

# The Performance of Cache-Based Error Recovery in Multiprocessors

Bob Janssens and W. Kent Fuchs, *Senior Member, IEEE*

**Abstract**—Several variations of cache-based checkpointing for rollback error recovery from transient errors in shared-memory multiprocessors have been recently developed. By modifying the cache replacement policy, these techniques use the inherent redundancy in the memory hierarchy to periodically checkpoint the computation state. Three schemes, different in the manner in which they avoid rollback propagation, are evaluated in this paper. By simulation with address traces from parallel applications running on an Encore Multimax shared-memory multiprocessor, we evaluate the performance effect of integrating the recovery schemes in the cache coherence protocol. Our results indicate that the cache-based schemes can provide checkpointing capability with low performance overhead, but with uncontrollable high variability in the checkpoint interval.

**Index Terms**—Fault-tolerant computing, cache-based checkpointing and rollback recovery, shared-memory multiprocessors, trace-driven simulation

## I. INTRODUCTION

IN fault-tolerant computing systems, checkpointing and rollback are often used to allow recovery from detected errors without a global restart of computation. Even under adverse conditions, most of the computation cycles in a typical system are error-free. It is therefore important to minimize the overhead of the error recovery mechanisms. Cache-based checkpointing is a hardware approach to user-transparent checkpointing from transient processor errors [13]. It uses the inherent redundancy in the memory hierarchy for maintaining checkpoints. The technique has been extended to shared-memory multiprocessors by integrating the checkpointing and recovery algorithms with cache coherence protocols [1], [26].

This paper presents an analysis of the impact on system performance for several variations of cache-based error recovery in bus-based multiprocessors. Simulation with address traces from an Encore Multimax shared-memory parallel processor is employed to gather statistics on the performance of the memory hierarchy in multiprocessors with several variations of cache-based error recovery. These statistics are then used to derive the overall system performance impact.

Manuscript received October 12, 1992; revised September 17, 1993. This work was supported in part by the National Aeronautics and Space Administration (NASA) under Grant NASA NAG 1-613, in cooperation with the Illinois Computer Laboratory for Aerospace Systems and Software (ICLASS), and in part by the Office of Naval Research under Contract N00014-91-J-1283. An initial version of this paper was presented at the 1991 International Conference on Parallel Processing [14].

The authors are with the Center for Reliable and High-Performance Computing, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA; e-mail: blj@crhc.uiuc.edu.

IEEE Log Number 9403100.

Several techniques have been proposed to provide rollback recovery in general purpose architectures. Some commercial machines have implemented the capability to retry an instruction if an error is detected during execution [11], [23]. Tamir and Tremblay proposed micro-rollback, a generalization of instruction retry that supports rolling back execution a number of clock cycles by delaying writes to the register file and cache and backing up individual state registers [25]. Research has shown that some of the overhead of instruction retry can be reduced by using compiler techniques [2], [18]. Instruction retry schemes have the disadvantages that processors can roll back only a few cycles, and that modifications have to be made in the processor design.

Bowen and Pradhan have developed a scheme that supports checkpointing and recovery at a higher level, in the virtual memory translation hardware [10]. Before a memory page is modified, a backup copy is made. The overhead of this scheme due to an increase in page manipulations can be made small by making the checkpoint interval large. Trace-driven simulations have shown an overhead of less than 1% for a checkpoint interval of 1 000 000 memory references [10]. The scheme requires modification of the microprocessor TLB.

The use of a cache to save the checkpoint state was first proposed by Lee *et al.* [17]. Their recovery cache simply stores all data that are part of the checkpoint state and that have been overwritten in memory. Every write to a memory location must be preceded by a read to that location to maintain the data in the recovery cache. The original cache-aided rollback error recovery (CARER) scheme for uniprocessors, proposed by Hunt and Marinos [13], does not require an extra cache for checkpointing and adds no extra read cycles. It modifies the replacement policy of the regular cache to prevent replacement of dirty data, thereby keeping a checkpoint state in memory. When dirty data lines need to be replaced, a checkpoint has to be taken.

In shared-memory parallel processors, a recovery scheme needs to be able to limit rollback propagation. The Sequoia multiprocessor's cache-based recovery scheme accomplishes this by flushing the cache on every processor interaction [9]. To avoid these costly flushes, Wu *et al.* extended the CARER scheme by always checkpointing the source processor immediately after it communicated with another processor [26]. Ahmed *et al.* presented two alternative algorithms for multiprocessors [1]. In one of the schemes, checkpointing is fully synchronized to occur at the same time on all processors. In a modification of this simple scheme, a checkpoint on one processor induces concurrent checkpoints on all processors

that have had any communication activity since the previous checkpoint. Algudady *et al.* have also extended Wu's scheme to multistage interconnection networks [3]. An alternative approach for bus-based shared-memory multiprocessors, proposed by Banâtre *et al.*, centralizes dependency tracking at the memory element [4], [5].

Hunt and Marinos evaluated their uniprocessor CARER technique by trace-driven simulation and found a modest performance overhead for small caches [13]. Until recently, the multiprocessor cache-based recovery methods had been analyzed only by using estimates of the probabilities of various events. Wu *et al.* used analytical modeling to show that for certain workload and system parameters, their scheme results in slight performance degradation, especially when a special buffer is used to store data lines needed only for checkpointing. Using the same approach, Ahmed *et al.* arrived at similar results for their recovery schemes. Algudady *et al.* used a queuing model to predict an overhead of 2% to 12%, depending on the degree of sharing. The performance predicted with each of these models depends highly on the estimates of the probabilities of various cache events. In our work, we actually measure the probabilities of the cache events by trace-driven simulation and use these probabilities to derive a more accurate and realistic measure of the performance impact of multiprocessor cache-based error recovery. As part of the evaluation of their checkpointing approach, Banâtre *et al.* independently simulated the communication-induced scheme [5]. In this paper, we evaluate the performance of Wu's communication-induced scheme, and Ahmed's fully synchronized and flagged synchronized schemes.

## II. CACHE-BASED CHECKPOINTING IN MULTIPROCESSORS

### A. Description of Algorithms

In the CARER cache-based checkpointing proposal [13], a checkpoint is taken every time a dirty cache line is written back. The replacement policy for a set associative cache is modified so that a checkpoint is avoided by selecting clean lines for replacement whenever possible. At a checkpoint, the internal processor registers are saved, and all dirty lines in the cache are marked *unwritable*. Unwritable lines may be read, but have to be written back to memory before they are modified. When a rollback is necessary, all cache lines, except the unwritable lines, are invalidated, and the saved processor registers are restored, thereby restarting the computation at the last checkpoint. Because the checkpoint state is stored partially in the cache and partially in main memory, both should be highly reliable or protected by their own error detection and recovery mechanisms.

Multiprocessor systems present the problem of rollback propagation. A rollback of one processor may necessitate a rollback of other processors that have communicated with it. These extra rollbacks may in turn cause further rollbacks, resulting in a domino effect that may extend completely back to the beginning of execution [22]. In a shared-memory multiprocessor, communication occurs when a processor reads a variable that has been written by another processor. If

cache-based checkpointing is used, only data that have been checkpointed are written back to main memory. Therefore, rollback propagation occurs only when data are moved directly between caches.

In a shared-memory multiprocessor that caches its synchronization variables, rollback propagation can be eliminated by checkpointing the source processor after every communication [26]. In this communication-induced scheme, the destination processor is allowed to roll back past an interaction, which will not affect correctness unless the program uses a synchronization variable to order memory accesses. In that case, the accesses to the synchronization variables in the cache automatically enforce the correct checkpointing.

Another way to avoid the domino effect is by limiting the amount of rollback propagation through fully synchronized checkpointing. A checkpoint on one processor immediately causes a checkpoint on all other processors. Similarly, a rollback on one processor causes all others to roll back to the last checkpoint.

An enhancement to this simple scheme, originally proposed by Ahmed *et al.*, is flagged synchronized checkpointing and rollback [1]. We modify their algorithm slightly by allowing the receiver of a message to roll back past the interaction. Two flags are required in every cache controller. The *R* flag is set when a cache has received data since the last checkpoint. The *S* flag is set when a cache has sent data since the last checkpoint. A checkpoint on one processor raises the *establish checkpoint* bus line if its *R* flag is raised. Processors with *S* flags raised take a checkpoint upon sensing the *establish checkpoint* signal. Similarly, a rollback on a processor raises the *rollback* line if its *S* flag is set. Only processors with their *R* flags raised will actually take a checkpoint upon sensing the *rollback* signal.

The Berkeley write-invalidate coherence protocol [16] was used in this study. Fig. 1 shows the transition diagram for the Berkeley protocol. To integrate the checkpointing schemes into the protocol, an *unwritable (owned)* and a *shared unwritable* state were added to the four original states of *invalid*, *valid*, *dirty (owned)*, and *shared dirty*. In the communication-induced checkpointing scheme, a bus miss on a dirty line will always cause a checkpoint, forcing the state into *shared unwritable*. Therefore, the *shared dirty* state is not needed. Figs. 2 and 3 show the transition diagram for the modified Berkeley protocols.

### B. Performance Impact

There are several ways in which cache-based checkpointing affects the overall performance of a computer system. For set associative caches, the modification of the least recently used (LRU) replacement policy affects the miss ratio. However, the effect of the replacement policy on cache miss ratio is generally small [12]. Also, the addition of an *unwritable* state affects bus traffic. At any time, a certain percentage of the modified cache lines will be *unwritable* instead of *dirty*. On a write hit, an *unwritable* line has to be written back to main memory before it can be updated, causing a delay and extra bus traffic. In multiprocessors with a write-invalidate cache

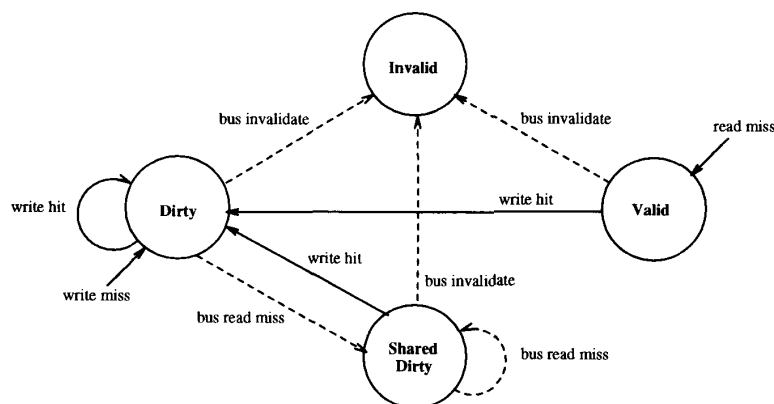


Fig. 1. The Berkeley cache coherence protocol.

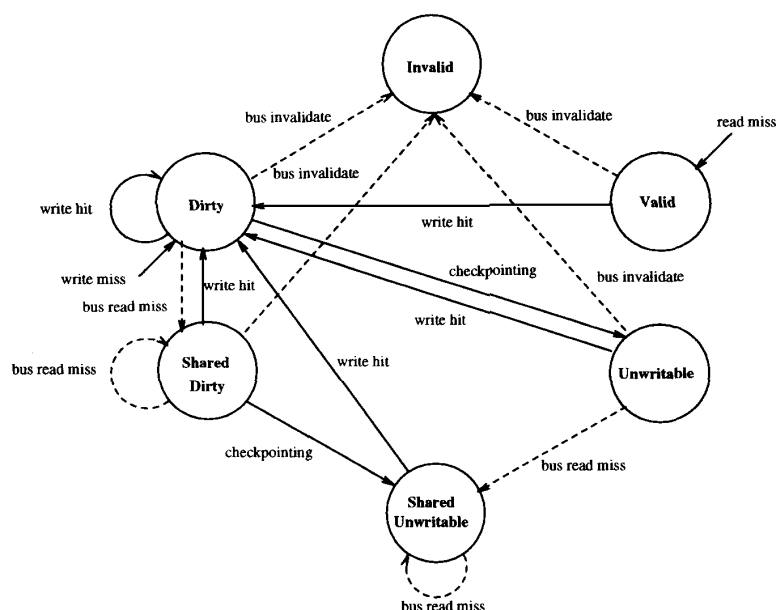


Fig. 2. Berkeley protocol modified for synchronized checkpointing schemes.

coherence protocol, the bus traffic is increased even further by the need to write back unwritable lines before they can be invalidated. In both cases, the data written back are checkpoint data; from the computation's point of view, they have been overwritten. To eliminate the extra bus traffic, these data can be stored in a recovery buffer local to every cache [26].

The most direct effect on the performance of recoverable systems is created by the time needed to take checkpoints. This overhead increases with both the length and frequency of checkpointing events. If the internal processor state can be saved in one cycle and checkpoint identifiers are used, the average time to take a checkpoint can be made less than two cycles [26]. However, the time to take a checkpoint may need to be increased to allow for error latency. Because the cache-based schemes can take checkpoints very quickly, relatively high checkpointing frequencies can be tolerated.

The checkpointing frequency depends on the behavior of the program and the caches. Parts of the program that have consecutive writes to the same cache line will checkpoint frequently, to enable writeback of the dirty data. Increasing cache associativity increases the probability of finding a clean line to replace, thereby decreasing the number of necessary checkpoints.

In the communication-induced checkpointing schemes, program sections with high intercache communication, such as areas of contention for synchronization variables, will also have a high checkpointing frequency. The flagged synchronized scheme is much less sensitive to communication, because a checkpoint is only taken when a dirty cache line needs to be replaced. The fully synchronized scheme always induces checkpoints in all processors, so its performance is independent of the amount of communication, but is always

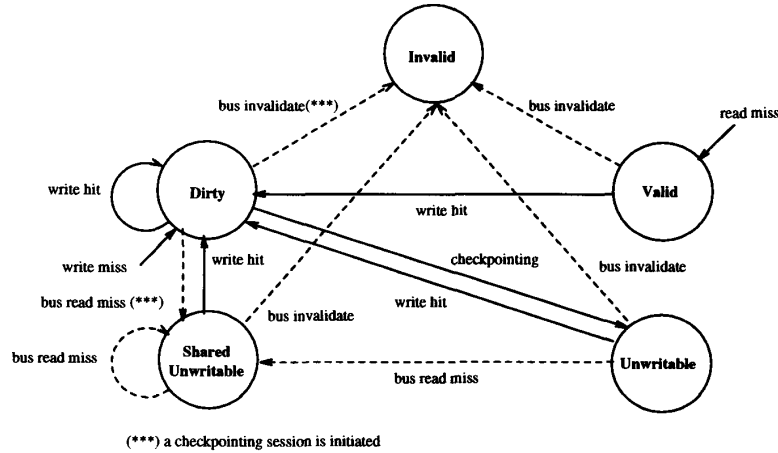


Fig. 3. Berkeley protocol modified for communication-induced checkpointing scheme.

TABLE I  
ADDRESS TRACES USED IN THE SIMULATIONS

| program | description       | static code size<br>(instructions) | input                                    | num.<br>of cpus |
|---------|-------------------|------------------------------------|--|-----------------|
| gravsim | N-body simulator  | 6,985                              | two colliding 128-body clusters          | 7               |
| tgen    | test generator    | 16,179                             | ISCAS benchmark s208 w. 20 faults        | 7               |
| fsim    | fault simulator   | 13,451                             | ISCAS benchmark s208 w. 80 input vectors | 7               |
| pace    | circuit extractor | 16,101                             | two bit slices of an ALU                 | 7               |
| phigure | global router     | 19,046                             | 469 cell circuit                         | 7               |

worse than that of the flagged scheme. Because checkpoints can occur in bursts, there may be sections of programs where performance is significantly degraded. Therefore, the schemes may not be suitable if the program needs to meet real-time constraints.

### III. TRACE-DRIVEN SIMULATION EXPERIMENTS

#### A. Methodology and Workload

The address traces used in this study were generated by a version of the TRAPEDS address trace generator on the Encore Multimax shared-memory multiprocessor [24]. The interleaving of the processor traces in our experiments was determined by using the microsecond timer available on the Multimax. The traces were saved to disk and later fed into a multiprocessor recovery cache simulation.

The traces used were from five parallel C programs, written using Encore's standard parallel library, running on an eight-processor Multimax 510. The applications and inputs used to generate the traces are described in Table I. Gravsim is a gravitational *N*-body simulator using the Barnes and Hut algorithm [8]. Tgen is a test pattern generator for sequential and combinational circuits using a parallel search algorithm [21]. Fsim is a parallel fault simulator for digital circuits [21]. Pace is a parallel circuit extractor [6], [7]. Phigure is global router using structured hierarchical decomposition of independent tasks [7]. The amount of synchronization in each program varies. At one extreme, gravsim synchronizes

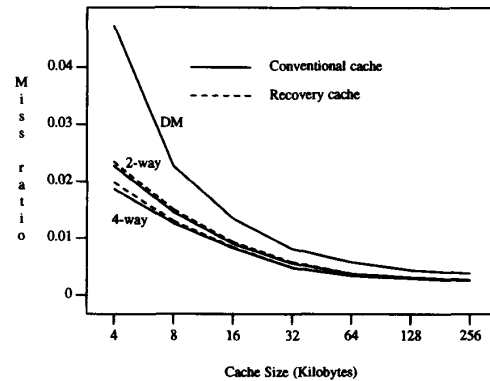


Fig. 4. Miss ratios for direct-mapped, two-way set associative, and four-way set-associative caches.

repeatedly at barriers and has multiple critical sections. On the other hand, phigure does not use any barrier synchronization and has only one critical section.

All traces were generated on seven processors. The eighth processor was used to monitor the progress and disk usage of the trace. Table II describes the characteristics of the generated traces. The total number of memory references is separated into data reads, data writes, and code reads. All memory reference numbers are the total references from all seven processors. All applications were traced from beginning to end, resulting in traces of more than 10 000 000 memory references per processor in every case.

TABLE II  
CHARACTERISTICS OF THE ADDRESS TRACES

| program  | tot. num. of references | data reads |            | data writes |           | code reads |
|----------|-------------------------|------------|------------|-------------|-----------|------------|
|          |                         | total      | shared     | total       | shared    |            |
| gravesim | 92,178,814              | 33,266,880 | 12,484,455 | 6,392,078   | 251,694   | 52,519,859 |
| tgen     | 101,264,382             | 32,613,809 | 16,550,450 | 4,461,889   | 642,796   | 64,188,674 |
| fsim     | 149,918,375             | 50,950,933 | 39,326,911 | 3,958,919   | 999,127   | 95,008,523 |
| pace     | 87,861,165              | 23,266,576 | 1,286,787  | 7,842,338   | 348,524   | 56,752,251 |
| phigure  | 132,998,231             | 38,244,233 | 4,281,207  | 11,530,981  | 1,876,400 | 83,223,027 |

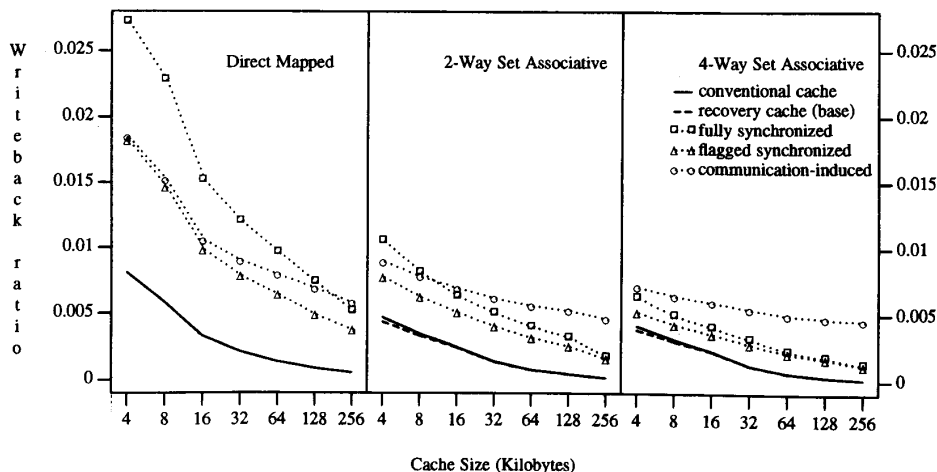


Fig. 5. Writeback traffic.

### B. Experimental Results

Simulations were performed using the Berkeley cache coherence protocol for the fully synchronized, flagged synchronized, and communication-induced checkpointing schemes. The versions of the flagged synchronized and communication-induced schemes simulated allow the receiver to roll back past an interaction. Cache sizes were varied from 4 kilobytes to 256 kilobytes, whereas line size was fixed at 16 bytes. Direct-mapped, two-way set associative, and four-way set associative caches were simulated. Cache and checkpointing statistics were calculated on a per-processor basis and then averaged to arrive at a single number for each application.

As expected, the modified replacement policy used in the cache-based error recovery schemes has only a slight effect on the performance. The average miss ratios for the five programs are shown in Fig. 4. The fraction of references that cause writebacks, the writeback ratio, is shown in Fig. 5. In the direct-mapped cache, the miss ratio and the base writeback ratio (traffic resulting from writebacks needed for the computation) remain the same. In the set associative caches, they vary by an insignificant amount with the modified replacement policy of the recovery schemes. The total number of writebacks consists of those needed for checkpointing, in addition to the base writebacks. The frequency of these additional writebacks depends heavily on the frequency of checkpointing and is therefore discussed after introducing the checkpointing frequency data.

Fig. 6 presents, on a logarithmic scale, the average of the checkpointing frequencies for all the programs except

phigure. The base checkpointing frequency includes all checkpoints that are taken because of the necessity of writing back a dirty cache line. The total checkpointing frequency also includes the checkpoints induced in other processors. The base checkpointing frequency decreases approximately linearly with increasing cache size; it decreases significantly with increasing associativity. Fig. 7 presents the checkpointing frequency for the phigure program, in which a large percentage of the data accesses are write accesses (see Table II). The base checkpointing frequency for phigure is about an order of magnitude higher than it is for the other programs. In the two-way set associative cache, the frequency decreases less than expected with cache size until it drops dramatically at 256 kilobytes. This behavior is probably due to a few variables in the same cache set that are frequently modified, and therefore need to be written back upon replacement, causing a checkpoint.

The extra induced checkpointing frequency varies with the recovery scheme used. In the fully synchronized scheme, a checkpoint on one processor causes simultaneous checkpoints on all other processors. Therefore, the total checkpointing frequency remains at approximately seven times the base frequency. The large number of induced checkpoints actually reduces the number of base checkpoints by clearing the cache frequently of dirty lines. The flagged synchronized scheme drastically reduces the induced checkpointing frequency, because checkpoints are induced only if communication has occurred with the originating processor. Even in programs with large amounts of communication, the flagged synchronized

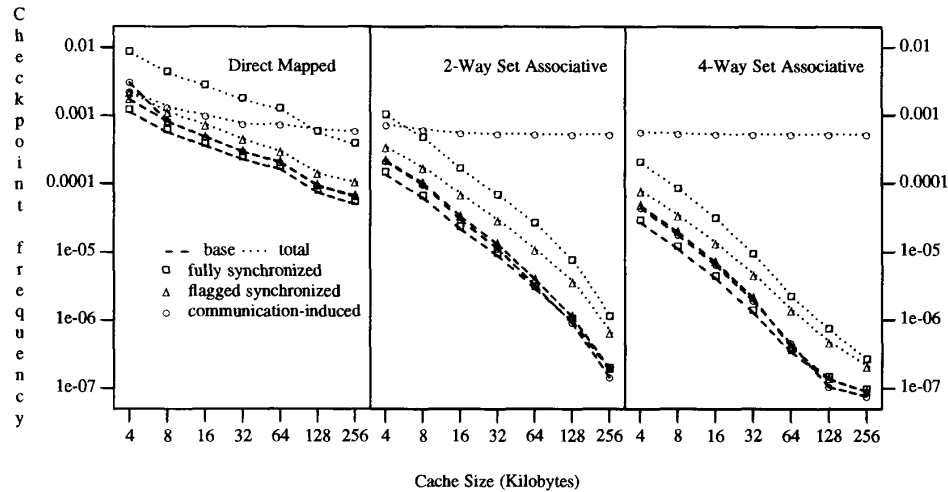


Fig. 6. Average checkpointing frequency for all programs except phigure.

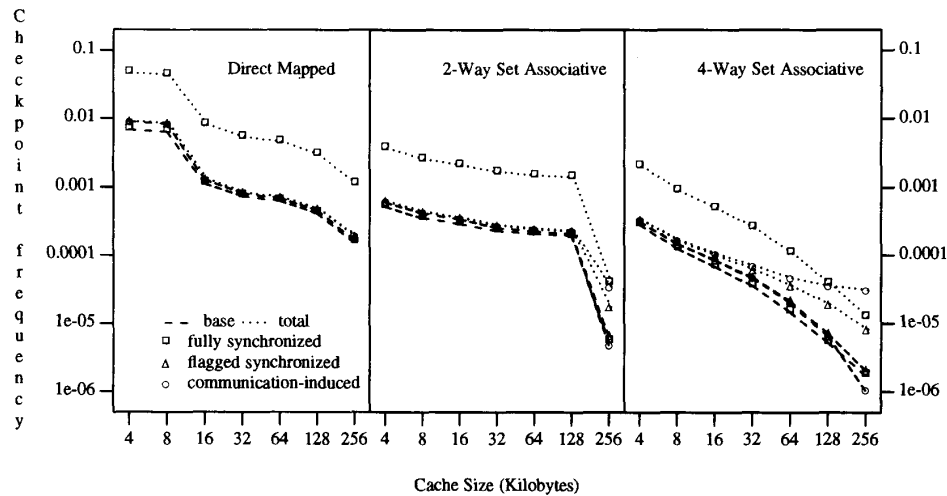


Fig. 7. Checkpointing frequency for phigure.

scheme performs much better than the fully synchronized scheme. In programs with little communication, the increase over the base checkpointing frequency is negligible. The number of extra checkpoints in the communication-induced scheme is fixed, regardless of cache size. On average, it performs well compared to the synchronized schemes only for small cache sizes. In the case of phigure, which has little communication, there are very few induced checkpoints, but the performance of the communication-induced scheme is still inferior to that of the synchronized schemes in large set associative caches.

In all three schemes, even when the average checkpointing frequency is low, there is no guaranteed minimum checkpoint interval. Fig. 8 plots the average length of the checkpoint interval, and the length of the largest and smallest intervals. In all cases, the smallest interval is of length 0, whereas the

largest intervals are in the tens of millions of instructions. The presence of a small amount of large intervals increases the overall average checkpoint interval. Even in the largest caches, the majority of checkpoint intervals are still small. To ensure fast recovery, it might be necessary to limit the maximum checkpoint interval. Unfortunately, the checkpointing schemes do not allow control over the minimum checkpoint interval. Therefore, the checkpoint frequency will always be variable.

The total writeback traffic is influenced by the checkpoint frequency. The chance of an extra writeback resulting from the modification or invalidation of an unwritable cache line is determined by the percentage of unwritable lines present in the cache. The more frequently checkpointing occurs, the more likely it is for a modified block to be in the unwritable rather than dirty state. A comparison of Figs. 5 and 6 shows that the number of extra writebacks indeed increases with increased

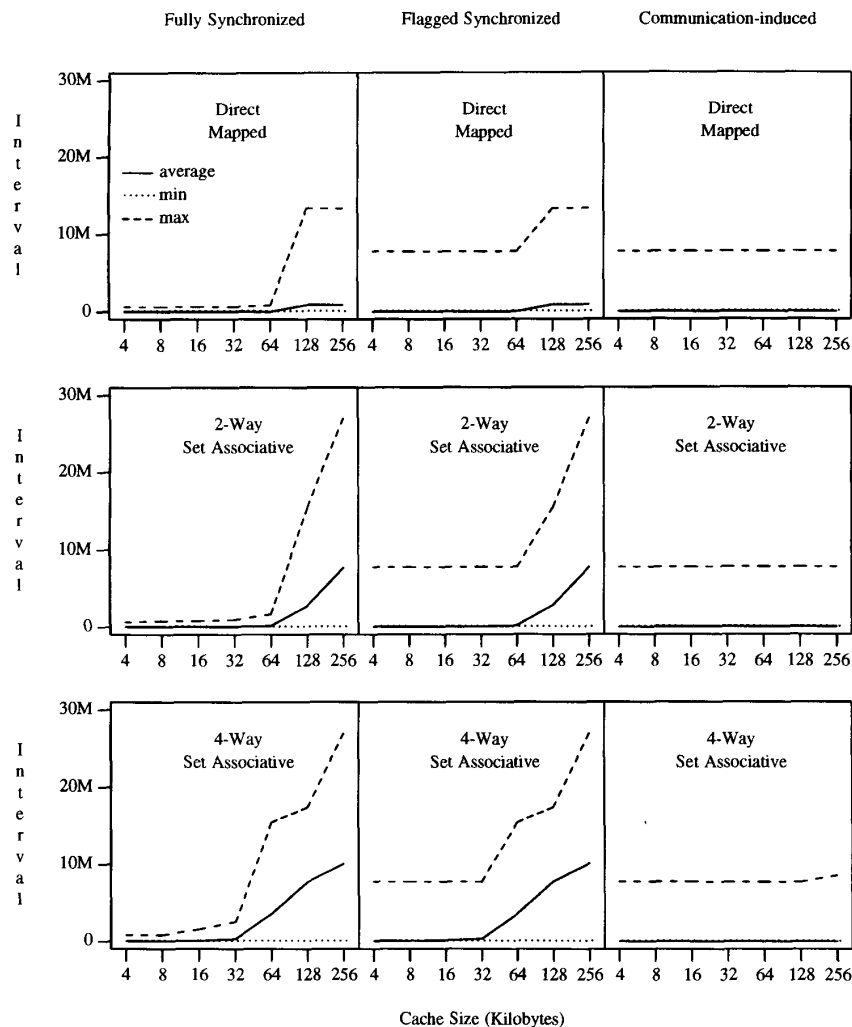


Fig. 8. Average, minimum, and maximum length of checkpoint interval.

checkpoint frequency. The extra writeback frequency of the synchronized schemes shows variability with the cache size, whereas that of the communication-induced scheme does not, just as in the case of checkpoint frequency. The extra writebacks can easily double the total traffic. In large caches, almost all of the traffic is due to extra writebacks.

Extra writebacks to main memory can be eliminated, at the cost of extra silicon, by storing the replaced unwritable lines in a buffer local to the cache. The buffer is cleared at every checkpoint. If it fills up before a checkpoint is taken, the writebacks proceed to the main memory. Fig. 9 shows the percentage of extra writebacks that can be stored in a buffer with one-sixteenth the capacity of the cache. Because the checkpoint frequency is very high in direct-mapped caches, nearly 100% of all writebacks can be stored in the recovery buffer. When checkpointing is less frequent, the recovery buffer fills up more often. In all cache organizations, however, the recovery buffer catches at least 70% of the extra writebacks in the traces simulated.

#### IV. SYSTEM PERFORMANCE

##### A. Performance Model

The cache performance and checkpointing frequency figures generated from our address traces interact to affect overall system performance. Their impact will vary with the specific architecture of the multiprocessor system involved. However, it is instructive to determine the performance impact of supporting cache-based error recovery in a typical multiprocessor. The performance metrics generated by the cache simulation were used in a simple processor and bus behavior model similar to that developed by Patel [19], [20]. Other research has used a similar model with estimated cache behavior parameters [1], [26]. In this study, we get a more realistic view of cache performance by using the actual measurements from our trace-driven simulations.

The model consists of three equations, with three unknowns representing the actual execution time  $Z$  for one unit of useful

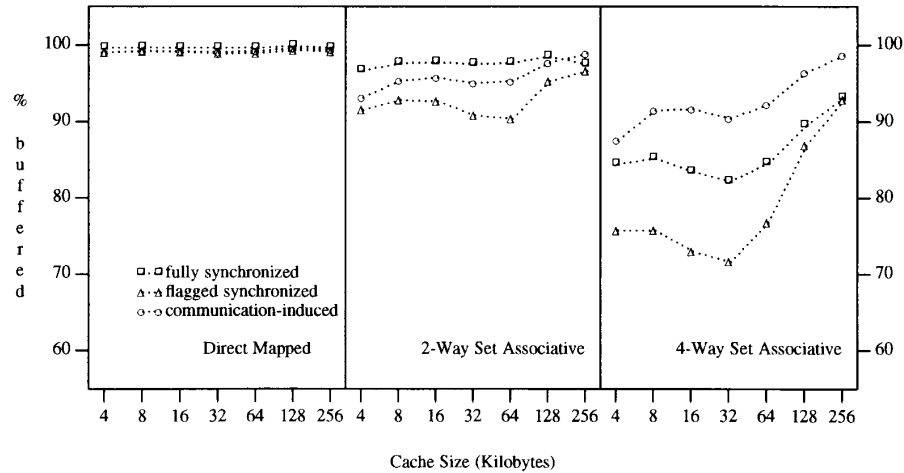


Fig. 9. Extra writebacks captured by a recovery buffer with one-sixteenth the capacity of the cache.

TABLE III  
PARAMETERS GENERATED FROM SIMULATION USING ADDRESS TRACES

| Parameter | Description  |
|-----------|--|
| $N$       | number of processors (7)   |
| $m$       | miss ratio   |
| $w_b$     | base writeback ratio   |
| $w_x$     | extra writeback ratio due to write hits to unwritable lines                                    |
| $f_b$     | fraction of extra writebacks that go to recovery buffer  |
| $i$       | invalidate ratio: fraction of references that cause an invalidation                            |
| $o$       | other cache ratio: fraction of references that cause data to be transferred from another cache |
| $c$       | checkpoint frequency   |

TABLE IV  
ASSUMED PARAMETERS

| Parameter | Value used | Description   |
|-----------|------------|---|
| $a$       | 0.70       | processor memory reference rate                               |
| $T$       | 4          | number of cycles to transfer a cache line                     |
| $I$       | 4          | number of cycles to invalidate a cache line                   |
| $A$       | 2          | number of cycles for bus arbitration                          |
| $R$       | 2          | number of cycles to store a cache line in the recovery buffer |
| $C$       | varied     | average number of cycles to take a checkpoint                 |

work, the bus utilization  $B$ , and the average waiting time  $W$  per bus request. Table III describes the parameters of the model that are generated directly from the trace-driven simulations. Other parameters depend on the specifics of the multiprocessor. Their description and assumed values are given in Table IV.

The performance model is described in the most general terms by assuming that the cache has recovery capability and a recovery buffer. If recovery is not implemented, the equations can be used with  $w_x = c = 0$ . If there is no recovery buffer,  $f_b = 0$ . The model uses the variable  $b$ , the average number of bus requests per cycle, calculated as follows:

$$b = ma + ia + (1 - f_b)w_x a,$$

where  $ma$  represent the bus requests due to cache misses,  $ia$  represents the bus requests due to invalidates of shared lines

in other caches, and  $(1 - f_b)w_x a$  represents the bus requests due to extra writebacks not caught by the recovery buffer.

The execution time  $Z$  for one unit of useful work is calculated as follows:

$$Z = 1 + bA + maT + [w + (1 - f_b)]aT + iaI + bW + f_b w_x aR + caC + Q. \quad (1)$$

It includes one cycle to do the actual work and overheads of  $bA$  for bus arbitration,  $maT$  for cache misses,  $[w + (1 - f_b)]aT$  for writebacks not caught by the recovery buffer,  $iaI$  for invalidations,  $bW$  for bus wait time,  $f_b w_x aR$  for writes to the recovery buffer,  $caC$  for taking checkpoints, and  $Q$  for cache interference. Cache interference occurs when memory requests collide with invalidation or transfer requests. It is



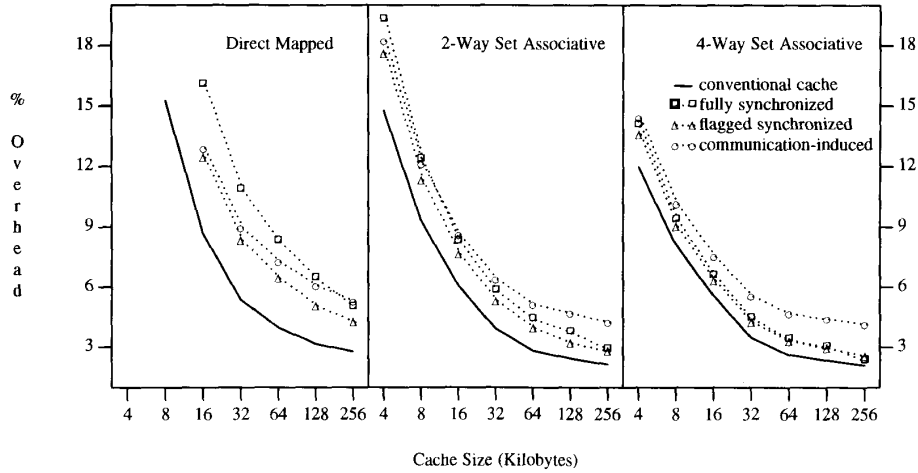


Fig. 10. Performance overhead without recovery buffer,  $C = 2$ .

calculated as follows:

$$Q = \frac{a(ia + oaT)}{1 + bA + bW}.$$

The average bus utilization  $B$  can be calculated two ways. The probability that no processor is requesting the bus is the unit request rate as seen from the bus subtracted from 1 and then raised to the power of the number of processors. The bus utilization  $B$  is in turn 1 minus this quantity, or the following:

$$B = 1 - \left(1 - \frac{z - 1 - bA - f_b w_x a R - caC - Q}{Z}\right)^N. \quad (2)$$

The bus utilization can also be derived by multiplying  $N$  by the actual bus time used by a processor, and averaged over the execution period, as shown below:

$$B = \frac{N(z - 1 - bA - f_b w_x a R - caC - bW - Q)}{Z}. \quad (3)$$

### B. Performance Results

To compare the performance of the varying recovery cache scheme with a conventional cache, we define the percentage overhead associated with the memory system as follows:

$$O = 100(1 - Z).$$

In a perfect system with no cache misses,  $O$  would equal zero; in a real cache-based system,  $O$  represents the percentage increase in execution time over such a perfect system.

Fig. 10 presents the overheads for the recovery schemes without a recovery buffer and with the average checkpoint penalty  $C = 2$ . Overheads of over 20% are not shown, in order to allow increasing the scale of the graphs. Even in small direct-mapped caches, the additional overhead of the checkpointing schemes is never higher than a few percentage points. The communication-induced scheme performs badly in caches that are large or have high associativity. The fully synchronized scheme performs badly in caches that are small or have low associativity. The flagged synchronized schemes performs the best in all cases.

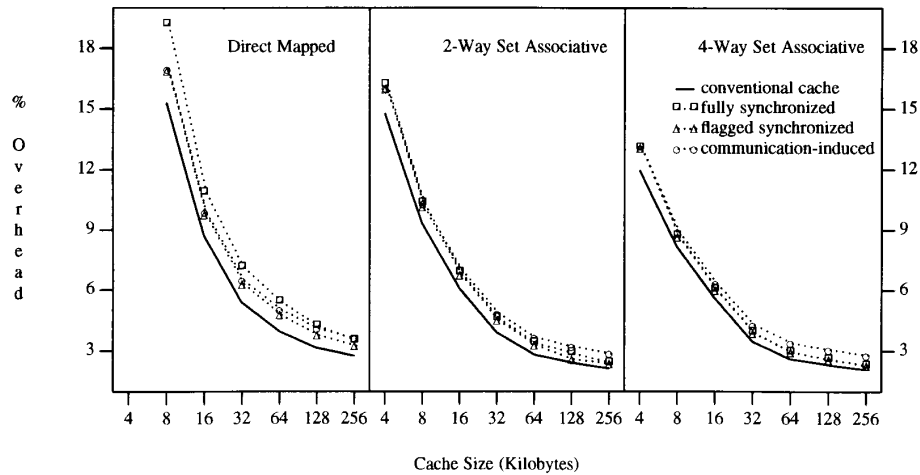
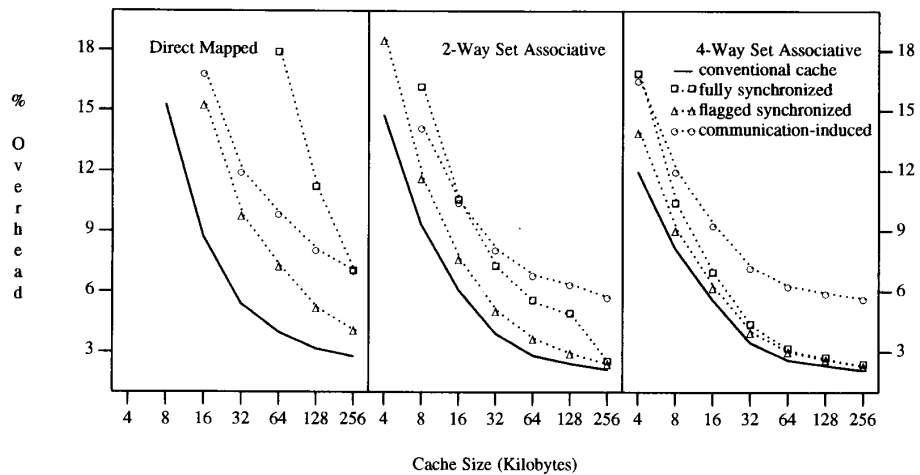
The addition of a recovery buffer greatly improves the performance, especially when checkpoint intervals are small. Fig. 11 presents the overhead for the recovery schemes with a recovery buffer with one-sixteenth the capacity of the cache. With the recovery buffer, all schemes increase the overhead by only a few percentage points from that of the conventional cache. Even though the flagged synchronized scheme still performs best, its comparative advantage is reduced.

An average checkpoint penalty of two cycles may not be realistic in some systems. It may be impossible to save the internal processor state in one cycle. It may also be necessary to insert a delay of a number of cycles before a checkpoint to allow for error latency. Fig. 12 presents the overheads with a recovery buffer when the average checkpoint penalty  $C$  is increased to 100. With the high checkpoint penalty, the overhead increases noticeably when checkpoint frequency is high. The communication-induced and fully synchronized schemes are affected the most by the higher checkpoint penalty, whereas the flagged synchronized scheme is affected by a much smaller amount. In the set associative caches, the additional overhead of flagged synchronized checkpointing over a conventional cache is still less than 1%.

### V. CONCLUSION

Trace-driven simulation was used to measure the performance of cache-based error recovery in a shared-memory multiprocessor. The results answer many questions that remained unresolved after previous approximate analytical performance studies of the cache-based recovery methods. The cache miss ratio is only slightly degraded by the addition of the recovery scheme to the cache protocol. The writeback traffic increases significantly. However, a recovery stack at every cache can intercept many of the extra writebacks caused by the recovery schemes.

The addition of cache-based recovery degrades performance by at most a few percentage points. The communication-induced scheme does not improve as cache size and associativity are increased. In the range of cache sizes used

Fig. 11. Performance overhead with recovery buffer,  $C' = 2$ .Fig. 12. Performance overhead with recovery buffer,  $C' = 100$ .

today, the performance of the communication-induced scheme compares unfavorably with that of the other schemes. However, our recent work has shown that the performance of the communication-induced scheme can be improved significantly by relaxing the memory consistency model [15]. The fully synchronized scheme works well only in large caches with high associativity. At the cost of extra cache hardware, the flagged, synchronized scheme outperforms both other schemes in all cases. Its performance degradation is lowest in highly associative caches. But even in a two-way set associative cache with a high checkpoint penalty, it degrades performance by 1% or less.

A major disadvantage of all the cache-based recovery schemes is the uncontrollability and variability of the checkpoint frequency. This instability manifests itself both between different programs and within the execution length of a program. The performance of a system with cache-based checkpointing will therefore be less predictable than that of a system without recovery capability, even though average

overhead is minimal. Future research should be directed at finding low-cost methods that eliminate this disadvantage.

#### REFERENCES

- [1] R. E. Ahmed, R. C. Frazier, and P. N. Marinos, "Cache-aided rollback error recovery (CARER) algorithms for shared-memory multiprocessor systems," in *Proc. 20th Int. Symp. Fault-Tolerant Computing*, 1990, pp. 82-88.
- [2] N. J. Alewine *et al.*, "Branch recovery with compiler-assisted multiple instruction retry," in *Proc. 22nd Int. Symp. on Fault-Tolerant Computing*, 1992, pp. 66-73.
- [3] M. S. Algudady, C. R. Das, and M. J. Thazhuthaveetil, "A cache-based checkpointing scheme for MIN-based multiprocessors," in *Proc. Int. Conf. Parallel Processing*, 1991, pp. 1-497-1-500.
- [4] M. Banâtre and P. Joubert, "Cache management in a tightly coupled fault tolerant multiprocessor," in *Proc. 20th Int. Symp. on Fault-Tolerant Computing*, 1990, pp. 89-96.
- [5] M. Banâtre *et al.*, "An architecture for tolerating processor failures in shared-memory multiprocessors," Tech. Rep. 707, IRISA, Rennes, France, Mar. 1993.
- [6] K. P. Belkale and P. Banerjee, "Parallel algorithms for VLSI circuit extraction," *IEEE Trans. Comput.-Aided Design of Integrated Circuits Syst.*, vol. 10, pp. 604-618, May 1991.

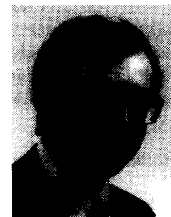
- [7] K.P. Belkale, R.J. Brouwer, and P. Banerjee, "Task scheduling for exploiting parallelism and hierarchy in VLSI CAD algorithms," *IEEE Trans. Comput.-Aided Design Integrated Circuits Syst.*, vol. 12, pp. 557-567, May 1993.
- [8] M. Bellon, Motorola Urbana Design Center, Urbana, IL, USA, personal commun., 1990.
- [9] P.A. Bernstein, "Sequoia: A fault-tolerant tightly coupled multiprocessor for transaction processing," *Comput.*, vol. 21, pp. 37-45, Feb. 1988.
- [10] N.S. Bowen and D.J. Pradhan, "Virtual checkpoints: Architecture and performance," *IEEE Trans. Comput.*, vol. 41, pp. 516-525, May 1992.
- [11] M.L. Ciacelli, "Fault handling on the IBM 4341 processor," in *Proc. 11th Int. Symp. Fault-Tolerant Computing*, 1981, pp. 9-12.
- [12] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann, 1990.
- [13] D.B. Hunt and P.N. Marinos, "A general purpose cache-aided error recovery (CAREER) technique," in *Proc. 17th Int. Symp. on Fault-Tolerant Computing*, 1987, pp. 170-175.
- [14] B. Janssens and W.K. Fuchs, "Experimental evaluation of multiprocessor cache-based error recovery," in *Proc. Int. Conf. on Parallel Processing*, 1991, pp. I-505-I-508.
- [15] —, "Relaxing consistency in recoverable distributed shared memory," *Proc. 23rd Int. Symp. Fault-Tolerant Computing*, 1993, pp. 155-163.
- [16] R.H. Katz *et al.*, "Implementing a cache consistency protocol," in *Proc. 12th Int. Symp. Computer Architecture*, 1985, pp. 276-283.
- [17] P.A. Lee, N. Ghani, and K. Heron, "A recovery cache for the PDP-11," *IEEE Trans. Comput.*, vol. C-29, no. 6, pp. 546-549, June 1980.
- [18] C.-C. J. Li, S.-K. Chen, W.K. Fuchs, and W.-M.W. Hwu, "Compiler-assisted multiple instruction retry," *IEEE Trans. Comput.*, to appear, 1994 (also available as Tech. Rep. CRHC-91-31, Univ. of Illinois, Urbana, IL, USA, Dec. 1991).
- [19] M.S. Papamarcos and J.H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," in *Proc. 11th Int. Symp. Comput. Architecture*, 1984, pp. 348-354.
- [20] J.H. Patel, "Analysis of multiprocessors with private cache memories," *IEEE Trans. Comput.*, vol. C-31, no. 4, pp. 296-304, Apr. 1982.
- [21] S. Patil, "Parallel algorithms for test generation and fault simulation," Ph.D. dissertation, Tech. Rep. CRHC-90-12, Univ. of Illinois, Urbana, IL, USA, Sept. 1990.
- [22] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, vol. SE-1, no. 2, pp. 220-232, June 1975.
- [23] L. Spainhower *et al.*, "Design for fault-tolerance in System ES/9000 Model 900," in *Proc. 22th Int. Symp. Fault-Tolerant Computing*, 1992, pp. 38-47.
- [24] C.B. Stunkel, B. Janssens, and W.K. Fuchs, "Address tracing of parallel systems in TRAPEDS," *Microprocessors and Microsyst.*, vol. 16, pp. 249-261, 1992.
- [25] Y. Tamir and M. Tremblay, "High-performance fault-tolerant VLSI systems using micro rollback," *IEEE Trans. Comput.*, vol. 39, pp. 548-554, Apr. 1990.
- [26] K.-L. Wu, W.K. Fuchs, and J.H. Patel, "Error recovery in shared memory multiprocessors using private cache," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, pp. 231-240, Apr. 1990.



**B. Janssens** (S'94) received the B.S. and M.S. degrees in electrical and computer engineering from the University of Illinois at Urbana-Champaign, USA, in 1987 and 1991, respectively.

At present, he is a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, USA. He is also a Research Assistant at the Coordinated Science Laboratory, focusing on recovery from errors in parallel systems. His research interests and experience include computer architecture, operating

systems, parallel and distributed computing, and fault-tolerant computing. He has held a summer research position at the IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, and a guest scientist position at Siemens Research, Munich, Germany.



**W.K. Fuchs** (S'83-M'85-SM'90) received the B.S.E. degree in electrical engineering from Duke University, Durham, NC, USA, in 1977, the M.S. degree in electrical engineering from the University of Illinois at Urbana-Champaign, USA, in 1982, the M.Div. degree from Trinity Evangelical Divinity School, Deerfield, IL, USA, in 1984, and the Ph.D. degree in electrical engineering from the University of Illinois in 1985.

He is currently a Professor in the Department of Electrical and Computer Engineering, Computer Science, and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, USA. His research interests include parallel computing and very large scale integration (VLSI) system design, with emphasis on reliable computation and testing.

Dr. Fuchs's research awards include the Senior Xerox Faculty Award for Excellence in Research, from the College of Engineering, University of Illinois (1993); selection as a University Scholar, University of Illinois (1991); appointment as a Fellow in the Center for Advanced Studies, University of Illinois (1990); the Xerox Faculty Award for Excellence in Research, University of Illinois (1987); the Digital Equipment Corporation Incentives for Excellence Faculty Award (1986-88); and the Best Paper Award, IEEE/ACM Design Automation Conference (DAC), simulation test category (1986). He was the Guest Editor of the May 1992 Special Issue on Fault-Tolerant Computing, IEEE TRANSACTIONS ON COMPUTERS, and was Co-Guest Editor of the April 1992 Special Issue on Wafer-Scale Integration Architectures and Algorithms, IEEE COMPUTER. He is currently a member of the Editorial Board for IEEE TRANSACTIONS ON COMPUTERS and IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS.