Architecture and Realization of the Modular Expandable Multiprocessor System MEMSY

M. Dal Cin, W. Hohl, S. Dalibor, T. Eckert, A. Grygier, H. Hessenauer, U. Hildebrand, J. Hönig, F. Hofmann, C.-U. Linster, E. Michel, A. Pataricza[1], V. Sieh, T. Thiel, S. Turowski

Universität Erlangen-Nürnberg
IMMD III, Martensstraße 3, 91058 Erlangen, FRG
hohl@informatik.uni-erlangen.de

Abstract

The experimental multiprocessor system MEMSY[2] will be described. This system was built to validate the concept of a scalable multiprocessor architecture based on local shared-memory. Main application areas are scientific computations with high demand for processing power and memory capacity. In designing the hardware architecture the extensive use of standard components and fault tolerance were prerequisites. The programming model of MEMSY is custom made reflecting its true hardware structure whereas the operating system is a Unix extension.

Keywords: Multiprocessor, MIMD, Scalability, Fault Tolerance

1. Introduction

There are some well known reasons to develop massively parallel multiprocessors [7] as:

- the demand for increasing amount of computing power and memory capacity, to attack e.g. the "Grand Challenges" [5]
- the insight that a broad spectrum of these applications is unsuitable to vector processors
- the evolution in technology towards very large scale integration connected with decreased cost, increased speed and a dramatic reduction in power consumption
- extremely better price/performance ratio of RISC to vector processors
- technological limitations for the mainframes.

Among the different kinds of parallel computers MIMD-systems are especially versatile due to their independent instruction streams. In this class of multiprocessor systems, those with global shared-memory are normally the favourite of application programmers because of the simple programming model.

Main application areas comprise scientific computations with high demand for processing power and memory capacity, such as numerical simulation of physical phenomena or quantum mechanical computations. Analysis of typical problems reveals that a broad class of these problems is regularly structured and can easily be partitioned and mapped onto a regularly structured multiprocessor. Therefore, global shared-memory is not what is really needed by these applications. Local shared data is sufficient to solve the problems.

Besides, multiprocessors with global shared-memory all suffer from a lack of scalability. By making clever use of fast buses and caching techniques this effect may be postponed, but each system has an upper limit on the number of processing nodes. We believe that our type of shared-memory called *distributed shared communication memory* can be more efficient than virtual shared-memory implementations. Therefore, we decided not to build a global or virtual shared-memory machine. Our machine is called MEMSY - Modular Expandable Multiprocessor System.

Furthermore, massively parallel systems like MEMSY represent a new challenge for fault tolerance. The designers of such systems cannot expect that no parts of the system will fail. With the significant increase in the complexity and number of components the chance of a single or multiple failure is no longer negligible. It is clear that the redundancy, reconfigurability and diagnosis techniques must therefore be considered not as a subsequent add-on but at the design stage itself.

1. Guest researcher from TU Budapest, Dept. Measurement and Instrumentation Engineering

2. Supported by the DFG (Deutsche Forschungsgemeinschaft) as part of the "Sonderforschungsbereich" SFB 182

First Intl. Conf. on Massively Parallel Computing Systems (MPCS'94), Ischia, May 2 - 6, 1994,
IEEE 1994, pp.7-15

The MEMSY architecture was defined with the following design goals in mind [8]:

- *Efficiency*: The system should be based on state-of-the-art high performance microprocessors. The computing power of the system should be high enough to handle problems of scientific and engineering research.
- *Scalability*: The architecture should be scalable with no theoretical limit. The communication network should grow with the number of processing elements in order to accommodate the increased communication demands in larger systems.
- *Flexibility*: The architecture should be usable for a great variety of user problems.
- *Fault-Tolerance*: The system should provide long uninterrupted computation times and reliable access to a large amount of data over a long time span.
- *Economy*: The system should be almost completely based on off-the-shelf components.

In the following we give an overview on the realization of these goals. In Section 2 the hardware architecture of MEMSY is described. In Section 3 and 4 we present the programming model and the operating system, respectively. Section 5 describes some fault tolerance aspects of the system.

2. Hardware Architecture

The MEMSY architecture consists of tightly coupled processor nodes arranged in a two-level hierarchy of four-neighbour toroidal grids. The processors of nearest-neighbouring nodes communicate through shared memories (called *shared communication memor*y). All nodes have the same internal structure.

2.1 Topology of MEMSY

MEMSY consists of two levels of nodes corresponding to their different tasks: applications at the A-level and system control at the B-level (Fig. 1). At each level, the processor nodes form a rectangular grid closed to a torus. Each processor node has an associated communication memory, which it shares with its four neighbouring processor nodes (not shown in Fig. 1). Each node has its own disk and access to the front-end by FDDI connection (Fig. 2). The B-level processors are responsible for long-distance communication and data transfer to the global disk.

A processing element of the upper level has access to the shared-memory modules of the four processing elements

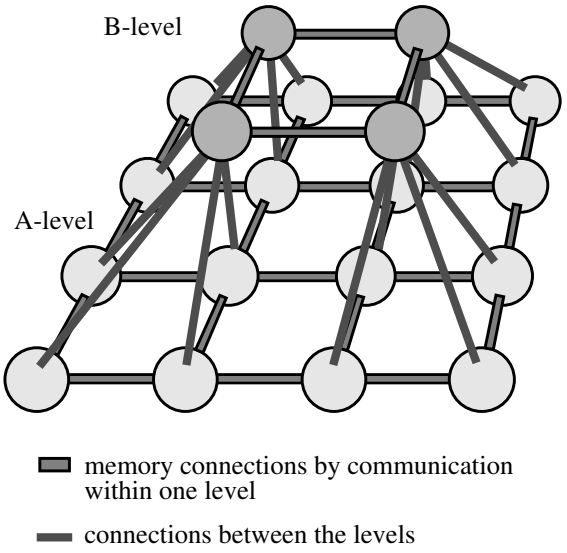

Fig. 1 MEMSY topology

directly below it, thereby forming a small pyramid. There are four times as many processing elements in the lower level than in the upper.

This topology offers constant local interconnection complexity, an important condition for scalable systems.

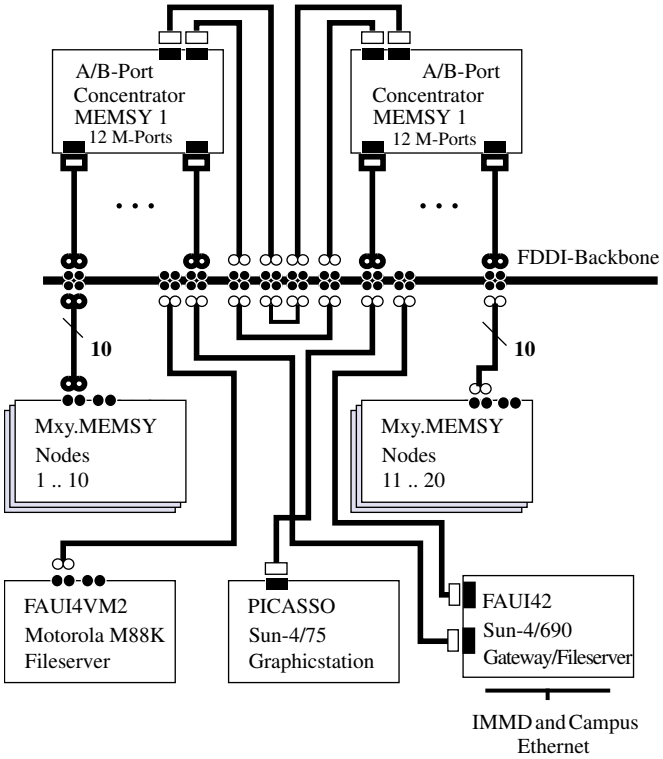

Fig. 2 FDDI Connections

2.2 The MEMSY prototype

We have built an experimental prototype consisting of four functional units:

- 4 + 16 *processor nodes* with 4 processors each
- one *communication memory* at each node
- the *interconnection network* which provides the communication paths between processor nodes and communication memories
- FDDI *interconnection* for long distance data exchange and connection to the front-end.

Fig. 3 shows four racks containing four communication memories and three coupling units each with the corresponding interconnections in the upper level and processor nodes under that.

This system containing 20 processor nodes with a total of 80 processors offers the following performance data:

- 2000 MIPS peak, ca. 1440 MIPS sustained
- 1000 MFLOPS peak, ca. 800 MFLOPS sustained
- 10 MB physical cache
- 640 MB local memory (expandable to 3840 MB)
- 80 MB communication memory (expandable to 320 MB)
- 10 GB local disc (expandable)
- Bandwidth 100 MB/sec I/O to disc (expandable to 300MB/sec)
- 100 Mbit/sec FDDI
- 400 MB/sec to local memory
- 177 MB/sec to communication memory

2.3 Processor nodes

Each of the processor nodes of MEMSY is based on the Motorola MVME 188 board system supplemented by additional hardware which was designed and implemented within the project. (In Fig 4 showing the logical node structure these parts have a lighter background colour.)

The MVME188 board consists of four VME modules: The system controller board, holding e.g. timers and serial interfaces; two memory boards, each holding 16 MB local memory; and the main logic board, carrying the processor modules.

The processor module comprises four MC88100 RISC CPUs and eight MC88200 cache and memory management units (CMMU), which provide eight 64 KB caches. Internally the processors have a Harvard-architecture, i.e. each one has a separate data and instruction bus.

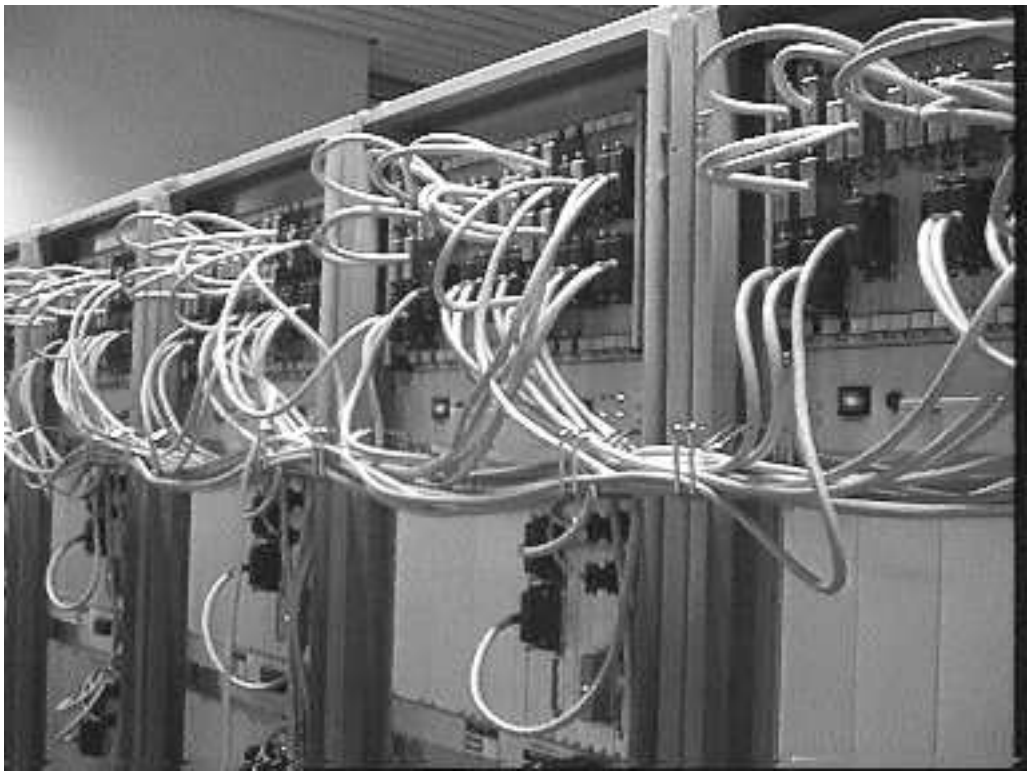

Fig. 3 Prototype hardware

Special features of a processor module are:

- *Cache coherency* is supported by hardware
- There exists a *cache copyback* mode which writes data back to memory only if necessary and a *write-through* mode
- There exists an atomic memory access which is necessary for efficiently implementing spinlocks and semaphores in a multiprocessor environment
- The caches provide a burst mode allowing atomic read/write access of four consecutive words while supplying only one address.

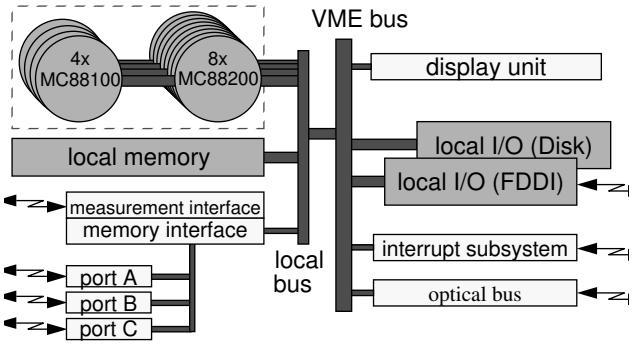

Fig. 4 Logical node structure

All VME modules mentioned above, are interconnected by a high speed local bus. Each node can be extended by additional modules via this local bus or the VME bus. The communication memory interface is attached to the local bus. Its function is to recognize and execute accesses to the communication memories. The interface hardware provides three ports to which the communication memories are connected either directly or via an interconnection network (described in Section 2.4).

To the processor node the memory interface looks like a simple memory module. The address decoder of the MVME188 works in such a way, that the highest two address bits determine whether the address space of the local memory boards, of the VME bus or of the memory interface is accessed. In case of the memory interface, the next two address bits determine the port which should be used for each memory access. Four further bits of the address determine the path through the coupling unit and which communication memory is to be accessed.

Addresses and data are transferred in multiplexed mode. The connection is 32 data bits plus four parity bits wide. The parity bits are generated by the sender and checked by the receiver. If the communication memory detects a parity error in address or data, it generates an error signal, otherwise a ready signal is generated. If the memory interface receives an error signal or detects a parity error during a read access, it transmits the error signal to the processor, otherwise a ready signal is sent.

The memory interface hardware supports the atomic memory access and the burst mode. Counters have been included in the interface hardware to count the various types of errors in order to investigate the reliability of the connection. The counters can be read and reset by a processor. In addition, there is a status register which contains information about the last error occurred. This can be used in combination with an error address register to investigate the error.

In addition, the memory interface contains a measurement interface to which an external monitor can be connected. A measurement signal is triggered by a write access to a particular register of the memory interface and the 32-bit word written is transferred to the monitor.

2.4 Interconnection network

According to the MEMSY topology, each node has access to its communication memory and to the communication memories of four neighbouring nodes. Additionally, every node of the B-level has access to the communication memories of four assigned A-level-nodes.

A static implementation of this topology requires 9 ports at each node and 6 ports at each communication memory. To reduce this complexity, a dynamic network component, called *coupling unit*, has been developed. The use of these coupling units reduces the number of ports needed at the memory interface and the communication memory to three.

Each coupling unit is a blocking, multistage, dynamic network with fixed size. It provides logically complete interconnections between 4 input ports and 4 output ports. Thus, the interconnection structure of MEMSY is a hybrid network with global static and local dynamic network properties, cf. Fig. 5.

Each node and each memory module is connected to two coupling units such that the nearest-neighbouring torus topology can easily be established. A square torus with $N=n^2$ nodes requires N/2 coupling units. The connection of each B-level node to the four communication memories of the A-level is also implemented by coupling units connected to the third node port.

In our implementation of the interconnection network, accesses to the communication memories via coupling units are executed with an efficient memory access proto-

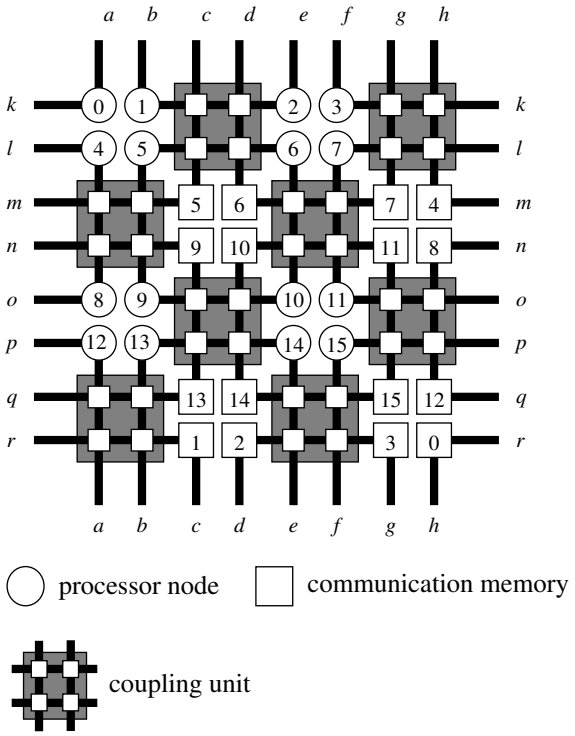

Fig. 5 Interconnection structure of a MEMSY level

col [6]. The interconnection network operates in a circuit-switching mode by building up a direct path between a node and a communication memory for each memory access.

A coupling unit consists of the following subcomponents (Fig. 6):

- 4 p-ports allowing the access to the coupling unit by processor nodes
- 4 m-ports providing the connection to communication memories
- 4 internal subpaths performing data transfer within the coupling unit
- 1 control unit which controls the dynamic interconnection between p-ports and m-ports
- 4 switching elements providing the dynamic interconnection of p-ports and m-ports

The port structure is basically identical to a memory interface with a multiplexed 32 bit address / data bus. The direction of the control flow is different for p-ports and m-ports. An activity (a memory access) can be only initiated at a p-port.

The control unit always has the complete information about the current switch settings of all switching elements. If a new request is recognized by receiving a valid address, the control unit can decide at once whether the requested access can be performed or has to be delayed. A request is accepted if all switching elements contained in the communication path to be built-up are either inactive or possess the switch settings required for the interconnection.

Due to the longer transfer path and the switch setting, an access to shared data in the communication memories requires a significantly higher access time than an access within the node. Shared memory access time in our prototype implementation is normally 1 µs and up to 1.3 µs if blocking occurs due to a quasi-simultaneous access.

Since only data which is held in the communication memories, such as boundary values of subarrays, the increased access time has only a small influence on the overall computing time as measurements have shown. Thus, reducing the complexity of the network by using the coupling units causes only a small reduction in performance compared to a static point to point network.

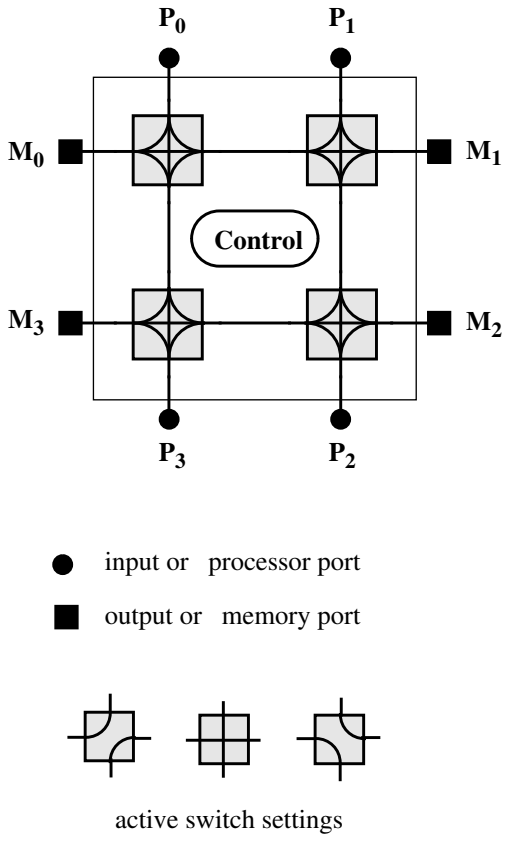

Fig. 6 Internal structure of a coupling unit

3. Programming Model

The programming model of MEMSY was designed to give the application programmer direct access to the power of the system. Unlike in many systems, where the programmer's conception of the system is different from the real structure of the hardware, the application programmer for MEMSY should have a conception of the system which is very close to its real structure. In our opinion this enables the programmer to write highly efficient programs which make the best use of the system [8].

In addition, the programmer should not be forced to a single way of using the system. Instead, the programming model defines a variety of different mechanisms for communication and coordination[1]. From these mechanisms the application programmer may pick the ones which are best suited for his particular problem.

The programming model is defined as a set of library calls which can be called from C and C++. It is open for extensions which will be based on experiences we gain from real applications. Specific problems will show whether additional mechanisms for communication and coordination are needed and how they should be defined. The mechanisms presently provided by the programming model are:

Shared-Memory. The use of the shared-memory is based on the concept of 'segments', very much like the original shared-memory mechanism provided by UNIX System V. A process which wants to share data with another process (possibly on another node) first has to create a shared-memory segment. To have the operating system select the correct location for this memory segment, the process has to specify the neighbouring nodes which share the segment.

After the segment has been created, other processes may map the same segment into their address space by means of an 'attach' operation. Since addresses in these address spaces are unrelated, pointers may not be passed between the processes. Segments may be unmapped and destroyed dynamically.

There is a disadvantage to the shared-memory implementation on MEMSY. To ensure a consistent view for all nodes the caches of the processors must be disabled for accesses to the shared communication memory. But the application programmer may enable the caches for a single segment as long as he is sure that inconsistencies between the caches on different nodes are not possible, for example, if only one node is using this segment or if it is only being read.

1. We use the term *coordination* instead of *synchronization* to express that not the simultaneous occurring of events (e.g. accesses to common data structures) is meant but their controlled ordering.

There are two different message mechanisms offered by the programming model, messages and transport:

Messages. The first message mechanism allows the programmer to send short (2 word) messages to another processor. The messages are buffered at the receiving side and can be received either blocking or non-blocking. They are mainly used for coordination and are not optimized for high-volume data transfer.

Transport. The transport mechanism was designed to allow for high volume and fast data transfer between any two processors in the system. The operating system is free to choose the method and the path this data is to be transferred on (using shared-memory, FDDI-ring or bus). It can take into account the current load of the processing elements and data paths.

Semaphores. To provide a simple method for global coordination, semaphores have been added to the programming model. They reside on the node on which they have been created, but can be accessed uniformly throughout the whole system.

Spinlocks. Spinlocks are coordination variables which reside in shared-memory segments. They can be used to guard short critical sections. In contrast to the other mechanisms this is implemented totally in user-context using the special machine instruction 'XMEM'. The main disadvantage of the spinlocks is the 'busy-wait' performed by the processor. This occurs if the process fails to obtain the lock and must wait for the lock to become free. To minimize the effects of programming errors on other applications, a time-out must be specified, after which the application is terminated (there is a system-imposed maximum for this time-out).

I/O. Traditional UNIX-I/O is supported. Each processing element has a local data storage area. There is one global data storage area which is common to all processing nodes.

To express parallelism the programmer has to create multiple processes on each processing element by a special variant of the system call 'fork'. Parallelism between nodes is handled by the configuration to be defined: an initial process is started by the application environment on each node on which the application should run. In the current implementation these initial processes are identical on all nodes.

Processes can obtain various information from the operating system regarding their positions in the whole system and the state of their own or other nodes.

4. Operating System

The operating system of MEMSY - MEMSOS - is based on the UNIX SYSTEM V/88 Release 3 of Motorola. Instead of designing and implementing a completely new operating system we have decided to use an existing system and adapt it to the new hardware [7]. Unix supplies a good development environment and many useful tools. The multitasking / multiuser feature of Unix is included with no additional effort.

Each processor node has its local operating system which is adapted to the multiprocessor architecture of the processor board. The operating system has a peer processor architecture, meaning that there is no special designated processor, e.g. master processor. Every processor is able to execute user code and can handle all I/O requests by itself. The kernel is divided into two areas. One area contains code that can be accessed in parallel by all processors, because there is either no shared data involved or the mutual exclusion is achieved by using fine grain locks. The second area contains all the other code that can not be accessed in parallel. This area is secured with a single semaphore. For example, all device drivers can be found here. In SYSTEM V/88 the usual multiprocessor concepts are implemented, such as message-passing, shared-memory, interprocessor communication and global semaphores.

The original operating system is, however, not able to deal with distributed memory such as our communication memory. Therefore, certain extensions and additions have been made to it. Only little changes have been made to the kernel itself. Standard Unix applications are runnable on MEMSY because the system-call interface stayed intact.

The so called application concept [8] makes it possible to control and monitor distributed user programs. Hence, more than one application can be allowed to run in parallel on MEMSY.

Because MEMSY is an experimental multiprocessor system we implemented as many communication mechanisms as possible. An important aspect in doing this was to be able to compare their usefulness and performance and, therefore, be able to validate our multiprocessor concept.

5. Aspects of Fault Tolerance

Fault tolerance is the ability of a system to tolerate the presence of a bounded number of faults and to continue in operation until scheduled maintenance can take place. Fault tolerance requires redundancy in hardware, software or time.

It is obvious that for massively parallel computers hardware redundancy has to be employed as sparingly as possible. Fault handling may be, therefore, more adequate and more cost-effective than providing fault-masking hardware redundancy. This holds true particularly for very small failure rates. Fault handling includes error detection, error location, fault confinement and damage assessment as well as error recovery and fault treatment in conjunction with continued service to the user. These techniques must be implemented in a manner that does not severely affect performance and scalability. Yet, high error coverage and low error latency have to be achieved.

The fault tolerance of MEMSY [4] is based on:

- the fault-tolerant interconnection network architecture with integrated communication memory modules
- concurrent error detection
- concurrent, application controlled checkpointing.

5.1 Concurrent error detection

The primary goal in designing error detection mechanisms is high fault coverage, while keeping redundancy on a moderate level. This can be achieved by combining built-in standard mechanisms on chip level with efficient hardware error detection mechanisms on system level [3].

Concurrent error detection implies that each processing node of a massively parallel computer has the capability to check itself concurrently to program execution. Testing and test management are too time consuming. Moreover, the majority of faults in a computer is transient [18] and cannot be detected with high enough probability by (off-line) testing.

Although software techniques can be used for error detection, they are, however, likely to have high error latency. Therefore, we explored different hardware solutions for concurrent error detection:

- duplication of each processing node and lock step comparison of results; for its realization in MEMSY see [2], [9]
- attachment of watchdog processors to monitor the program execution of one or more processing nodes asynchronously [12]. Task of a watchdog processor is the on-line detection of sequencing errors as well as of instruction modification errors in program execution; for realization in MEMSY see [13],[14],[15].

Failures in communication lines and memories are detected by error detecting codes.

5.2 Fault treatment

As soon as an error is detected, it must be handled. If rollback is part of the error handling mechanism, each processing node must periodically checkpoint its state in a memory with stable storage property. The stable storage property is necessary to hinder the faulty node to corrupt its state information and to protect this information against latent memory errors. (Latent memory errors can be detected e.g. by memory scrubbing.) With checkpointing, a processing node is enabled to resume a failed computation by reading the relevant state information in case of a temporary fault. Checkpointing also enables a functioning node to resume computation of a failed node in case of a permanent fault (reconfiguration). In massively parallel systems as in MEMSY there is no global memory. Hence, checkpoints are to be stored distributed in such a way that a consistent global checkpoint can be maintained, even if some nodes or communication links become faulty.

Therefore, the rollback-recovery scheme for MEMSY is based on the notion of distributed snapshots [1], [11]. The state of an application program is defined by the state of every participating process plus the state of its shared memory segments. Other approaches, such as conversations [16], [17] or message logging schemes [10] are not considered suitable, the latter because logging is impossible, as shared-memory communication does not necessarily involve a message passing programming model.

Furthermore, we consider numerical applications as the primary use for memory-coupled multiprocessors [8]. These applications are characterized by high communication loads. Most numerical applications are basically iteration loops, offering suitable spots for global checkpointing. Thus, to optimize checkpointing, we assume the programmer specifies the type of data to be stored and the exact places within the execution of a program, when checkpoints shall be taken. A fully transparent scheme would cost much more coordination overhead.

6. Conclusions

At the University of Erlangen-Nürnberg the fault-tolerant highly parallel computer MEMSY (Modular Expandable Multiprocessor System) for numerical applications has been developed and realized and is now under evaluation. MEMSY is a hierarchically organized, distributed memory MIMD computer with locally shared memory. The potential size of the system is unlimited because of its interconnection structure with constant local complexity. The communication between neighbouring processors (arranged as a toroidally closed array) is based on a fault-tolerant interconnection subsystem consisting of coupling modules and multiport memories allowing mutual access.

Acknowledgments

Authors want to thank the Deutsche Forschungsgemeinschaft (DFG) for supporting this work as part of the Collaborative Research Center 182 and all the colleagues involved in the design and realization of MEMSY.

References

[1] Chandy, K. M.; Lamport, L.: Distributed Snapshots: Determining Global States of Distributed Systems, ACM T.o.C.S., vol. 3, no. 1, pp. 63-75, 1985

[2] Dal Cin, M.; Hohl, W.; Michel, E.; Pataricza, A.: Error Detection Mechanisms for Massively Parallel Multiprocessors, Proc. Euromicro Workshop on Parallel and Distributed Processing, Gran Canaria, 27. - 29. Jan. 1993, pp. 401-408

[3] Dal Cin, M.: New Trends in Parallel and Reliable Computing: Massive Parallelism and Fault Tolerance. Invited paper, Proc. µP'92, 7th Symposium on Microcomputer and Microprocessor Applications., Budapest, April 1992, pp. 1-10

[4] Dal Cin, M.; Grygier, A.; Hessenauer, H.; Hildebrand, U.; Hönig, J.; Hohl, W.; Michel, E.; Pataricza, A.: Fault Tolerance in Distributed Shared Memory Multiprocessors, in: A. Bode, M. Dal Cin (eds.), Parallel Computer Architectures, pp. 31-48, Springer LNCS 732, 1993

[5] Grand Challenges: High Performance Computing and Communications. The Fiscal Year 1992 U.S. Research and Development Program. Report by the Committee on Physical, Mathematical, and Engineering Sciences, NSF Washington 1992

[6] Hildebrand, U.: A Fault Tolerant Interconnection Network for Memory-Coupled Multiprocessor Systems, In: Dal Cin, M.; Hohl, W.(eds.): Proc. 5th Int. Conf. Fault Tolerant Computing Systems, Informatik-Fachberichte 283, pp. 360-371, Springer 1991

[7] Hofmann, F.: Das Querschnittsprojekt "MEMSY", Ein Modulares Erweiterbares Multiprozessorsystem, Proc. Euro-Arch'93, pp. 567-577, Informatik-aktuell, Springer 1993

[8] Hofmann, F.; M. Dal Cin, A. Grygier, H. Hessenauer, U. Hildebrand, C.-U. Linster, T. Thiel, S. Turowski: MEMSY - A Modular Expandable Multiprocessor System, in A. Bode, M. Dal Cin (eds), Parallel Computer Architectures, pp.15-30, Springer LNCS 732, 1993

[9] Hohl, W.; Michel, E.; Pataricza, A.: Hardware Support for Error Detection in Multiprocessor Systems - A Case Study, Microprocessors and Microsystems, Vol. 17, No. 4, 1993, 201-206

[10] Kai Li; Naughton, J. F.; Plank, J. S.: Checkpointing Multicomputer Applications, Proc. 10th Symposium on Reliable Distributed Systems, pp. 2-12, 1991

[11] Koo, R.; Toueg, S.: Checkpointing and Rollback-Recovery for Distributed Systems, IEEE T.o.S.E., pp. 23-31, Jan. 1987

[12] Mahmood, A; McCluskey, E. J.: Concurrent Error Detection Using Watchdog Processors - A Survey, IEEE, T.o.C., Vol. 37, No. 2, pp. 160-174, 1988

[13] Michel, E.; Hohl, W.: Concurrent Error Detection Using Watchdog Processors in the Multiprocessor System MEMSY, Proc. 5th Int. Conf. Fault-Tolerant Computing Systems, Nürnberg, Informatik Fachberichte 283, pp. 54-64, Springer, September 1991

[14] Pataricza, A.; Majzik, I.; Hohl, W.; Hönig, J.: Watchdog Processors in Parallel Systems, Microprocessors and Microprogramming 39, 69-74, 1993

[15] Pataricza, A.; Majzik, I.; Dal Cin. M.; Hohl, W.; Hönig, J.: Fast Watchdog Processors for Multiprocessor Systems, submitted to the FTCS-24, Austin, 1994

[16] Russell, D. L.; Tiedeman, M. J.: Multiprocess Recovery Using Conversations, Proc. 9th FTCS, pp. 106-109, 1979

[17] Shrivastava, S.; Mancini, L.; Randell, B.: On The Duality Of Fault Tolerant System Structures. In: J. Nehmer (ed.), Experiences With Distributed Systems, Proc. Int. WS. Kaiserslautern 1987, pp. 10-37, Springer LNCS 309, 1988

[18] Siewiorek, D. P.: Faults And Their Manifestation, pp. 244-261, Springer LNCS 448, 1987