

Fault-Tolerant Parallel Processing with Real-Time Error Detection and Recovery

Chung-Ho Chen

Dept. of Electrical Engineering
University of Washington
Seattle, WA 98195

Arun K. Somani

Dept. of EE and Dept. of CSE
University of Washington
Seattle, WA 98195

Abstract

The performance of parallel processing for real-time application is very sensitive to the reliability of the system. This paper presents a unique error recovery mechanism based on new cache states, verified and non-verified, to detect and recover errors produced by the processor or cache memory or both due to transient faults. The proposed scheme remedies the insufficiency of the error-correcting code when facing with processor transient fault. This cache-based recovery method not only recovers errors in a local cache memory but also prevents the propagation of errors to other caches. We show that this new error recovery scheme can be easily integrated with existing cache coherency protocols.

1 Introduction

Using redundancy to achieve hardware fault-tolerance is not an uncommon approach for critical real-time applications [1]–[5]. A fault in a processor or cache memory may cause the processor execution states to diverge from the majority computation sequence by the corrupted cache or the internal CPU states. The fault could be permanent, intermittent, or transient. Studies have shown that a large fraction of errors detected are caused by transient faults. If an error occurs in a cache line by a direct transient fault in the cache memory or the faulty processor due to a write access or both, the immediate recovery of the erroneous cache data can be crucial for the system's reliability because

- the recovery of faulty data in a cache line can prevent the cache memory from being corrupted by the running away processor. Thus, the time consuming reload process for the main memory

or the re-synchronization of the failed channel can be possibly avoided.

- the real-time error recovery prevents unnecessary initiation of the reconfiguration process since a transient fault may disappear at or before the time the reconfiguration is finished.

Conventional method of using error-correcting codes (ECC) [6] in the cache memories to improve system reliability has two major limitations. First, the errors in the cache memory can not exceed the error-correcting capability of the code, otherwise the processor may use erroneous data. Second, errors induced from the transient faults of a processor can not be detected if the incorrect result is written into the cache memory as seen a *valid* data. Our proposed scheme overcomes the insufficiency of the error-correcting code when facing with processor transient fault.

Previous researches have proposed using caches to assist error recovery [7, 8]. These methods are primarily based on checkpoint rollback. Most researches have assumed that the detection mechanisms are provided, and errors are detected immediately without corrupting the checkpoints. Unlike these works, the methodology we proposed allows both the processor and the cache memory to be liable for transient faults, and we integrate the detection and recovery schemes as a whole.

To ensure the preservation of a correct checkpoint, the cache-based rollback schemes require, in general, significant overheads in terms of time and memory spaces to save the checkpoint, especially for a large cache memory. Computation is also lost when rollback occurs. This is quite undesirable in real-time processing because (1) a process may miss its deadline, and (2) frequent checkpointing due to I/O events affects performance. Instead of using cache-based

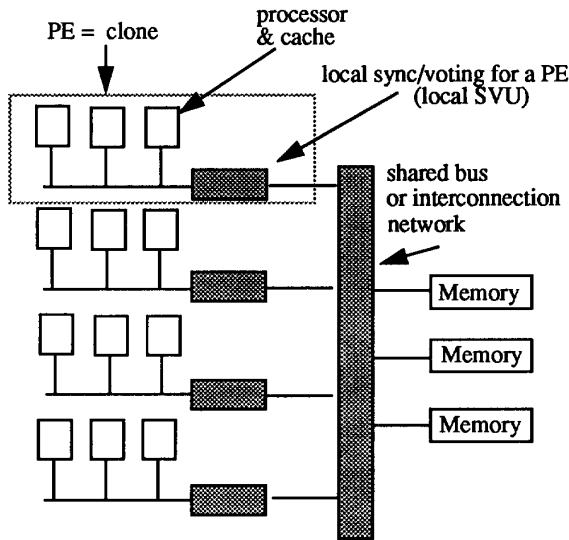


Figure 1: Fault-tolerant parallel processing architectures.

rollback methods, the scheme proposed in this paper uses redundant and synchronized cache-processor modules and, a *broadcasting* algorithm to realize transient restorations of erroneous cache data for shared-bus multiprocessor systems. In the following, we first present the parallel processing model and the recovery scheme, then prove the scheme.

2 Fault-tolerant parallel processing model

The model for fault-tolerant parallel processing is shown in Figure 1 in which the system has several PEs for parallel processing. A PE may consist of several processors for fault-tolerant purpose. The local synchronization and voting unit (SVU) is used for the fault detection and recovery of the caches for processors in a PE. The shared bus is assumed to have its own fault-tolerant mechanism for the main memory modules.

In this model, an application is parallelized among the PEs, and the task assigned to a PE is replicated by the number of processors that constitute the PE. A PE is also referred to as a clone. Figure 2 shows a PE organization. A processor needs to execute from its local ROM and RAM for initialization and self-test before synchronizing with other processors.

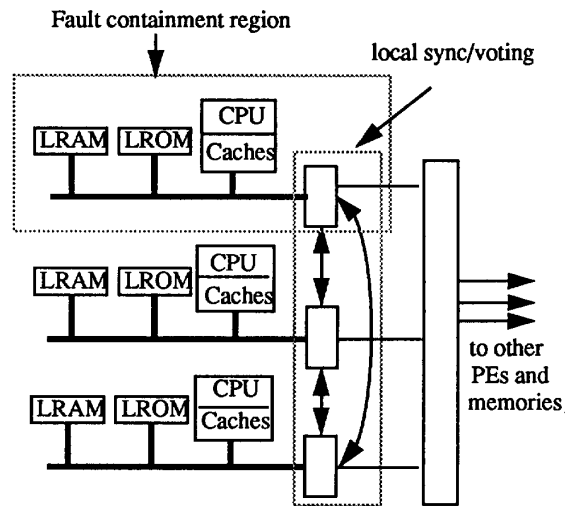


Figure 2: PE organization.

2.1 Local cache line error recovery

We propose a *read-first-dirty-broadcasting* algorithm to recover and detect the local cache line error in a PE. Figure 3 shows the algorithm. The algorithm requires the cache controller to broadcast a dirty line to the sync/voting unit when a dirty line is read for the first time. It is observed that, to reduce possible frequent broadcasts resulting from a sequence of consecutive reads and/or writes on a cache line, only the first read on a dirty line is broadcasted. We will show later that the read-first-dirty-broadcasting algorithm can prevent error propagation and recover a contaminated cache in a finite period of time.

2.2 Fault-tolerant coherence protocol

The read-first-dirty-broadcasting algorithm can be easily integrated with existing cache coherence protocols [9]. In this section, we use a write-update protocol as an example. The state transition of the protocol used by the Firefly multiprocessor system is shown in Figure 4 [9]. In the write-update policy, the cache coherence is maintained by broadcasting the shared cache line to all other caches, which then update their own copies. The data migration due to the coherent activities among caches may propagate the errors originating in one of the cache memories. To prevent the error propagation, a cache line that is broadcasted or requested by other caches needs to be verified.

```

If read hit on a non-verified dirty line for the first time
then
  each cache controller broadcasts the dirty line
  local sync/voting unit votes on the broadcasted lines
  if error detected
  then
    the erroneous line is replaced by the voted output
    voted output is given to all processors in a clone
  else /* no error is detected */
    each processor proceeds using its own cache line
else if write miss or flushing
then
  local sync/voting votes on the outputs of the processors
  if error detected
  then
    voted output is written to the memory

```

Figure 3: The read-first-dirty-broadcasting algorithm for write-back cache.

The fault-tolerant write-update protocol integrated with the *read-first-dirty-broadcasting* algorithm is shown in Figure 5. The **non-switched LP-read** is the read requests made on the same cache line before switching to any other cache lines by a local processor or a processor within a PE. There might be processor's writes in the same line for a **non-switched LP-read** sequence. The **switched LP-read** is the first read on a cache line after the processor switches its read or write requests from any other cache lines. The labels with **LP-read**, or **LP-write** make no distinction of whether switching the lines or not.

A verified state occurs when (1) the cache line is brought into cache from the main memory or from other caches, (2) the cache line is broadcasted to the local sync/voting unit by a **LP-switched read**. Specifically, the cache controller takes the following actions with respect to processor reads or writes for the fault-tolerant write-update protocol:

- write-hit: If the line is dirty or valid-exclusive,¹ then the write is performed, and the resultant state is dirty & non-verified. Referring to Figure 5, this corresponds to transition from state A to B, or C to B, or B to B.

If the line is in a shared state, and there are other shared copies, the written line is broadcasted to

¹When there is no mention of verified or non-verified, the context applies to both cases.

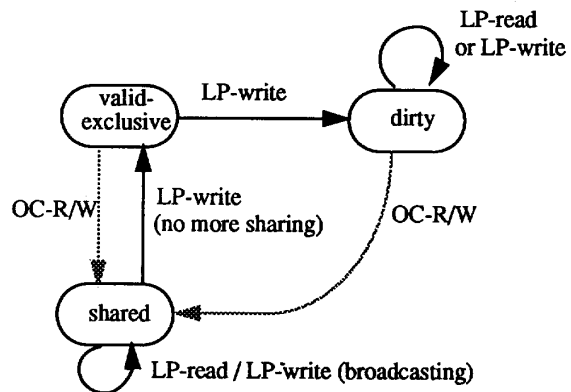


Figure 4: The state transition of a write-update coherence protocol. Dashed lines indicate coherence actions initiated by the read or write miss update of other caches (OC-R/W). Solid lines indicate actions initiated by local processor (LP).

the sync/voting unit which performs the detection and necessary recovery. Locally, any erroneous copy in the clone is recovered with the voted result. The local state of the cache lines in the clone becomes shared & verified. The state transition is from state E to F, or from F to F.

All other caches use the voted copy to update their corresponding copies, and the resultant states are shared & verified. If there are no other shared copies, but the current line state is shared & non-verified, then next state is valid-exclusive & verified, and no broadcasting takes place. The state transits, for this case, from E to A, or F to A.

- write-miss: If any other caches has the requested copy², the copy is read in first (detection can be inhibited). After the copy is read in, the write is performed, and the shared memory and other caches that have the old copy are updated with the new data by the voted result.

The requesting cache and other caches with a copy of that line set their states to shared & verified. The state transition for the supplying cache

²The states in the other caches for this copy could be shared, or dirty, or valid-exclusive.

and caches with the shared copy could be from A to F, B to F, C to F, or D to F.

If there is no such a copy in other caches, the requested copy is supplied by the memory, and the line state becomes dirty & verified.

- read-miss: If there are shared copies, or dirty copy, or valid-exclusive copy existent in the caches of a clone, the missing line is supplied by the voting result of the caches in the clone, and next line state is shared & verified.

Otherwise the shared memory supplies the copy, and the new state is valid-exclusive & verified.

Note that for all of the above cases, if any erroneous cache line belonging to a clone that supplies a requested copy will always be detected and recovered.

- read-hit: If the line is dirty & non-verified, perform the *read-first-dirty-broadcasting* algorithm for local error recovery. The state transition in this case is from B to D. In the original cache coherence protocol, read-hits require no cache coherent actions.

In our proposed scheme, for any cache line when requested by other caches, the detection and recovery are always carried out due to the redundant hardware. So local errors are not propagated to the requesting cache. However, if an error occurs in a non-verified state (dirty or valid-exclusive) of a cache line, subsequent reads (any non-switched LP-reads in state B, or C) of an erroneous line might generate write-hits on the other cache lines. The error may come from a transient fault of the processor, or the cache itself. This situation may potentially cause the propagation of the error in the cache.

In the following, we show that, if this does occur, a contaminated cache can be totally recovered by the *read-first-dirty-broadcasting* scheme in a finite period of time. First, some definitions are given to develop an error contamination model for a write-back cache.

Definition 1 Let cache line L become erroneous at time t . Then the set of lines which are contaminated due to the subsequent use of erroneous data in line L is denoted by C_L . Notice that line L can be in non-verified dirty or non-verified clean state.

Definition 2 $RD(L)$ is the sequence representing the order of lines read from C_L for the first time. $RD(L)_i$ is the i^{th} element of $RD(L)$.

It is noted that subsequently consecutive reads on the same dirty line are not part of the $RD(L)$.

Theorem 1 The *read-first-dirty-broadcasting algorithm (RFDB)* prevents further propagation of errors in C_L to other cache lines, if the following conditions are true: (1) for reading line $RD(L)_i$, where $i \geq 1$, the erroneous cache line is recovered during the majority voting phase, and (2) the redundant processors remain synchronized, i.e., execute the same sequence of code.

PROOF: With the *RFDB*, since all lines in C_L are dirty, a read reference to any of these lines results in the broadcasting of the line and, therefore, the erroneous line is recovered by the majority voting. Thus, the *RFDB* algorithm prevents further error propagation. \square

Definition 3 Between reading cache lines $RD(L)_i$ and $RD(L)_{i+1}$, a set of dirty lines which are in C_L , flushed back to the main memory due to read miss or write miss is denoted by $CRW(L)_i$. All such lines flushed before $RD(L)_1$ is read are denoted by $CRW(L)_0$.

Definition 4 The set of the dirty lines which belong to C_L , transits into shared states between reading cache lines $RD(L)_i$ and $RD(L)_{i+1}$ is denoted by $DS(L)_i$. All such lines changing to state shared before $RD(L)_1$ is read are denoted by $DS(L)_0$.

When a read miss or write miss occurs in other caches, a dirty line that is the requested copy is sent to the requester and the resultant state of that line becomes shared.

Theorem 2 The *fault-tolerant write-update protocol* recovers the contaminated cache lines within finite time constrained by $(i + \sum_{j=0}^{i-1} |CRW(L)_j| + \sum_{k=0}^{i-1} |DS(L)_k|) \geq |C_L|$.

PROOF: After the line $RD(L)_i$ is read, there are i lines recovered by the *RFDB* algorithm. At that time, $\sum_{j=0}^{i-1} |CRW(L)_j|$ lines are recovered by the cache flushing and $\sum_{k=0}^{i-1} |DS(L)_k|$ lines are recovered in the cache to cache transferring activity. Therefore, when $(i + \sum_{j=0}^{i-1} |CRW(L)_j| + \sum_{k=0}^{i-1} |DS(L)_k|) \geq |C_L|$, the error contaminated cache is recovered. It is essential to know that those lines in C_L do not generate second order of error pollutions as proved in Theorem 1. \square

For other cache coherence protocols such as the write-once policy, we can apply the *RFDB* scheme in a similar way.

3 Conclusions

This paper presents a unique error detection and recovery mechanism for fault-tolerant computing systems. The proposed real-time error detection and recovery mechanism which allows the processor, or cache memory, or both to be liable for transient faults remedies the insufficiency of error-correcting code. The proposed scheme maintains cache data consistency in redundant copies, ceases possible error propagation in the cache memory, and restores an error-polluted cache memory.

We show that this new error recovery scheme can be easily integrated with existing cache coherence protocols. Our scheme best fits in the systems where each node has several processing elements which can be dynamically configured into either redundant module or a shared-bus multiprocessor as suggested in [5]. The next step of our work is to study the performance impact imposed by the broadcasting overhead.

References

- [1] P. A. Bernstein, "Sequoia: A Fault-Tolerant Tightly Coupled Multiprocessor for Transaction Processing," *IEEE Computer*, Vol. 21, pp. 37-45, February 1988.
- [2] A. L. Hopkins, Jr., T. B. Smith, and J. H. Lala, "FTMP-A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft," *Proceedings of the IEEE*, Vol. 66, No. 10, pp. 1221-1239, Oct. 1978.
- [3] C. Babikyan, "The Fault Tolerant Parallel Processor Operating System Concepts and Performance Measurement Overview," *IEEE/AIAA/NASA 9th Digital Avionics Systems Conference*, pp. 366-371, 1990.
- [4] E. S. Harrison and E. J. Schmitt, "The Structure of System/88, a Fault-Tolerant Computer," *IBM System Journal*, Vol. 26, No.3, 1987.
- [5] S. Hariri, A. Choudhary, and B. Sarikaya, "Architectural Support for Designing Fault-Tolerant Open Distributed Systems," *IEEE Computer*, June 1992.
- [6] C. L. Chen and M. Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review," *IBM J. Res. Develop.*, Vol. 28, No. 2, March 1984.
- [7] D. B. Hunt and P. N. Marinos, "A General Purpose Cache-Aided Rollback Error Recovery (CARER) Technique," *Proceeding 17th Symposium Fault-Tolerant Computing*, pp. 170-175, 1987.
- [8] K. Wu, W. K. Fuchs, and J. H. Patel, "Error Recovery in Shared Memory Multiprocessors Using Private Caches," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, pp. 231-240, April 1990.
- [9] P. Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessors," *IEEE Computer*, June 1990.

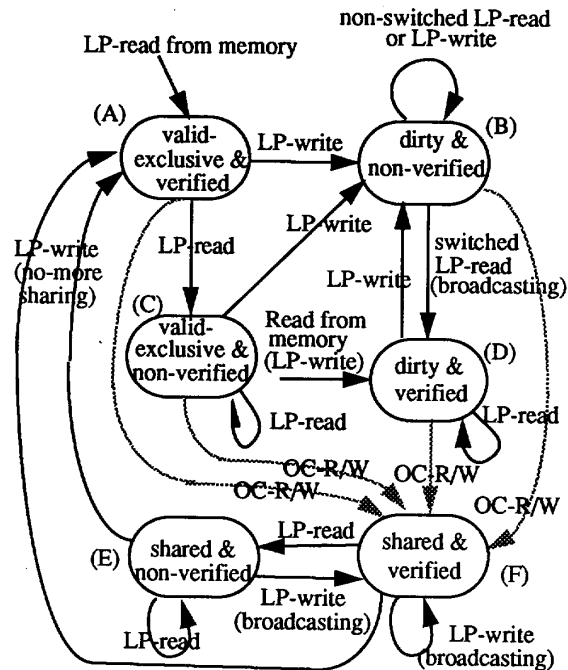


Figure 5: Fault-tolerant write-update protocol. Dashed lines indicate coherence actions initiated by the read or write miss update of other caches (OC-R/W). Solid lines indicate actions initiated by local processor (LP).

Acknowledgment

This research is supported in part by the Washington Technology Center.