A Reflective Memory System for Personal Computers

Tomasevic, Milo; Protic, Jelica; Savic, Savo; Jovanovic, Milan; Grujic, Aleksandra; and Milutinovic, Veljko

Abstract— ***In the introductory part, this paper defines the general environment for this research and defines the terms of interest. Then we define the research problem: how to design a cost-effective board that connects a personal computer as a node in an RMS system and how to increase the efficiency of the basic RMS solution. Existing solutions to the problem are briefly surveyed, and their drawbacks are underlined. Then, the essence of the proposed solution is presented: designing a board that interfaces a personal computer to the RM bus. The fifth part defines conditions and assumptions of the analysis to follow. Elements of a simulation study are presented in part six. The conclusion, as a summary of new ideas, acquired during this project is presented in the part seven.***

1. INTRODUCTION

DISTRIBUTED shared memory (DSM) architecture represents a successful hybrid of shared memory multiprocessors (SMP) and distributed systems (DS) that inherits advantages of both architectures [1]. All processors in a DSM system access shared logical address space, which simplifies the programming model, and makes many applications portable from SMP systems. However, underlying architecture is typically a DS, so the logically shared memory is physically distributed, which means that, with an appropriate interconnection network, the system has good potential for scalability.

A DSM system is composed of nodes connected by hypercube, LAN, bus hierarchy, etc). Each node can be a single-processor system, or even an SMP (like in Dash system [2]). In both cases, it contains a portion of local memory mapped to the unique shared address space of DSM. In general, shared data stored in local memories can migrate from one node to another, depending on the access pattern of an application. It can also have multiple copies replicated in local memories of several nodes, which requires that all copies have to be kept up to date according to some consistency rules defined by memory consistency model (MCM).

There is a variety of algorithms for consistency maintenance [3], and the majority of them can be classified as multiple reader single writer (MRSW) and multiple reader multiple writer (MRMW). Consistency related information is stored in system tables or directory that can be centralized or distributed, corresponding to the responsibility for management of DSM system.

The meaning of the word “consistency” has been the most variable aspect of DSM systems from the early efforts in this area, in mid-80 till their today’s maturity. Memory consistency model (MCM) determines its meaning by defining the legal ordering of memory references issued by some processor, as observed by other processors in the system [4, 5]. While the most strong MCMs, sequential and processor consistency do not distinguish between ordinary shared data accesses and synchronization accesses, more sophisticated models like weak release, lazy release, and entry consistency take advantage of the fact that shared data accesses are typically protected by some synchronization operations. Therefore, the points of enter/exit of critical sections and barriers are used as suitable moments when some consistency-related actions should take place. The global trend in most sophisticated MCMs [6, 7] is to minimize the necessary communication between nodes, paid by making the programming model more complex. This trend distances DSM systems from their original goal (simplification of the programming model) but greatly improves their performance by reducing and/or hiding the communication latency.

The mechanism that provides the illusion of shared memory in a physically distributed memory system can be implemented in software, in hardware, or in cooperation of hardware and software, which results in a hybrid implementation. The first DSM systems were implemented on the underlying network of workstations, typically connected by Ethernet [8]. They were implemented in software, as runtime systems and/or modifications of the original operating system. Some of the systems developed more recently also included (or were primary based) on compiler modifications [6]. On the other hand, a class of DSM systems implements the DSM mechanism in hardware, which brings full transparency to all software layers, besides their performance advantages.

Manuscript received May, 2006. This work was supported in part by the Encore, HP.

Tomasevic Milo, Protic Jelica, Savic Savo, Jovanovic Milan, Grujic Aleksandra, and Milutinovic Veljko are with Faculty of Electrical Engineering, University of Belgrade.

Virtually the best potential for further improvements in the field of DSM is in combining software approach that can better explore the characteristics of the application, known by the programmer, with the hardware approach, that helps in performing critical operations more efficiently. Therefore, majority of the most recent DSM systems can be classified as hybrid implementations.

Reflective memory systems (RMS) [9] belong to the class of hardware implemented DSM. Their main characteristic is the non-demand, write-through update consistency mechanism. The majorities of DSM systems generally keep track of valid copies of shared data present in local memory of the node, and if data is not present in the valid state, fetch it from another node(s) on demand. Unlike them, an RMS performs all necessary updates automatically, using broadcast over the RM bus as an appropriate mechanism. Each node maps some segments in its local reflective memory to the shared address space, treating these segments as open "windows." Each write operation to an open window is immediately propagated, and received by all other nodes that also have opened windows containing that particular address. This method makes the write operation costly, but the read operation is not delayed, since the valid data is always present in the local reflective memory.

2. *Problem Statement*

Although the first RM systems [10] appeared about a decade ago, they suddenly gained emerging popularity in the last few years. In the mean time, many variants of demand-based DSM systems were built, proving the validity of the concept, but few of them became commercial products. The reason for that can be found in the high overhead of operating system and the considerable time consumption of the layered protocol software. In addition, communication latency in such a system is hard to predict, which is not tolerable for real-time applications.

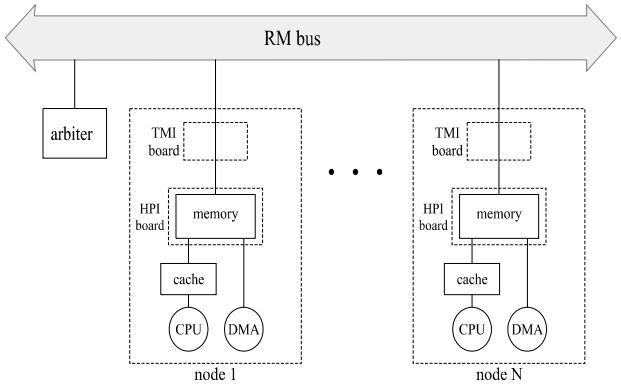

Figure 1: Architecture of the reflective memory system (RMS).
Legend: CPU - central processor unit; DMA - direct memory access unit.

All these problems are avoided in RM systems, with their simple and efficient hardware mechanisms for consistency maintenance. The communication in these systems is usually overlapped with computation, and the shared memory access time is predictable and relatively low.

The earlier RMS systems were usually implemented in minicomputer environment, and used for a wide range of applications, from real-time to on-line transactions processing. However, in early 90's the expansion of DSM concept to a NOW (Network of Workstations) and NOPC (Network of Personal Computers) became one of the most important development trends in this field [11]. Encore was one of the first who understood this trend, and concluded that using a standard and relatively powerful machine such as PC, as a node in the RM system, could lead to a cost-effective system with high configuration flexibility. That was the motivation to start this project, with prospective to build a system which costs less, connects more nodes and supports bigger traffic.

The first technical goal of the project was the design of a fully operational RMS board, which plugs into a PC, and turns it into a node within the bus-based DSM system, using reflective memory approach. The board had to be compatible with the RM bus interconnect (Encore's proprietary bus), and also had to provide a larger amount of local memory.

Besides the strictly technical goal of designing the board, Encore was also interested in some theoretical research, which would go beyond the currently operational systems and result in the development of the know-how, which could create new break-through in the near future. The requests for this part of the project were not so strict: while the management stuff of our research sponsor insisted on the solution for 64 or 128-node RM system, engineering stuff was skeptic and suggested the generation and elaboration of several ideas that would probably improve the performances of the system, and their detailed simulation analysis. The essence of this strategy was to achieve a relatively large global performance improvement through a series of small incremental improvements. This was probably the only way to go, since the RM concept existed for years, and a considerable advance could be obtained only through a superposition of relatively small advances in different directions.

3. *Existing Solutions*

There are multiple types of systems based on non-demand write-through update, some of them referred to as mirror memory systems [12] or replicated memory systems [13]. However, the term reflective memory is the most frequently used. This class of systems includes some commercial products (Modcomp [14], Systran

[15], and VMIC [16]) targeted to the real-time market, as well as ATC's ring-based approach [17] and DEC's Memory Channel [18]. The SHRIMP project of the Princeton University [19] also belongs to this class. An almost exhaustive survey [20] covers the majority of RMS systems. The first RM system was made and patented by Gould Electronics in 1985. Encore acquired Gould in 1989, and updated the system, which later evolved into a number of RMS variants. The latest developments resulted in passing the technology from Encore to DEC corporation, and its promotion of DEC Memory Channel, which is now Compaq. This paper focuses on Encore's development of RMS.

In the RMS of [21] each node connected to the RM bus consists of a processor with cache, local memory, DMA device, and the RM board with memory and interface to the RM bus (see Figure 1). Bus arbitration is centralized, using a round-robin synchronous arbitration algorithm. Processor (responsible for word transfers) is connected to the RM board via the local processor/memory and/or the host system bus. The DMA unit (responsible for block transfers) is attached via the host system bus. Transmit/receive mapping tables are used to configure local memory pages as reflective (shared) or private (non-shared), as well as for address mapping, since copies of a shared page could be at different addresses on different nodes. On each write to the local RM memory, a check is made in the transmit mapping table if the address belongs to an open window, and, if so, data word together with the translated address in reflected shared space is put into the transmit FIFO buffer. The node with a pending write requests the RM bus from the RM bus controller. When the node obtains the bus grant, bus write operation (address + data) is initiated on the RM bus. All other nodes check if the address falls into a local open receive window. In that case, the data and translated receive address are written into the receive FIFO. Finally, data from the receive FIFO are forwarded to the local RM memory.

The state-of-the-art among RM systems is RM/MC (Reflective Memory/Memory Channel) which supports blocked updates. The RM/MC system is based on the multiplexed synchronous 64 bits wide bus, connecting up to eight processing nodes, with the 15000 feet maximal distance (for a fiber optic cable). RM/MC supports both single word and block update transfers. Each node in the system is composed of two boards: TMI (Transition Module Interface) for the RM/MC bus and HPI (Host Port Interface) for the host system bus. There are implementations for the VME bus, the EISA bus, and the PCI bus.

Since RMS and similar systems are based on asynchronous, non-instantaneous communication, they typically can not guarantee strict consistency. However, if all writes are seen by all processors in the system in the same order (which is true for centralized arbitration and no bypassing in transmit/receive FIFO buffers) sequential consistency can be achieved. More relaxed consistency models can be implemented if we allow some bypassing and/or deleting in transmit/receive FIFO buffers. However, the latest developments in this area introduced some hybrid solutions [22]. Encore proposed LAM - a software-based consistency maintenance for RM/MC underlying hardware. The LAM uses library routines to manage allocation of the RM space and to synchronize the accesses to this space, providing the programmer with the means of entry consistency-like MCM.

4. Proposed Solution and Its Advances

The first goal of the project was to design a board that interfaces a personal computer to the RM bus. Since the interface between the local RM memory and the RM bus is similar for different computers, it is moved into a separate hardware module called TMI (Transition Module Interface). The TMI is connected with the host RM board by a special cable with multiplexed address and data lines.

Host board (Figure 2) contains the system bus interface, local RM memory, and memory arbiter and control logic. The memory on the host board and all data paths are 32-bits wide. There are no FIFO buffers between the memory and the system bus, since the on-board memory cycle is synchronized with the system bus cycle. The system bus burst cycles are also supported. The board is designed to have 16MB or 64MB of DRAM organized in four standard 72-pin 36-bit SIMMs. The system base address of the memory is programmable in 16MB/64MB quantum. Memory arbiter gives the highest priority to the DRAM refresh logic, then to the system bus interface, and at last to the receive FIFO (data coming from the RM bus). The priority can be changed when the receive FIFO is almost full, or on the system bus time-out.

The host board also contains transmit translation SRAM. This mapping table keeps the shared/private indication bit for each 8K-segment of the memory, and a global RMS address where a block should be mapped if it is shared. From the memory side, address and data FIFOs are accessed by separate buses, while on the host-TMI cable side address and data lines are multiplexed to reduce the bulk of cabling. The TMI demultiplexes the address and data lines, puts the address and data into separate FIFOs, and connects them to the RM bus. The receive translation SRAM is placed on the TMI to keep the information about mapping from the RMS address space to the local RM memory. If an address belongs to a block (which is reflected),

data and the translated address are written into the receive FIFOs on the TMI board, then multiplexed, sent to the FIFOs on the host board, and finally written to the memory.

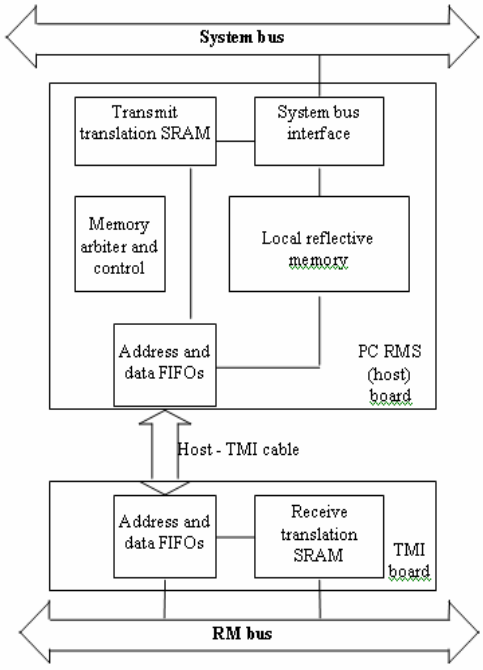

Figure 2: Implementation scheme of the PC-RMS. Legend: TMI - Transition Module Interface.

5. Conditions and Assumptions of the Research to Follow

After the design of the PC-RMS board was finished, it was carefully analyzed, in order to notice the drawbacks of the RM concept and its implementation, and then, based on its criticism, to propose and elaborate several improvements [23]. Each proposed solution was separately simulated, in order to examine its potential effects, and evaluate its impact in comparison with other proposals. After that, a specific decision-making methodology (modified for DSM) [e.g., 24] was applied, in order to choose a subset of proposals that will give the solution with a relatively large global performance improvement, achieved through a series of small incremental improvements. However, the final decision about the incorporation of proposed improvements into the final product was left to the research sponsor; our task was only to give recommendations, having in mind results of particular simulations. Therefore, simulation of the overall system, implementing all proposed improvements was not done, having in mind time limitations, and the fact that overall effects will be measurable on the implemented system.

6. Simulational Analysis

Here we present four suggested improvements that we found the most interesting:

Improvement #1: Caching of shared RM regions - in the existing RM systems, the shared reflected regions of the RM memory are not cached. RM board appears as a memory slave on the host system bus, and can not initiate any action on this bus when a write from the RM bus occurs. Since the effects of caching to the overall system performance are well known [25], we suggested some changes that make caching possible in RM systems. Each block in the RM memory should be associated one bit, which denotes its caching status (C-bit). When a block is cached, its C-bit should be set. After that, when a write from the RM bus to a shared location comes, the C-bit will be checked, and if it is set, invalidation will be sent to that cache, and the C-bit will be reset. Only one additional bit per block, together with the very simple comparator logic represents reasonable complexity paid for significant performance gain (Figure 3a). The simulation shows that caching of the shared memory segments results in the improvement of about 10% for the CPU utilization. The improvement is better for smaller node counts, since for the larger node counts, the contention on the RM bus caused by multiple invalidations becomes significant. The benefits of caching should have been even more pronounced if the system follows write-back policy.

Improvement #2: Double RM bus - since the shared bus represents notorious bottleneck in bus-based systems, the natural solution for the restricted scalability is to employ multiple busses. We have analyzed the solution with a double RM bus. When the need for accessing the RM bus arises, the bus request is issued on both buses at the same time, yet only one bus will be granted that transfer (whichever grant comes first). The proposed solution can be efficient both in cases of transient and permanent bus overloads, in terms of traffic of writes to the shared memory regions. The overall effect is that the average time to obtain the bus is decreased, which results in the higher system throughput. This also offers a potential for increased system reliability. However, the cost of this approach is significant because of the introduction of the second bus, and the hardware complexity of the bus interface logic is increased. This complexity increase can be justified only by a satisfactory performance improvement (Figure 3b). While in the conditions of saturated bus traffic in the single-bus RM system average waiting time for the bus grant linearly increases, in the double bus system this time is insignificant and practically insensitive to the system size.

Improvement #3: Improved RM bus arbitration The RM bus controller assumes equal priority for all requests coming from each node. However, the overload of the nodes can be different, so some FIFOs can be full, while FIFOs in other nodes are empty. The solution for this problem is to give a higher priority to the overloaded node, from the moment it reaches the “almost full” condition. When the number of items held in the transmit FIFO buffer reaches certain threshold,

the "urgent need" for the RM bus will be signaled. Then, the RM bus arbiter temporarily departs from the round-robin policy, and gives the upper hand to that node. This priority will be assigned temporarily (only for one transfer on the RM bus). When no "urgent need" signals are asserted, the arbiter is back to the round-robin arbitration. For the purpose of announcing the priority request, bus lines reserved for the node ID can be used, since they are free during the RM bus transfer. This approach introduces no additional bus lines, but requires very careful timing. The simulation shows that in conditions of the increased bus traffic, this method improves processor utilization, but only for a lower number of nodes, while in the hypothetical systems with 12 or 16 nodes it has no advantages (see Figure 3c).

*Improvement #4:*Write Filtering - the essence of this improvement is the filtering of subsequent write requests issued to the same address by the same node, while these requests are sitting in the transmit FIFO of the writer. This method helps in reducing the traffic on the RM bus. According to the number of previous writes in the FIFO that should be compared to the issued write operation, we have proposed three solutions: (a) variant “all” - comparison with the entire contents of the transmit FIFO, (b) variant “last 4” - comparison with the last four writes, and (c) variant “last 1” - comparison with the previous write only. Performance improvements for all these variants are given in Figure 3d.

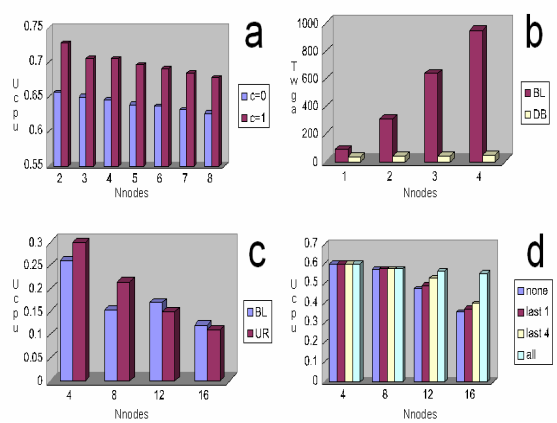

Figure 3: Performance results for four proposed improvements.

Legend: Ucpu - processor utilization; Nnodes - number of the RM nodes; Twga - average waiting time for the RM bus grant; C=0 - without caching; C=1 - with caching; BL - baseline system with single bus and round-robin policy; DB - double bus system; UR - urgent request arbitration.

Description: Figure 3a shows the comparison of processor utilization with and without caching. Figure 3b shows the comparison of average waiting time for the bus grant in a single (BL) and double (DB) bus RM systems. Figure 3c shows comparison of processor utilization for a system with the round-robin policy (BL) and the dynamic priority policy with urgent request arbitration (UR). Figure 3d shows the impact of four write-filtering variants on the processor utilization.

The best effects of this solution can be expected for applications with high processor locality of memory references (long write runs). When applying this improvement, one must be careful about the consistency model assumed by the programmer, since the loss of some write operations is allowed only for relaxed consistency model.

Simulations were performed for no write filtering, and for variants “last 1”, “last 4” and “all”. Processor utilization is almost the same for all four variants in the systems with less then 8 nodes; however, in systems with larger node counts, the difference becomes significant, in favor of write filtering applied to the longer sequences in the transmit FIFO buffer.

7. CONCLUSION

Fortunately, hard work always leads to results. While exploring the possible enhancements of RM concept, in order to achieve the theoretical goal of the project, after the incubation period has passed, we come up with some general ideas, applicable in the wider area of DSM systems, as well as some solutions in the field of cache consistency in multiprocessor systems. Even two years after the project was finalized, these ideas were applicable in other projects, and resulted in several thesis and research papers, summarized in [26]. The most important new concept was the STS idea, which suggests the separation of data that expresses predominantly spatial locality from the data that expresses predominantly temporal locality, and different treatment of these two data spaces [26]. The continued efforts inspired by the write filtering idea presented here resulted in even more sophisticated improvements of the RM system [27], while the application of some concepts found in RM to the cached data in multiprocessor systems resulted in a new solution for lazy prefetching [28].

Finally, was it worth the effort? Instead of the research paper published in a respected journal (which is probably what every researcher dreams of when starting a new project), we ended up with a working board (to the best of our knowledge, the first implemented multiprocessor system with reflective memory DSM mechanism, using the PC-based computer nodes) and a lot of experience and accumulated knowledge, that resulted in a mature view of the field, presented in [MILU2000]. The detailed survey of DSM concepts and systems that we did, inspired by this project, and the great collection of papers we read, did not finish in our archive; instead, we also published a tutorial book with the state-of-the art works in the field of DSM [29].

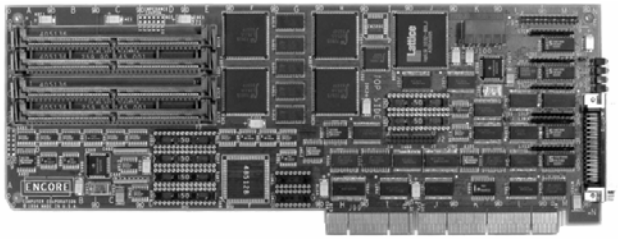

Figure 4: The photo of the PC-RMS board

This project was a pioneer effort from two points of view: it was organized in cooperation between a university and an industrial company located on two distant continents before the concept of globalization and virtual project teams became commonplace, and it resulted in the first commercial DSM system based on the network of PCs. Projects like this bring great benefits to the industry, allowing efficient outsourcing of some research and development activities to the appropriate research centers. Thanks to the information superhighway, geographical limitations no longer apply when choosing the most suitable partners. On the other hand, university researchers get the opportunity to cope with the real world problems, and to incorporate their knowledge into the technology development. However, the obstacles to the process of globalization and sharing the results with the research community are industrial secrets and proprietary limitations, which slow down the free flow of information. The avenue we opened with this project was proved to be the prospective one: in the years after we have completed our research, several other approaches in the field of DSM for PC were initiated and proved to be successful.

REFERENCES

[1] Protić, J., Tomašević, M., Milutinović, V., "Distributed Shared Memory: Concepts and Systems," *IEEE Parallel and Distributed Technology*, Summer 1996, pp. 63-79.

[2] Lenoski, D., Laudon, J., et al., "The Stanford DASH Multiprocessor," *IEEE Computer*, Vol. 25, No. 3, March 1992, pp. 63-79.

[3] Stumm, M., Zhou, S., "Algorithms Implementing Distributed Shared Memory," *IEEE Computer*, Vol. 23, No. 5, May 1990, pp. 54-64.

[4] Gharachorloo, K., et al., " Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," *Proceedings of the 17th International Symposium on Computer Architecture*, 1990, pp. 15-26.

[5] Protić, J., Tartalja, I., Tomašević, M., "Memory Consistency Models for Shared Memory Multiprocessors and DSM Systems," *Proceedings of the 8th Mediterranean Electrotechnical Conference Melecon '96,* Bari, Italy, May 1996, pp. 1112-1115.

[6] Bershad, B., N, Zekauskas M., J., Sawdon, W., A., "The Midway Distributed Shared Memory System," *Proceedings of the OMPCON 93,* February 1993, pp. 528-537.

[7] Keleher, P., Cox, A., L., Dwarkadas, S., Zwaenepoel, W., "TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems," *Proceedings of the USENIX Winter 1994 Conference,* January 1994, pp. 115-132.

[8] Fleisch, B., Popek, G., "Mirage: A Coherent Distributed Shared Memory Design," *Proceedings of the 14th ACM Symposium on Operating System Principles,* ACM, New York, 1989, pp. 211-223.

[9] Lucci, S., Gertner, I., Gupta, A., Hedge, U., "Reflective-Memory Multiprocessor," *Proceedings of the 28th IEEE/ACM Hawaii International Conference on System Sciences*, Maui, Hawaii, USA, January 1995, pp. 85-94.

[10] "RMS Functional Specification," *Encore Computer Corporation*, Fort Lauderdale, Florida, USA, 1990.

[11] Li, K., "Development Trends in Distributed Shared Memory," *Panel of the 22th International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, June 1995.

[12] Maples, C., Wittie, L., "Merlin: A Superglue for Multicomputer Systems," *COMPCON'90*, March 1990, pp.73-81.

[13] Oguchi, M., Aida, H., Saito T., "A Proposal for a DSM Architecture Suitable for a Widely Distributed Environment and its Evaluation," *Proceeding of the Fourth IEEE International Symposium on High Performance Distributed Computing* (HPDC-4), Washington, D.C., USA, August 1995, pp.32-39.

[14] Furht, B., "Architecture and Performance Evaluation of the MMM," *IEEE TCCA Newsletter*, March 1997, pp. 66-75.

[15] "SCRAMNet+ Protocol Description," *SYSTRAN Corporation*, Dayton, Ohio, USA, 1995.

[16] "VMIC's Reflective Memory Network," *VME Microsystems International Corporation,* Huntsville, Alabama, USA, 1995.

[17] Ramanjuan, R., Bonney, J., Thurber K., "Network Shared Memory: A New Approach for Clustering Workstations for Parallel Processing," *Proceedingsof the Fourth IEEE International Symposium on High Performance Distributed Computing* (HPDC-4), Washington, D.C., USA, August 1995, pp. 48-56.

[18] Gillett, R., "Memory Channel Network for PCI," *IEEE Micro*, February 1996, Vol. 16, No. 1, pp. 12-18.

[19] Blumrich, M., et al., "Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer," *Procedings of the 21th Annual International Symposium on Computer Architecture*, IEEE Computer Society Press, Los Alamitos, California, USA, 1994, pp. 142-153.

[20] Jovanović, M., Milutinović, V., "An Overview of Reflective Memory Systems," *IEEE Concurrency*, Vol. 7, No. 2, Spring 1999.

[21] Gertner, I., "The Reflective Memory / Memory Channel System Overview*,* " Technical Report, *Encore Computer Systems,* Fort Lauderdale, Florida, USA, 1993.

[22] Denton, R., Johnson, T., "Distributed Shared Memory Using Reflective Memory: The LAM System, " Technical Report (TR96-021), *University of Florida*, Gainesville, Florida, USA, July1996.

[23] Savić, S., Tomašević, M., Milutinović, V., "Improved RMS for the PC Environment," *Microprocessor Systems*, Vol. 19, No. 10, December 1995, pp. 609-619.

[24] Hoevel, L., Milutinović, V., "Terminology Risks with the RISC Concept in the Risky RISC Arena," *IEEE Computer*, December 1991, pp. 136.

[25] Tomašević, M., Milutinović, V., (editors) "Cache Coherence Problem in Shared Memory Multiprocessors: Hardware Solutions, " *IEEE Computer Society Press*, Los Alamitos, California, USA,1993.

[26] Milutinović, V., "Some Solutions for Critical Problems in The Theory and Practice of DSM," *IEEE TCCA Newsletter*, September 1996, pp. 7-12.

[27] Jovanović, M., Tomašević, M., Milutinović, V., "A Simulation-Based Comparison of Two Reflective Memory Approaches," *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS-95),* IEEE Computer Society Press, Los Alamitos, California, USA, 1995, pp. 140-149.

[28] Milenković, A., Milutinović, V., "Lazy Prefetching," *Proceedings of the HICSS-98*, Mauna Lani, Hawaii, USA, January 1998.

[29] Protić, J., Tomašević, M., Milutinović, V., (editors) "Distributed Shared Memory: Concepts and Systems," *IEEE Computer Society Press*, Los Alamitos, California, 1998.