Fault-tolerant Distributed-Shared-Memory on a Broadcast-based Interconnection Network

Diana Hecht[1] and Constantine Katsinis[2]

[1]Electrical and Computer Engineering, University of Alabama in Huntsville, Huntsville, AL 35899, hecht@ece.uah.edu
[2]Electrical and Computer Engineering,Drexel University, Philadelphia, PA 19104, ckatsini@ece.drexel.edu

Abstract. The Simultaneous Optical Multiprocessor Exchange Bus (SOME-Bus) is a low-latency, high-bandwidth interconnection network which directly links arbitrary pairs of processor nodes without contention, and can efficiently interconnect over one hundred nodes. Each node has a dedicated output channel and an array of receivers, with one receiver dedicated to every other node's output channel. The SOME-Bus eliminates the need for global arbitration and provides bandwidth that scales directly with the number of nodes in the system. Under the distributed shared memory (DSM) paradigm, the SOME-bus allows strong integration of the transmitter, receiver and cache controller hardware to produce a highly integrated system-wide cache coherence mechanism. Backward Error Recovery fault-tolerance techniques can rely on DSM data replication and SOME-Bus broadcasts with little additional network traffic and corresponding performance degradation. This paper uses extensive simulation to examine the performance of the SOME-Bus architecture under DSM and Backward Error Recovery.

1 Introduction

In distributed shared memory (DSM) systems, an important objective of current research is the development of approaches that minimize the access time to shared data, while maintaining data consistency. The success of DSM depends on its ability to free the programmer from any operations that are necessary for the only purpose of supporting the memory model, and therefore it is critical that interconnection networks be developed, connecting hundreds of nodes with high-bisection bandwidth and low latency, that result in the least possible adverse impact on DSM performance.

As the number of nodes in the system implementing DSM increases, so does the likelihood that the system will experience node failures. For this reason, tolerating node failures becomes essential for parallel applications with large execution times. Coherence protocols for fault-tolerant DSM systems are described in [3]. A popular approach to fault tolerance, Backward Error Recovery (BER) enables an application which encounters an error to restart its execution from an earlier, error-free state. Much research is currently being conducted in order to exploit the data replication mechanism already existing in a DSM to reduce or hide the overhead of implementing BER. An example is the ICARE system [2], a software-based recoverable DSM for a network of workstations.

Due to advances in fiber-optics and VLSI technology it is possible to design an architecture that relies on broadcasts to support hardware-based DSM, allowing the implementation of coherence protocols at the cache block level through interactions of cache, memory and network interface controllers. Fault-tolerance protocols on this system rely on data replication and broadcasts to implement Backward Error Recovery resulting in little additional cost in terms of network traffic.

The Simultaneous Optical Multiprocessor Exchange Bus (SOME-Bus) [1] is such a network. One of its key features is that each node has a dedicated broadcast channel, realized by a specific group of wavelengths in a specific fiber, and an input channel interface based on an array of receivers shown in Fig. 1, which simultaneously monitors all channels. This design results in an effectively fully-connected network.

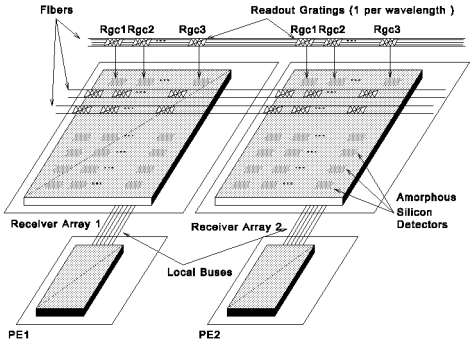

Fig. 1. SOME-bus Parallel Receiver Array and Output Coupler

Although the SOME-bus can utilize software techniques for implementing cache coherence, it allows strong integration of the transmitter, receiver and cache controller hardware to produce a highly integrated system-wide cache coherence mechanism.

2 Fault Tolerant DSM on the SOME-bus

A multicomputer system based on the SOME-Bus can be composed of hundreds of nodes. As the number of nodes in the system increases, however, so does the likelihood that the system will experience node failures or reboots. For this reason, the ability to tolerate node failures becomes essential for parallel applications with large execution times.

Backward Error Recovery is a mechanism for providing fault tolerance through the periodic saving of system state, which is known as checkpointing. Typically, in order to ensure that the checkpoint is consistent over the entire system, the processes are synchronized before the checkpoint is established and then resume normal execution after the checkpoint has been saved. Care must be taken that the checkpoint is saved to a storage medium that can be accessed after a failure occurs. In [2], checkpoints are saved in the memory of another node rather than to disk thereby reducing the amount of time that is required to save the checkpoint. We have adopted this approach for our simulations.

This paper explores the performance of a fault tolerant DSM system based on the SOME-bus. An event simulator was developed to evaluate the performance of the proposed system. The simulated multicomputer consists of a set of 64 nodes organized as a CC-NUMA system interconnected by the SOME bus architecture. The shared virtual address space is distributed across local memories which can be accessed both by the local processor and by processors from remote nodes, with different access latencies.

A multithreaded execution model is assumed and each processor executes a program which contains a set of parallel threads. A sequential consistency model is adopted and statically distributed directories are used to enforce a write-invalidate protocol. Each node contains a processor with cache, memory, an output channel and a receiver which can receive messages simultaneously on all N channels. Each node contains a processor controller, directory controller, cache controller and channel controller. There is a separate input queue associated with cache and directory controllers and the channel controller is associated with the output queue for the node. In addition, the directory controller contains an internal waiting-message queue used to hold the requests that are waiting for invalidation or downgrade acknowledge messages to be received.

We examine two mechanisms for providing fault tolerance in our DSM system. In both approaches, each node keeps a full copy of its local memory which will be referred to as its as recovery data. In addition, every node also contains a copy of the recovery data of another node. It is not necessary to recreate an entire copy of the memory at each checkpoint interval. Instead the checkpoints can be incrementally updated since it is only necessary to save the data that has been modified since the last checkpoint[2]. After the processors are synchronized in preparation for taking a checkpoint, all of the cache controllers write back any exclusively owned cache blocks. Once the directory controllers have received all of the cache writeback data, information that will be used to update the recovery memory is compiled. A copy of the compiled updates must be sent to the node with the backup copy of the recovery data. When all copies of the recovery data have been updated successfully, the processors synchronize again and then resume normal operation.

As the checkpoint interval increases, the total number of checkpoints taken over the course of the application execution decreases. Depending upon the data access pattern, however, the time required to perform the cache writeback, assemble the recovery data updates, and send them to the backup node will increase. This approach might lead to a burst of traffic appearing on the bus during the establishment of the checkpoint.

Another approach would be for a node to contain a copy of another node's local memory as well as the recovery data. If node X contains the recovery data and copy of node Y's data, we can refer to node Y as the home location for its local memory and node X as the home2 location for node Y's memory. If the cache coherence protocol was modified so that both copies of Node Y's memory remain consistent, updating the copy of Node Y's recovery data could be handled locally on Node X. Instead of sending all the data that was modified since the previous checkpoint, the cache controller will multicast the writeback blocks to both the home and home2 nodes.

This would reduce the burst of network traffic that occurs in the first approach during the establishment of a checkpoint. In order to keep both copies of the memory consistent,

additional coherence messages will be required. The number of invalidation, downgrade and writeback messages will have to be increased to include the home2 node. Since these messages can be multicast, the additional overhead for the SOME-bus will not be as high as it would for other types of interconnection architectures. Furthermore, the home2 data can be used to fill read requests. This increases the probability of having a local miss rather than a remote one. For example, it is possible to service a cache read miss on node X with either its own local memory or the copy of node Y's local memory.

Two algorithms, FT1 and FT2, were implemented based on the ideas described in the preceding paragraphs. The FT1 algorithm maintains two consistent copies of the local memory and the corresponding recovery data for each node. Updating both copies of the recovery data can be accomplished without additional traffic during the creation of the checkpoint. The FT2 algorithm keeps copies of the recovery data only, and must transmit updates to the node containing the remote copy during the creation of a new checkpoint.

In each of the experiments reported below, the simulation was executed for a period of 100,000 simulation time cycles. Data was collected for five checkpoint intervals (10000, 20000, 30000, 40000 and 50000), which resulted in a total of 9, 4, 3, 2, and 1, checkpoints for the FT1 algorithm and 8, 4, 3, 2, and 1 for the FT2 algorithm. Fig. 2 provides the average time requred to create a new checkpoint.

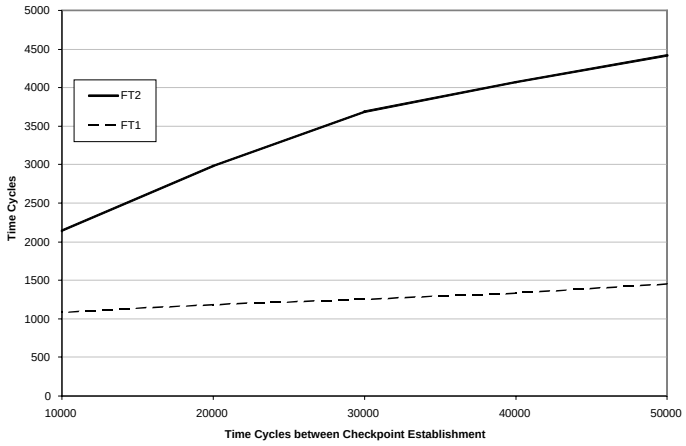

Fig. 2. Average Time Required to Establish a Checkpoint

As the checkpoint interval is increased from 10000 to 50000, the time required to create the checkpoint for the FT2 algorithm more than doubles while the time for the FT1 algorithm increases by less than 500 time cycles. The recovery data update messages are responsible for the large difference. The recovery data update messages will vary in size based on the number of distinct data elements modified since the previous checkpoint was established. As the checkpoint interval is increased, more data will be placed in the update messages. The message size affects the message transfer time and the time it takes for the directory controller to write the data to memory.

The FT1 algorithm involves some overhead due to the fault tolerance during normal operation. This is due to the necessity of keeping the home2 copy of a node's memory consistent with the home copy. For this reason, the processor and channel utilization for the FT1 algorithm does not change significantly when the checkpoint interval is varied. The overhead for the FT2 algorithm occurs only when establishing a checkpoint. The processor utilization for the FT2 algorithm decreased 12% for a 20% increase in checkpoint frequency. The channel utilization results indicate that the message transfer time is not the major cause of the decrease in performance for the FT2 algorithm. Although the processor utilization declines when the checkpoint interval is reduced, the channel utilization remains almost the same.

The service time for the output queues and directory waiting-message queues does not change dramatically for either of the algorithms when the checkpoint interval is changed. The service time for the FT1 algorithms, however, is double that of the FT2 algorithm for both queues. The increase in the output queue is due to the increase in the number of messages required to keep the second copy of memory consistent. When a message is removed from the directory input queue or waiting-message queue, the directory begins to process the message. The Directory service time represents the time necessary to make the required accesses to the local memory. The directory service time is the major source of overhead for the FT1 algorithm. The increase in service time is due to the additional time each node must spend reading or writing to the home2 copy of another node's memory. The additional memory access time required for the coherence messages reduces the ability of the directory controller to process the normal data and ownership requests in a timely manner.

3 Conclusion

Advances in optical technology have made broadcast interconnection networks a realistic, highly competitive alternative that can achieve high bandwidth, low latency and large fan-out. Backward Error Recovery fault-tolerance techniques can rely on SOME-Bus broadcasts and DSM data replication with little additional network traffic and corresponding performance degradation. This paper used extensive simulation to examine the performance of the SOME-Bus architecture under DSM and Backward Error Recovery. Two mechanisms were proposed to implement fault-tolerance on a SOME-bus system. The performance of the two algorithms was evaluated and compared using a simulated SOME-bus-based DSM. The results showed a minimal impact on the performance (less than 16% for FT1) for the simulated system with the addition of fault tolerance.

References

1. Kulick J. H., W. E. Cohen, C. Katsinis, "The Simultaneous Optical Multiprocessor Exchange Bus," IEEE Conference on Massively Parallel Processor Optical Interconnects, 1995.
2. Kermarrec, A-M, Morin, C., Banatre, M., "Design, implementation and evaluation of ICARE: an efficient recoverable DSM", Software - Practice and Experience, vol. 28, no. 9, pp. 981-1010, 25 Jul 1998.
3. Theel O., Fleisch B., "A dynamic coherence protocol for distributed shared memory enforcing high data availability at low costs", IEEE Transactions on Parallel and Distributed Systems", vol.7, no.9, p. 915-30, Sept. 1996.