem: An Architectural Perspective," in *Proceedings of the 11th International Conference on High-Performance Computer Architecture (HPCA2005)*, 2005.
- [5] A. Johnston, "Scaling and Technology Issues for Soft Error Rates," in *Proceedings of the 4th Annual Research Conference on Reliability*, Stanford, CA, October 2000.