[Dal91a] William J. Dally. Express cubes: Improving the performance of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 40(9):1016–1023, September 1991.

[Dal91b] William J. Dally. Fine-grain concurrent computing. *Research Directions in Computer Science: An MIT Perspective*, pages 127–154, 1991.

[DKO+84] David J. DeWitt, Randy H. Katz, Frank Olken, Lenard D. Shapiro, Michael R. Stonebraker, and David Wood. Implementation techniques for main memory database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–8, June 1984.

[DS87] William J. Dally and Charles L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[EJZ92] Elmootazbellah Nabil Elnozahy, David B. Johnson, and Willy Zwaenepoel. The performance of consistent checkpointing. In *Proceedings of the 11th Symposium on Reliable Distributed Systems*, 1992.

[FB89] S. Feldman and C. Brown. Igor: A system for program debugging via reversible execution. *ACM SIGPLAN Notices, Workshop on Parallel and Distributed Debugging*, 24(1):112–123, Jan 1989.

[FR86] R. Fitzgerald and R.F. Rashid. The integration of virtual memory management and interprocess communication in accent. *ACM Transactions on Computer Systems*, 4(2):147–177, May 1986.

[Gib90] Garth Alan Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*. PhD thesis, University of California, Berkeley, December 1990.

[GLL+90] Kourosh Gharchorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *The 17th Annual International Symposium on Computer Architecture*, pages 148–159, Seattle, Washington, May 1990.

[Hag86] Robert B. Hagmann. A crash recovery scheme for a memory-resident database system. *IEEE Transactions on Computers*, C-35(9):839–843, September 1986.

[Joh89] David B. Johnson. *Distributed System Fault Tolerance Using Message Logging and Checkpointing.* PhD thesis, Rice University, December 1989.

[JZ89] David B. Johnson and Willy Zwaenepoel. Recovery in distributed systems using optimistic message logging and checkpointing. *Computer Systems Research at Rice University: Annual Report 1988-1989*, pages 83–102, 1989.

[KMBT91] M. Frans Kaashoek, Raymond Michiels, Henfy E. Bal, and Andrew S. Tanenbaum. Transparent fault-tolerance in parallel Orca programs. Technical Report IR-258, Vrije Universiteit, Amsterdam, October 1991.

[KS86] Henry F. Korth and Abraham Silberschatz. *Database System Concepts.* McGraw-Hill, New York, 1986.

[KT87] Richard Koo and Sam Toueg. Checkpointing and rollback-recovery for distributed systems. *IEEE Transactions on Software Engineering*, SE-13(1):23–31, January 1987.

[Lam81] B.M. Lampson. Atomic transactions. In B.M. Lampson, M. Paul, and H.J. Siegert, editors, *Distributed Systems – Architecture and Implementation*, pages 246–264. Springer-Verlag, 1981.

[LH89] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *TOCS*, 7(4):321–359, Nov 1989.

[LLG+90] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy. The directory-based cache coherence protocol for the dash multiprocessor. In *The 17th Annual International Symposium on Computer Architecture*, pages 148–159, Seattle, Washington, May 1990.

[LN88] Kai Li and Jeffrey F. Naughton. Multiprocessor main memory transaction processing. In *Proceedings of the International Symposium on Database in Parallel and Distributed Systems*, pages 177–187, Dec 1988.

[LNP90] Kai Li, Jeffrey Naughton, and James Plank. Real-time, concurrent checkpoint for parallel programs. In *Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 79–88, Seattle, Washington, Mar 1990.

[LNP91] Kai Li, Jeffrey F. Naughton, and James S. Plank. Checkpointing multicomputer applications. In *Proceedings of the 10th Symposium on Reliable Distributed Systems*, pages 1–11, October 1991.

[LNP92] Kai Li, Jeffrey F. Naughton, and James S. Plank. An efficient checkpointing method for multicomputers with wormhole routing. *International Journal of Parallel Processing*, 20(3), June 1992.

[LS92] Michael Litzkow and Marvin Solomon. Supporting checkpointing and process migration outside the Unix kernel. In *Conference Proceedings, Usenix Winter 1992 Technical Conference*, pages 283–290, January 1992.

[LY87] Ten H. Lai and Tao H. Yang. On distributed snapshots. *Information Processing Letters*, 25:153–158, May 1987.

[MS87] P.R. McJones and G.F. Swart. Evolving the unix system interface to support multithreaded programs. Tech Report 21, DEC Systems Research Center, Sep 1987.

[Nug88] Steven F. Nugent. The iPSC/2 direct-connect communications technology. In *Proceedings of the Third Hypercube Conference*. ACM, November 1988.

[OCD+88] J.K. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch. The sprite network operating system. *IEEE Computer*, 21(2):23–36, February 1988.

[Pie88] Paul Pierce. *The NX/2 Operating System*, pages 51–57. Intel Corporation, 1988.

[Pie89] Paul Pierce. A concurrent file system for a highly parallel mass storage subsystem. In *Proceedings of the 4th. Conference on Hypercubes, Concurrent Computers and Applications*, volume I, pages 155–160, March 1989.

[PL88] Douglas Z. Pan and Mark A. Linton. Supporting reverse execution of parallel programs. In *Proceedings of the ACM Workshop on Parallel and Distributed Debugging*, Madison, Wisconsin, May 1988.

[PP83] Michael L. Powell and David L. Presotto. Publishing: A reliable broadcast communication mechanism. In *Proceedings of the ACM SIGOPS Symposium on Operating System Principles*, pages 100–109, october 1983.

[Pu85] Calton Pu. On-the-fly, incremental, consistent reading of entire databases. In *Proceedings of the Eleventh International Conference on Very Large Databases*, pages 369–375, Stockholm, Sweden, August 1985.

[Ran75] B. Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, SE-1(2):220–232, 1975.

[RLW85] Paul Rovner, Roy Levin, and John Wick. On extending modula-2 for building large, integrated systems. Research report 3, DEC Systems Research Center, 1985.

[Sei88a] C. L. Seitz *et al.* The architecture and programming of the Ametek series 2010 multicomputer. In *Proceedings of the Third Hypercube Conference*, pages 33–37. ACM, January 1988.

[Sei88b] C. L. Seitz *et al.* Submicron systems architecture project semiannual technical report. Technical Report TR-88-18, Caltech Computer Science Tech. Rep., November 1988.

[SGM87] Kenneth Salem and Hector Garcia-Molina. Checkpointing memory-resident databases. Technical Report CS-TR-126-87, Department of Computer Science, Princeton University, 1987.

[SK86] Madalene Spezialetti and Phil Kearns. Efficient distributed snapshots. In *Proceedings of The Sixth International Conference on Distributed Computing Systems*, pages 382–388, Cambridge, Massachusetts, May 1986. IEEE Computer Society.

[SP88] Abraham Silberschatz and James L. Peterson. *Operating Systems Concepts*. Addison-Wesley, Reading, Mass., 1988.

[Sta89] Mark E. Staknis. Sheaved memory: Architectural support for state saving and restoration in paged systems. In *Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 96–103, April 1989.

[SY85] Robert E. Strom and Shaula Yemini. Optimistic recovery in distributed systems. *ACM Transactions on Computer Systems*, pages 204–226, August 1985.

[TLC85] Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable remote execution facilities for the V-system. In *Proceedings of the Tenth ACM Symposium on Operating System Principles*, pages 2–11, Orchas Island and Washington, December 1985.

[TS87] C. P. Thacker and L. C. Stewart. Firefly: a multiprocessor workstation. In *Proceedings of Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 164–172, Oct 1987.

[TW86] David J. Taylor and Michael L. Wright. Backward error recovery in a UNIX environment. In *16th Annual International Symposium on Fault-Tolerant Computing Systems*, pages 118–123. IEEE Computer Society, July 1986.

[Ven89] S. Venkatesan. Message-optimal incremental snapshots. In *Proceedings of The Ninth International Conference on Distributed Computing Systems*, pages 53–60, Newport Beach, California, June 1989. IEEE Computer Society.

[Wit89] Larry D. Wittie. Debugging distributed C programs by real-time replay. *SIGPLAN Notices*, 24(1):57–67, January 1989.

Appendix A

Implementational Details of ICKP

As stated in Chapter 4, the iPSC/860 presents a somewhat different model of computation than the one used Chapter 2. The reason is that iPSC/860 messages are buffered in the kernel until the user process extracts them, thereby hiding the FIFO ordering of messages with different types. This section presents the mechanism implemented in ICKP that allows us to deduce a FIFO ordering for all messages, regardless of their types.

A.1 The M_{EXTRA} Message

Our solution to the kernel buffering problem is to send two messages for every user message. The first has a special reserved type, and is called M_{EXTRA}. Its contents consist of the type of the user message. The second message is the user message itself. With the M_{EXTRA} message, `crecv`, `irecv` and `hrecv` can all deduce in which order their messages have been received in the kernel by observing the order of the M_{EXTRA} messages. In the Chandy-Lamport and Network Sweeping algorithms, the m_{CL} and m_{NS} messages are implemented as M_{EXTRA} messages whose contents are reserved types. Therefore, ICKP can

use the ordering of the M_{EXTRA} messages to determine which messages to log, and which messages not to log.

The overhead imposed by these M_{EXTRA} messages is that the total number of messages sent by an application program is doubled. However, the M_{EXTRA} messages are all short (4 bytes), and therefore take a small period of time to send: around 0.001 seconds. In practice, the overhead imposed by M_{EXTRA} messages on the test programs of Chapter 4 never exceeds 1% of the total running time.

A.2 Extracting Message State from the Kernel

When ICKP has a node take a local checkpoint, there are three types of information in the kernel that must be extracted as part of the checkpoint:

- The buffered messages that have not been `recv`'d by the user process.
- `Irecv` requests. In these, the user's program has called `irecv`, and either the message has not been received by the kernel, or the user has not tested whether the `irecv` has completed.
- `Hrecv` requests. Here, the user has called `hrecv` for a certain message type, and either the message has not been sent, or it has, but the `hrecv` handler cannot be invoked until ICKP relinquishes control (this is because ICKP gains control via a `hrecv` handler).

ICKP extracts this information in a straightforward way. At checkpoint time, ICKP copies all messages that are in the system buffers to a queue in user space (this can be done by calling `crecv` for each message type). This queue is saved in the checkpoint as part of the processor's address space. Thus, whenever a node executes `crecv`, `irecv` or `hrecv`, it first checks the user queue for the message. If present in the user queue, the message is taken from that queue. Otherwise, the correct `recv` system call is made.

To deal with `irecv` and `hrecv` calls at checkpoint time, the checkpointer maintains a queue of pending `irecv` and `hrecv` calls in user space. At checkpoint time, ICKP tests all the `irecv`'s for completion, and marks the result in the queue. Upon recovery ICKP uses this information to decide whether to redo the `irecv` call: It makes a new `irecv` call for the `irecv`'s that were not completed at checkpoint time. No new call is made for `irecv`'s that were completed.

`Hrecv` calls are more subtle. ICKP uses the M_{EXTRA} messages to determine whether `hrecv` messages have been received at checkpoint time. If not, ICKP simply remakes the `hrecv` call upon recover. However, if an `hrecv` message has been received at checkpoint time, and is waiting for ICKP to finish before invoking the handler, then ICKP must remember that in the checkpoint, so that it may invoke the handler immediately following recovery.

Thus, the message state present in the kernel may be extracted and saved as part of the checkpoint, and rebuilt at recovery time.

Appendix B

Tables for the Multiprocessor Implementation

Heap Size (Mbytes)	Checkpoint Time (sec)			
	Sequential	Main Memory	CLL	Copy-on-Write
0.789	3.64	6.58	5.54	7.13
2.985	10.20	18.03	14.00	19.62
3.962	13.02	22.70	17.24	24.43
4.938	15.81	27.52	21.08	30.52
7.868	24.96	44.83	32.41	46.70
9.821	30.22	57.32	39.83	62.47
12.751	40.48	673.89	52.53	474.13

Table B.1: Checkpoint Time of Multiprocessor Algorithms