IEEE TENCON '93 / Beijing

Architecture and Operating System Design of the M^2 Database Machine *

Yen-Jen Oyang, David Jinsung Sheu,
Chih-Yuan Cheng, and Cheng-Zen Yang
Department of Computer Science
and Information Engineering
National Taiwan University
Taipei, Taiwan, China

Abstract

This paper discusses the architecture and operating system design of the M^2 Database Machine. The primary design goal of the M^2 is to provide a cost-effective solution to large database processing. The M^2 architecture is essentially a bus-based two-level hierarchical multiprocessor with a shared-memory multiprocessor structure at the lower level and a shared-disk structure at the higher level. The main distinction of the M^2 architecture is the high-bandwidth message-passing bus that forms the backbone of the second level, the higher level, of the hierarchy. The high bandwidth of the message-passing bus extends the scalability of the M^2 architecture to more than one hundred CPUs. Another distinction of the M^2 database machine is its system software. The M^2 incorporates a directory-base coherence protocol in its distributed file system. The directory-based coherence protocol allows efficient file sharing at the page or file block level.

1 Introduction

Parallel database machines are generally classified into three groups: shared memory, shared disk, and shared nothing [1]. Though shared-nothing machines attract most attention in today's research community [1,2], all three will eventually own a significant share in real-world database processing. It can be further expected that, as hardware technology continues to advance, the shared-disk machine will claim a larger share in the future. The main reason is that we now have been able to build shared-disk machines almost as cost-effectively as shared-memory machines and with a scalability of more than one hundred CPUs. Two factors contribute to this development. The first factor is the introducing of gigabyte message-passing buses such as Futurebus+ [3,4].

*This research was sponsored by National Science Council under grant NSC 82-0408-E-002-091

The second factor is the advances of VLSI and packaging technologies that make it possible to implement a shared-memory multiprocessor module in a printed circuit board. The introducing of gigabyte message-passing buses extends the scalability of a shared-disk machine to at least an order of magnitude higher than that of a shared-memory machine. Meanwhile, the high integration capacity made available by advanced VLSI and packaging technologies lowers the building costs of a shared-disk machine to the level of a shared-memory machine. Motivated by these observations, we started the M^2 database machine project.

In the following part of this paper, section 2 discusses the architecture design of the M^2. Section 3 elaborates the operating system development effort. Section 4 presents a prototype M^2 machine we have been building since Spring of 1991.

2 Architecture and Design Decisions

2.1 Overview of the M^2 Architecture

Figure 1 depicts the hardware block diagram of the M^2. The M^2 architecture comprises two levels of multiprocessing hierarchy. At the first level of the hierarchy, multiple CPUs, each with a private cache, and a shared cluster memory are placed on a printed-circuit board and connected through an on-board snooping bus to form a CPU cluster. In the M^2, the shared cluster memory in a CPU cluster serves as the main memory to the CPUs in the cluster. Therefore, structure-wise, there is no difference between a M^2 CPU cluster and a conventional shared-memory shared-bus multiprocessor.

The second level of the M^2 hierarchy is made up of multiple CPU clusters connected through a backplane message-passing bus. At this level of hierarchy, memory is distributed in both physical and logical senses.

That is, the memory of a cluster is accessible only to the cluster itself and is not accessible to other clusters. Communication between clusters is carried out through passing messages.

In the M^2, also connected to the backplane message-passing bus are I/O controllers. The I/O controllers and the CPU clusters operate under a client-server model. Some I/O controllers, e.g. disk controllers, are associated with a large memory which serves as the disk/file cache. In such cases, the memory in the CPU cluster and the memory associated with the I/O controller constitute a two-level disk/file cache. Data consistence between the memories in the CPU clusters and I/O controllers is maintained through executing a directory-based coherence protocol.

2.2 Architectural Features and Design Considerations

This subsection elaborates the main features and design considerations of the M^2 architecture. The M^2 is a two-level hierarchical multiprocessor. If compared with other hierarchical multiprocessors [5,6], the M^2 is distinctive in its memory organization at the second level of the hierarchy. In the M^2, a physically distributed with no remote access memory organization is employed. This design is aimed at avoiding severe page-swapping-induced inter-CPU interference. The page-swapping-induced inter-CPU interference is an inheritance of memory sharing among CPUs. For a group of CPUs that share memory, regardless of if the memory is physically distributed or not, each CPU must be notified with the page-swapping events occurring in the shared memory so that the CPU would write back the blocks that are cached in its private caches from the page being swapped out and update its TLB (Translation Lookaside Buffer) contents accordingly. Since the page-swapping-induced inter-CPU interference grows linearly with the number of CPUs that share memory, it is inappropriate to employ the shared-memory approach beyond a certain extent. Therefore, it was determined that a physically distributed with no remote access memory scheme should be employed at the second level of the M^2 hierarchy.

Nevertheless, the presence of the page-swapping-induced inter-CPU interference does not imply that a shared-memory design should not be used in any case. The shared-memory approach is still favorable up to certain extent due to its hardware simplicity and cost-effectiveness. This is the reason why VLSI chip sets that implement shared-memory shared-bus multiprocessors have become popular lately. Aiming to take advantage of this development, we decided to employ the shared-memory shared-bus structure at the first level of the M^2 hierarchy.

One important observation on the structure of the M^2 is that it is basically the same as a group of multiprocessors connected through a local area network. However, the M^2 is superior in system scalability since a backplane bus offers much higher communication bandwidth than a local area network. For example, a 256-bit Futurebus+ [3,4] can transfer up to 3.2 gigabytes, equivalent to 25.6 gigabits, of data per second. On the other hand, a FDDI network, as of today, can transfer 100 megabits of data per second and may be upgraded to 200 megabits per second in the near future, which is still two orders of magnitude lower than the bandwidth of the Futurebus+.

The last note on the M^2 architecture is that there is a natural match between the M^2 architecture and the architecture of the Mach operating system [7]. In the Mach, threads within a task are sharing-resource light-weight parallel entities. In the M^2, the CPU cluster, with multiple CPUs and a shared cluster memory, provides a good execution platform for multiple-thread tasks. At the higher level, Mach tasks, the heavy-weight parallel entities, can be dispatched to M^2 CPU clusters for parallel execution.

3 Operating System Development for the M^2

The M^2 will run the Mach operating system [7] with a restriction on process scheduling. On the M^2, Mach tasks are dispatched to CPU clusters in their entirety. In other words, the threads within a task are dispatched only to the CPUs of the cluster that the task is dispatched to and will not spread to other CPU clusters. One crucial issue in the system software design for the M^2 is how to maintain data consistence among the cached disk/file blocks in different CPU clusters. This issue is essentially the coherence problem in distributed file systems [8]. However, we decided not to adopt the mechanisms implemented in existing distributed file systems because the backplane message-passing bus in M^2, with its high bandwidth, can stand higher degree of data sharing among distributed disk/file caches than the local area network in a distributed system can. For this reason, we derived a new protocol from the directory-based cache coherence protocols proposed for large-scale multiprocessors [9]. The protocol used in the M^2 offers data sharing at the page or file block level and incorporates a distributed directory [9] with the ownership [10] and write-delayed mechanisms. In this protocol, each cached copy of a page is in one of the following 6 states:

1. INV (Invalid): The contents of this copy is invalid.
2. UNO (Unowned): The contents of this copy is valid but the CPU cluster is not the owner of this page.
3. OED (Owned exclusively and dirty): The CPU cluster owns a dirty copy of this page and no other CPU clusters has a duplicated copy of this page.
4. OEC (Owned exclusively and clean): The CPU cluster owns a clean copy of this page and no other

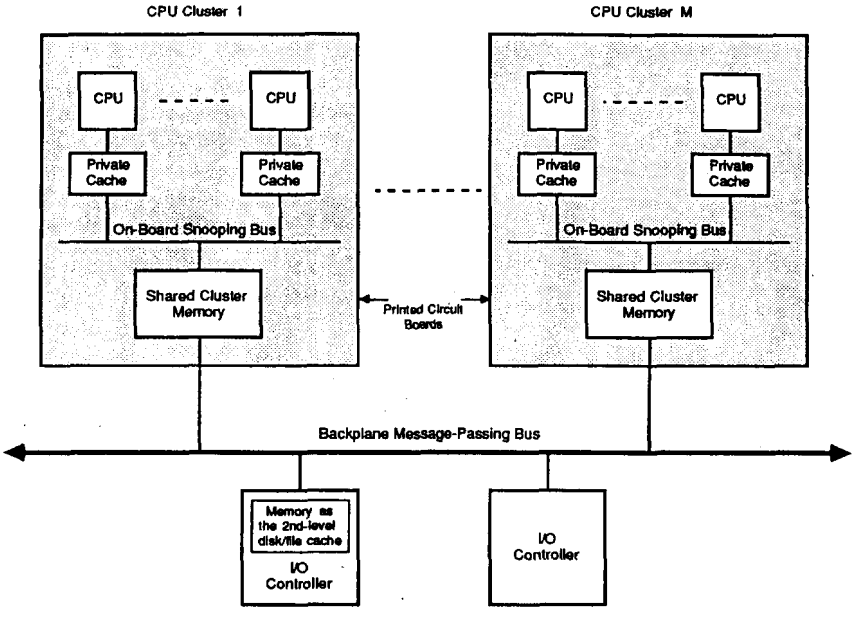

Figure 1: Hardware architecture of the M^2 database machine

CPU clusters has a duplicated copy of this page.

5. OND (Owned nonexclusively and dirty): The CPU cluster owns a dirty copy of this page and some other CPU clusters have a duplicated copy of this page.

6. ONC (Owned nonexclusively and clean): The CPU cluster owns a clean copy of this page and some other CPU clusters have a duplicated copy of this page.

The transition between the states is a response to the action taken by the CPU cluster. The possible actions that a CPU cluster may take when operating are listed in the following.

Read: When a CPU cluster intends to read the contents of a page that it does not has a copy, it sends a Read request to the I/O device that stores the page. If the I/O device determines that no CPU cluster owns the page at this moment, then it sends a copy of the page to the requesting CPU cluster and the requesting CPU cluster enters the OEC state. Otherwise, the I/O device will notify the CPU cluster that owns the page to send a copy to the requesting CPU cluster. In the later case, the requesting CPU cluster will enter the UNO state.

Read for Ownership: When a CPU cluster intends to update the contents of a page that it does not has a copy, it sends a Read for Ownership request to the I/O device that stores the page. The I/O device then notifies the CPU clusters that contain a cached copy of the page to invalidate their copies. If the page is currently owned by a CPU cluster, then the owner needs to send a copy of the page to the requesting CPU cluster as well as write back to the I/O device. If no CPU cluster owns the page, then the I/O device will presents the requesting CPU cluster with a copy. On completion of all these actions, the requesting CPU cluster enters the OED state.

Write for Invalidation: When a CPU cluster intends to update the contents of a page that it has a copy in UNO, ONC, or OND state, it sends a Write for Invalidation request to the I/O device that stores the page. The I/O device then notifies the CPU clusters that also contain a cached copy of the page to invalidate their copies. If the page is currently owned by a CPU cluster, then the owner needs to write back to the I/O device. On completion of all these actions, the requesting CPU cluster enters the OED state.

Write: When a CPU cluster intends to update the contents of a page it owns exclusively, it needs to change the state of the copy to the OED state if the copy is originally in the OEC state.

4 Development of a Prototype M^2

We have been building a prototype M^2 machine since the Spring of 1991. In the prototype machine, the Multibus II [11] is employed as the backplane message-passing bus and each CPU cluster comprises 2 Sparc CPUs along with a 64-megabyte cluster memory. The CPUs and the cluster memory are actually placed on two separate boards, the CPU board and the memory board. These two boards are connected through a local bus separate from the Multibus II. As of today, we have completed the hardware design of the CPU cluster and moved to operating system development.

5 Conclusion

In this paper, we elaborated the architecture and operating system design of the M^2 database machine. The primary design goal of the M^2 is to provide a cost-effective solution to large database processing. The M^2 machine is well characterized by the following two features:

1. A cost-effective hierarchical multiprocessor architecture that is scalable up to more than one hundred CPUs.
2. A directory-based coherence protocol for the distributed file system that maintains disk cache coherence at the page or disk/file block level.

References

[1] M. Abdelguerfi and A. K. Sood, "Database Machines: Trends and Opportunities", IEEE Micro, Vol. 11, No. 6, December, 1991.

[2] D. J. DeWitt and J. Gray, "Parallel Database Systems: The Future of Database Processing or a Passing Fad?", SIGMOD Record, Vol. 19, No. 4, December, 1990.

[3] IEEE, *IEEE Standard Backplane Bus Specification for Multiprocessor Architectures: Futurebus+*, IEEE Standard 896.2, 1991.

[4] IEEE, *IEEE Standard Backplane Bus Specification for Multiprocessor Architectures: Futurebus*, IEEE Standard 896.1, 1987.

[5] A. W. Wilson, "Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors", Proc. of the 14th Annual International Symposium on Computer Architecture, 1987.

[6] D. Cheriton, H. A. Goosen, and P. D. Boyle, "Paradigm: A Highly Scalable Shared-Memory Multicomputer Architecture", IEEE Computer, Feb., 1991.

[7] A. Tevanian Jr., "Architecture-Independent Virtual Memory Management for Parallel and Distributed Environments: The Mach Approach", Ph.D. Thesis , Dept. of Computer Science, Carnegie-Mellon University, 1987.

[8] E. Levy and A Silberschatz, "Distributed File Systems: Concepts and Examples", ACM Computing Survey, Vol. 22, No. 4, December, 1990.

[9] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-Based Cache Coherence in Large-Scale Multiprocessors", IEEE Computer, Vol. 23, No. 6, June, 1990.

[10] R. H. Katz, S. J. Eggers, D. A. Wood, C. L. Perkins, and R. G. Sheldon, "Implementing a Cache Consistency Protocol", Proc. of the 12th Annual International Symposium on Computer Architecture, 1985.

[11] Intel Corporation, *Multibus II Bus Architecture Specification Handbook*, Intel Corporation, 1984.