An Efficient Processor-Network Interface for Local Area Multiprocessors

Evangelos Markatos, Manolis Katevenis, George Kalokerinos, and Dimitrios Serpanos
Computer Architecture and VLSI Systems Group
Institute of Computer Science (ICS)
Foundation for Research & Technology – Hellas (FORTH)
Vassilika Vouton, P.O. Box 1385 GR 711 10 Heraklion, Crete, Greece
markatos@csi.forth.gr tel:+30 (81) 39 16 55, fax:+30 (81) 39 16 71

In Proceedings of the SCIzzL-4 Workshop on Local-Area MultiProcessors
Heraklion, Crete, October 1995

1 Introduction

Most computing environments today consist of a number of workstations or personal computers (PCs)* connected via a (high-speed) interconnection network. These environments are usually called workstation clusters, while acronyms like COWs (Clusters of Workstations) and NOWs (Networks of Workstations) are popular as well. Although workstation clusters have the aggregate processing, memory, and I/O capacity to execute high-performance applications, they usually lack the required hardware support that allows them to present a high-performance communication interface to user applications. Network interfaces and communication protocols designed for slow and untrusted networks, impose a number of overheads in both message passing and shared memory applications, resulting in loss of a large amount of bandwidth at the application, although the underlying network may provide significant bandwidth [14]. The main sources of these overheads include:

- *Operating System Intervention*: Typically, in message passing systems, applications have to suffer a system-call overhead each time they want to send or receive a message.
- *Message Copying*: On its way from the sender to the receiver, a message is usually copied several times from one address space to another, esp. in microkernels.
- *Message Reassembly*: Large messages are usually broken into smaller ones, which get interleaved with messages from various senders, and arrive at their destination in random order, where they have to be reassembled in their proper order to form a new message.

Despite the above disadvantages, traditional network interfaces and communication protocols were successful in the past because the message transfer time on the wire was the dominant percentage to the total message passing time. Thus, the software-imposed overhead just described contributed only a small percentage to the overall latency. Nowadays, the increasing network bandwidth makes the time spend on the communication medium a small percentage of the message passing overhead, thereby exacerbating the software-imposed overhead.

To reduce the mentioned overheads, we have designed and implemented *Telegraphos*

E. Markatos, M. Katevenis, and D. Serpanos are also with the University of Crete. D. Serpanos is also with IBM Research Division, T.J. Watson Research Ctr.

*In the rest of the paper we use the term workstation for both workstations and high-speed PCs.

[9, 10], † an efficient processor-network interface for workstation clusters which supports both message-passing and shared memory, and provides a high performance platform for efficient parallel processing.

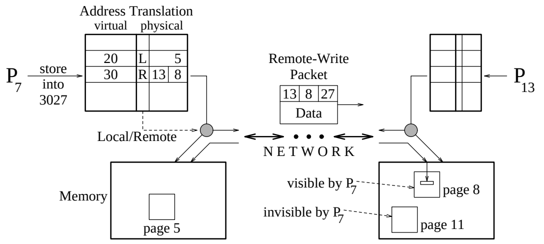

Figure 1: **Remote Write Operation in the Telegraphos Network Interface**

2 Telegraphos

A Telegraphos system is composed of a number of workstations (nodes) interconnected through a high speed network. The system implements a single-address space multiprocessor on top of the workstation cluster. Data replication/movement among systems is achieved through remote memory operations implemented with transfers of short, fixed size messages in the order of a few bytes; the network guarantees delivery of the messages in order. So, due to its architecture, Telegraphos supports both message-passing and shared-memory programming models. Its characteristics contributing to its efficiency are:

- *hardware support of non-coherent shared memory* through efficient implementation of remote read, write and atomic operations;
- *mapping the receiver's memory address space directly in the sender's address space*, so that messages go directly to their final destination, avoiding expensive copying and page remapping operations;
- *use of standard virtual-memory protection mechanisms* to avoid expensive OS calls on message transmission and reception;
- *sender-based buffer management* to avoid buffer overflows and throughput collapse;
- *page-level access counters* that can guide the page replication/migration decisions of the operating system for distributed shared memory systems [12], and network memory systems [13]. This is important, as in most distributed systems (including Telegraphos) expensive operations like cache (memory) coherence are left to software implementation at the OS or application level, which usually can achieve comparable results with hardware implemented cache coherence [6];
- *provision of special operations*, such as remote fetch-and-add, DMA, non-blocking read, etc., which are initiated from user space without OS intervention, in contrast to previous systems [15]. Standard virtual memory protection mechanisms prevent applications from performing special operations on memory addresses they are not allowed to.

All above properties are essential to providing an easy-to-use, low-latency, high-bandwidth substrate for parallel and distributed applications, ranging from scientific computations to multimedia applications.

2.1 The Remote Write Operation

Figure 1 illustrates the remote write operation. In a single address space system, each physical address refers to a particular word in the local memory of *one* particular processor (computer) on the network; in other words, the physical address space is shared among the processors. Thus, any virtual address generated by any processor on the network may refer, after address translation, either to a word in the local processor's memory or to a word in any other (remote) processor's memory. This is accomplished by treating the MS bits of every physical page number as a processor identifier, and the LS bits as a page number in that

†We call this project Telegraphos or $T\eta\lambda\acute{\epsilon}\gamma\rho\alpha\phi o\varsigma$ from the greek words $T\eta\lambda\acute{\epsilon}$ meaning remote, and $\gamma\rho\acute{\alpha}\phi\omega$ meaning write, because the *central* operation on Telegraphos is the remote write operation.

processor's memory. From the implementation point of view, when the virtual address of a store instruction translates into a physical address in the Telegraphos device I/O space, the Telegraphos network interface places the address and the data into a packet, and sends this packet through the network to the destination identified by the MS bits of the physical address. At the destination node, the Telegraphos interface performs the (local) memory write. Figure 1 illustrates an example of this procedure. Processor P_7 issues a store instruction on virtual address 3027. Virtual page number 30 is translated into physical page number 138 which denotes the physical page 8 that resides on processor P_{13}. The store operation to physical page 138 is directed to the Telegraphos interface, which prepares a network packet and sends it to node P_{13}, where the store to the actual memory location will be eventually performed. To reduce the latency of remote write operations, Telegraphos acknowledges the remote write operations as soon as it latches the data to be transferred to the remote processor. Thus does not wait for the data to arrive at their destination before it can proceed.

Remote read operations are similar to remote write operations, with the difference, that they wait for the data to arrive from the remote processor. Thus, remote read operations are slow, costing at least as much as one network round-trip delay. Both remote read and remote write operations are issued using a single load or store instruction, thus providing the end users the abstraction of a single-address space multiprocessor.

2.2 Special Operations

Telegraphos provides several operations (besides remote read and write), like DMA, remote atomic operations, non-blocking read, etc. We call these operations *special operations* because they are launched using more than one instructions, as current processors do not provide single hardware instructions for these (and other) special operations. All implementations of such special operations should meet the following criteria:

- Applications should be protected from each other when launching a special operation.
- Applications should be allowed to perform special operation only in memory locations they are allowed to read or write.

The simplest way of satisfying the above conditions is to provide system calls that will implement the special operations. Although system calls provide the necessary protection, they impose an unnecessary overhead to user applications which can be quite high if special operations are frequently used. Furthermore, system calls may require modification to operating system sources, which may not always be available.

To overcome the problems associated with operating system-level implementation of special operations, we use the notions of *Telegraphos contexts* and *shadow addressing* [2]. Each application is given at least one Telegraphos context which consists of registers that hold the arguments to the special operations. These contexts are mapped in the virtual address space of applications, so that an application will trap if it attempts to access a Telegraphos context it is not allowed to. Applications that want to start a special operation write the arguments in their Telegraphos contexts and complete the special operation with an access to a special register. The idea of hardware contexts allows applications to launch special operations from user space in a secure and efficient way: No application is allowed to tamper with another application's special operations; moreover, if an application gets interrupted while launching a special operation, the Telegraphos registers preserve their contents, so that the special operation will be launched when the application is resumed. ‡

Malicious users, however, may attempt to bypass hardware protection and initiate special operations in addresses they are not al-

‡In early versions of Telegrpahos we provided only one context. Applications that wanted to launch special operations had to execute without interruption. To achieve uninterrupted execution in user space we used the PAL mode of execution that the Alpha processor provides [18].

lowed to access in any other way. For example, they may try to store a physical address (that they normally have no access right to) in a Telegraphos register and ask for a special operation in it, effectively modifying the contents of that physical address. To avoid this improper use of Telegraphos we build on the notion of shadow addressing. For each virtual address that maps into a physical address, we introduce a shadow virtual address that maps into a shadow physical address. An address differs from its shadow only in the highest bit. When a user application wants to pass a physical address to Telegraphos to be used as an argument to a special operation, it makes a store to its corresponding shadow virtual address. Telegraphos catches this store operation, gets the physical address, strips the highest order bit, and uses the remaining address as an argument to a special operation. The argument of the store instruction contains the identification of the Telegraphos context where the physical address is to be placed, along with a key that verifies that the process issuing the store instruction is allowed to use this Telegraphos context. This combination of Telegraphos contexts, keys, and shadow addressing, albeit a little complicated, it manages to translate a virtual address to its corresponding physical one, and pass it to the network interface in a secure way, all in one instruction issued from user-level!

Special operations include atomic operations, DMA requests, and remote fetch operations. The remote *fetch* operation is like a remote read, but the processor does not wait for the returned value, which is written in a memory location and not a register; this value is returned into the memory of the requesting host, for later use by the processor. Normally, the fetch operation is used in order to prefetch data into a local page of the requesting processor. Thus, it is expected that the return host will be the same as the issuing host, and that the return address will point to a local page. The fetch operation does *not* include in itself any explicit test for its completion (any explicit method for the issuing processor to know when the return data have arrived). However, there are a number of indirect tests that are possible. One simple method is to know (by programming means) that the value to be returned has some particular property (e.g. non-zero, non-negative, etc.), and to initialize the return location to a value that does not have that property before issuing the fetch operation (e.g. initialize to 0, -1, etc.); then, completion of the fetch can be tested for by looking for a value with the desired property at the return location. The other methods deal with *collective* tests, i.e. tests for the completion of a number of remote fetches, but their detailed description is beyond the scope of this paper.

2.3 Page Access Counters

To aid the operating system in making various page replication/migration decisions, Telegraphos provides reference counters for each page. These counters can be used to count local and remote accesses to each page. Thus, the operating system is able to identify reference patterns for each page and find the frequently used pages. There are two counters for each remote page in the system and four counters for each local page. The two former are used to count the numbers of remote read requests, and the numbers of remote write/atomicOp requests, respectively, originating from this host. The four latter counters do the same for local accesses, but they count separately local accesses from the host and local accesses coming from the network. All counters count *down*; when a counter is decremented from 1 to 0, the host is interrupted. This is a method for the operating system to make certain page placement decisions, and then to be later invoked by means of an interrupt if these decisions lead to a number of accesses of certain types that exceeded a certain preset threshold. No interrupt is generated, and the counter value is not changed when the counter was already at zero; this is a method to selectively disable some counters from ever causing interrupts.

Simulation results suggest that these counters can be used to improve performance in distributed shared memory systems [12], and in network memory systems [13]. In distributed

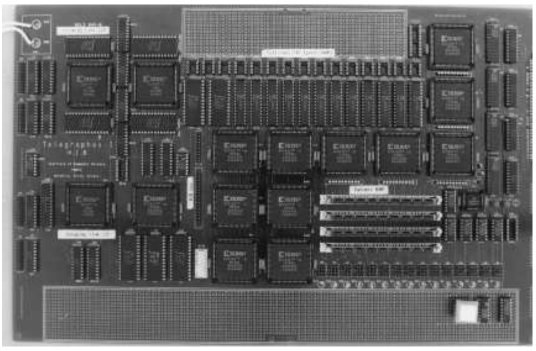

Figure 2: **Photograph of Telegraphos-I Network Interface**

shared memory systems they are used by the operating system to determine which pages to replicate/migrate close to which processor. The same counters can be used in network memory systems that use remote memory for paging, file system caching, etc. Such systems need to make sophisticated decisions regarding which pages they are going to keep in local memory, which pages are going to keep in remote memory, and which pages are going to keep in disk, in order to improve performance and avoid thrashing.

3 Telegraphos-I

Telegraphos I has been implemented using DEC Alpha 3000 model 300 workstations as nodes and a custom-made interconnection transferring 9-byte messages over links operating at 103 Mbps. Interfaces are attached to the workstations' TurboChannel I/O bus, and allow any attached workstation to map memory of any other workstation in the same cluster. The goal of this implementation is fast development of a cluster, mostly for experimentation with parallel and distributed applications. For this reason, Telegraphos I has been implemented using rapid-prototyping methods like FPGAs (see figure 2).

3.1 Performance Measurements

Although Telegraphos I is still being debugged, it is stable enough to run simple experiments that measure the performance of its basic operations: remote read, and remote write.

Our experimental hardware consists of two DEC 3000 model 300 workstations connected with the Telegraphos Network. We started one application on one workstation that makes both local and remote read and write accesses to the Telegraphos shared memory. Remote read and write accesses look just like ordinary load and store operations. Their only difference with local accesses from the programmer's point of view is that local accesses are faster than remote accesses. After starting the application, we measured the latency of local and remote read and write operations by performing 10000 operations. Our measured results are:

Operation	Elapsed Time per operation (μsec)
Local Read	0.87
Local Write	0.46
Remote Read	7.2
Remote Write	0.7

We see that remote write operations are very efficient: they take less than a microsecond! The reason is that Telegraphos acknowledges a write operation as soon as its is written onto the local HIB. Thus, applications that want to send small messages can do that very efficiently. Short batches of write operations execute even faster! For example, a stream of 100 remote write operations takes less than 50 μsec, thus each of the remote write operations takes less than 0.5 μsec. The reason is that long batches of write operations are eventually performed at the network transfer rate, while short batches of write operations may take advantage of Telegraphos buffering. However, the net result is that the programmer sees that a remote write operation takes less than 0.5 μsec!

Remote read operations are less efficient: they take a few microseconds, because they need to talk to the remote HIB, read the result, communicate it to the local HIB, to the TURBOchannel, and eventually to the processor who remains blocked throughout the entire operation.

Telegraphos local read and write operations may seem a bit expensive (0.87, and 0.46 μs respectively). The reason is twofold:

- In Telegraphos-I the local portion of the shared memory of each node resides on the TurboChannel network interface. Thus, local shared memory accesses have to pay at least one TurboChannel roundtrip delay. Telegraphos II, and more recent versions do not suffer from this problem, because the local portion of the shared memory is just a portion of the computer's main memory.
- In the first version of Telegraphos we have used (rather small) FPGAs. To perform even simple local operations several FPGAs need to cooperate. Each time some data travel from one FPGA to another, the cost is (usually) increased by one more TurboChannel cycle (80 ns). Recent versions of Telegraphos do not suffer from this problem as they are being designed in ASIC technology.

To place the Telegraphos Architecture in perspective we compare it with other architectures running from workstation clusters to large scale multiprocessors (table 1). We see that Telegraphos is significantly better than previous generation multiprocessors (like Intel IPSC/2), is comparable to modern workstation clusters (like SCI Dolphin Cluster, and the Memory Channel used for DEC's workstation clusters), and multiprocessors (like Intel Paragon XP/S, and Cedar), and is significantly better than Local Area Network whose communication is based on software-implemented TCP/IP on top of Ethernet. We believe that the performance of future Telegraphos systems will be significantly improved for the reasons mentioned above.

4 Telegraphos using SCI-over-ATM

We currently develop a new architecture for a Telegraphos system that provides:

1. PCI bus interface;
2. ATM network connectivity;
3. SCI framing.

There are several reasons for defining this new generation of the architecture. The main ones that drive this effort (which are also goals of the architecture) are two: **(i)** caching of all data for high performance, and **(ii)** use of standards for easy scalability and "openness" to a large number of systems provided by different vendors.

4.1 Data Caching

One of the main drawbacks of architectures such as Telegraphos-I, PRAM [17], SHRIMP [2], etc, is that shared data backed by remote main memory cannot be cached. The reason is that all these architectures are designed with their shared, network memories non-cacheable, since their interfaces are attached to the workstation's I/O bus. In a typical conventional architecture, all memory attached to the I/O bus cannot be cached for coherency reasons. New emerging processor architectures seem to be able to overcome this limitation, and provide mechanisms that allow designers to build systems with all memory space cacheable. Such interesting features currently appear in the specification of DEC's Alpha processor [18], but we expect that their usefulness will attract other manufacturers as well.

An analysis of a *remote read* operation shows how the various consistency problems may be solved with processor features such as the ones included in Alpha: let us assume that 2 processors, P_1 and P_2, are interconnected through a Telegraphos network, and they are caching all data. We also assume that processor P_1 reads a variable v which resides in P_2. Then:

1. P_1 issues a *load instruction* that causes a data miss that requests the data from the Telegraphos interface;
2. the interface issues a *remote read* operation to P_2 for the variable v;
3. the interface eventually receives a response from P_2 (from the network) containing the value v;
4. v is provided to the cache, and from there to the processor. The solution for

System name	Link Latency (μsec)	Throughput (Mbps)	Source
SCI Dolphin Workstation Cluster	4 (one way)	-	[19]
Memory Channel	<5 (one way)	-	[5]
Intel IPSC/2	350	39	[21]
Local Area Net (Ethernet - TCP/IP)	800	6	[21]
Intel Paragon XP/S	15	1600	[21]
Cedar	1.1	190	[21]
Telegraphos I	7.2 (roundtrip)	103	-

Table 1: Comparative performance of Telegraphos-I and related architectures.

caching in this case is coming from the Alpha architecture, considering that the interface is attached to the I/O bus: the processor allows cacheable data over the I/O bus [18]. Furthermore, the ability of the processor to implement a consistency protocol, enforces consistency between all data in the cache and the Telegraphos interface. Since current implementations of Alpha-based workstations seem to not provide all this functionality, as an alternative we consider provision of 2 bus interfaces on the card: one for the outgoing data through the I/O bus, and one for the memory bus of the used workstation. In this case, all incoming data can be cached, since they are read through the memory bus, but special consideration has to be paid to the fact that the shared data may be stale, if the system's cache is write-back.

Regarding processor P_2, the data validity issues are resolved as follows:

- if v is in P_2's cache, then the validity of the data read by P_1 is guaranteed, if P_2 has configured the memory portion where v resides as cacheable with write-through or write-back with update as the Alpha architecture handbook describes [18]. If the capability is not provided, then data must be consistent before the remote read is served; this can be implemented through memory barriers (MB instructions on Alpha);
- if v is only in P_2's main memory, then data is valid.

Thus, several alternatives exist for building an interface for Telegraphos on the I/O bus enabling caching as well.

As the features described above are not included in currently available systems, data caching of remotely read data can be implemented only through attachment of the Telegraphos interface on the memory bus (in addition to the I/O bus), so that incoming data arrives to the processor through that bus.

4.2 Use of Standards

The available Telegraphos prototype system is not an "open" design, since neither the interface nor the network use standardized, or widely used interfaces. It is important though to design interfaces and networks that can interconnect heterogeneous systems provided by a wide range of different vendors. So, the new version of the Telegraphos architecture is directed towards the use of standards. The main parameters that need to be considered in such an environment are: the minimum packet size and, the bandwidth/latency requirements. These are the main parameters, since the I/O bus choice is clearly made by the vendors providing the workstations. All these considerations led us to the choice of the following standards:

1. ATM network technology;
2. SCI packet framing;
3. PCI bus interface.

ATM is an emerging network transport technology that provides high bandwidth, low latency and interoperability with other ATM systems. The choice of ATM is an important one in our architecture for 2 reasons:

- It fits the requirements of the new generation Telegraphos which is expected to have larger packet size. Such longer packet size fits well with the characteristics of ATM, which has a cell size of 53 bytes with 48 bytes of useful data in the ATM Adaptation Layer-5 (AAL-5) which we intend to use.
- ATM is a technology that seems to be widespread in the near future in both LAN and WAN environments.

The remote memory operations implemented by Telegraphos require a reliable network that does not drop packets, and delivers packets in order. Fortunately, recent ATM switches [8, 20] provide flow control, and guarantee in-order packet delivery.

SCI is also a standard allowing scalability and ability to interconnect other SCI systems from different vendors. As coherence is an option in SCI and not a concern of the Telegraphos architecture, our interface will only provide SCI framing without using (or supporting) the SCI coherence options. To implement SCI-over-ATM, a number of ATM virtual circuits will be reserved to carry remote memory requests framed in an SCI format. When the host workstation issues a remote memory read or write operation, the Telegraphos interface will use one of the special ATM VCs to send this request to the appropriate host. The Telegraphos interface on the destination host will receive the ATM cell over the special VC number, and treat it as a shared memory operation. It will assemble the SCI packet from possibly several ATM cells, and execute the read or write operation requested. As long as the ATM network provides in order guaranteed delivery of packets, the shared memory operations will work without a problem.

Finally, PCI seems to be the choice of upcoming high performance workstations and PCs. Its ability to reach 1 *Gbps* throughput (increasing to 2 *Gbps* in PCI-2) as well as its low latency and the fact that it is a standard are attractive features that will allow development of high speed, "open" systems that can accommodate several vendors' interface cards.

5 Related Work

The design of efficient shared-memory systems has been the focus of several groups in the last decade. Building efficient shared-memory multiprocessor systems is crucial for applications that need high performance. Several shared-memory multiprocessors have been built, from small-scale bus-based multiprocessors [16] to large-scale distributed memory machines [4, 1].

Although networks of workstations may have an (aggregate) computing power comparable to that of a supercomputer (while costing significantly less), they have rarely been used to support high-performance computing, because communication on a network of workstations has traditionally been very expensive, making it prohibitively expensive for an application to use more then a few workstations.

There have been several projects to provide efficient communication primitives in networks of workstations via a combination of hardware and software: PRAM [17], MERLIN [22, 11], Galactica Net [7], Hamlyn [3] DEC's Memory Channel [5] and SHRIMP [2] provide efficient message passing on networks of workstations based on memory-mapped interfaces. Their shared-memory support, though, is limited because they do not provide individual single remote memory accesses; thus a processor that wants to access a few words out of a page is forced to replicate the whole page locally, and then access its data - moreover, as long as the page is replicated, it has to be kept coherent. SHRIMP and PRAM provide efficient methods of keeping copies of pages coherent but do not provide user applications the ability to access a remote page without keeping a local copy of this page as well. Thus, the total amount of shared memory a processor may see at any time is limited by the amount of its local memory. In Telegraphos, instead, the amount of shared memory that a processor may see at any time is the total amount of shared memory in the system. Besides that, Telegraphos provides several sophisticated shared-memory primitives like remote atomic operations, and non-blocking fetch operations.

6 Summary

In this paper we describe *Telegraphos*, a distributed system suitable for efficiently supporting both message-passing and shared-memory applications on top of high-speed networks. Telegraphos has a memory-mapped network interface that avoids almost all software imposed communication overhead. It uses the page mapping and protection mechanism, existing in almost all virtual memory systems, to implement protection in message passing. Telegraphos also implements a fast remote-write hardware primitive that enables one processor to send a message to another processor by simply writing directly into the receiver processor's memory. No software is involved in passing the message, apart from the initialization phase that makes sure that the sender is allowed to send messages to the receiver. The receiver gets the message by simply reading its local memory. Besides being efficient, Telegraphos is also affordable, because it can be connected into an existing workstation environment, and upgrade it into a loosely-coupled multiprocessor.

We believe that Telegraphos demonstrates that it is feasible to build inexpensive shared-memory systems based on existing workstations. The main idea is to provide hardware support for the necessary shared-memory operations while leaving complicated coherence decisions to software and to users that are willing to pay the cost of coherence if they are going to benefit from it.

Acknowledgments

Part of this work was developed in the ESPRIT/HPCN project "SHIPS", and will be used for the OMI project "Arches", funded by the European Union. We deeply appreciate this financial support, without which this work would have not existed.

We would like to thank P. Vatsolaki, D. Gustavson, G. Dramitinos, and C. Papachristos for useful comments in earlier drafts of this paper.

References

[1] BBN Laboratories. Butterfly Parallel Processor Overview. Technical Report 6148, BBN Laboratories, Cambridge, MA, March 1986.

[2] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *Proceedings of the Twenty-First Int. Symposium on Computer Architecture*, pages 142–153, Chicago, IL, April 1994.

[3] G. Buzzard, D. Jacobson, S. Marovich, and J. Wilkes. Hamlyn: a high-performance network interface, with sender-based memory management. In *Proceedings of the Hot Interconnects III*, August 1995.

[4] T. H. Dunigan. Kendall Square Multiprocessor: Early Experiences and Performance. Technical Report ORNL/TM-12065, Oak Ridge National Laboratory, May 1992.

[5] R. Gillet. Memory Channel. In *Proceedings of the Hot Interconnects III*, August 1995.

[6] Hakan Grahn and Per Stenstrom. Efficient Strategies for Software-Only Directory Protocols in Shared Memory Multiprocessors. In *Proceedings of the Twenty-Second ISCA*, Santa Margherita Ligure, Italy, June 1995.

[7] Andrew W. Wilson Jr., Richard P. LaRowe Jr., and Marc J. Teller. Hardware Assist for Distributed Shared Memory. In *PROC of the Thirteenth International Conference on Distributed Comput-*

ing Systems, pages 246–255, Pittsburgh, PA, May 1993.

[8] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis. Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip. *IEEE Journal on Sel. Areas in Communications*, 8(9):1265–1279, 1991.

[9] M. Katevenis, P. Vatsolaki, and A. Efthymiou. Pipelined Memory Shared Buffer for VLSI Switches. In *Proceedings of the ACM SIGCOMM '95 Conference*, August 1995. URL: file://ftp.ics.forth.gr/tech-reports/1995/ 1995.SIGCOMM95.PipeMemoryShBuf.ps.gz.

[10] Manolis Katevenis. Telegraphos: High-Speed Communication Architecture for Parallel and Distributed Computer Systems. Technical Report 123, ICS-FORTH, May 1994.

[11] C. Maples. A High-Performance Memory-Based Interconnection System for Multicomputer Environments. In *Proceedings of the Supercomputing Conference*, pages 295–304, 1992.

[12] E.P. Markatos and C.E. Chronaki. Trace-Driven Simulations of Data-Alignment and Other Factors affecting Update and Invalidate Based Coherent Memory. In *Proceedings of the ACM International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '94)*, pages 44–52, January 1994.

[13] E.P. Markatos and G. Dramitinos. Implementation of a Reliable Remote Memory Pager. In *Proceedings of the USENIX 1996 Technical Conference*, January 1996. Earlier version published as TR 129, at Institute of Computer Science, FORTH; URL: file://ftp.ics.forth.gr/tech-reports/1995/ 1995.TR129.remote_memory_paging.ps.gz.

[14] H. E. Meleis and D. N. Serpanos. Designing Communication Subsystems for High-Speed Networks. *IEEE Network Magazine*, 6(4):40–46, July 1992.

[15] R. Rettberg and R. Thomas. Contention is No Obstacle to Shared-Memory Multiprocessing. *Communications of the ACM*, 29(12):1202–1212, December 1986.

[16] Sequent Computer Systems Inc. *Balance 8000 System*, 1985.

[17] D. Serpanos. *Scalable Shared-Memory Interconnections.* PhD thesis, Princeton University, Dept. of Computer Science, October 1990.

[18] R. Sites. Alpha AXP Architecture. *Communications of the ACM*, 36(2):33–44, February 1993.

[19] Dolphin Interconnect Solutions. Dolphin Breaks Cluster Latency Barrier with SCI Adapter, 1995. press announcement.

[20] R. Souza, P. Krishnakumar, C. Ozveren, R. Simcoe, B. Spinney, R. Thomas, and R. Walsh. GIGAswitch System: A High-Performance Packet-Switching Platform. *Digital Technical Journal*, 1(6):9–22, 1994.

[21] B.K. Totty. Experimental Analysis of Data Management for Distributed Data Structures, 1992. Master Thesis, University of Illinois at Urbana-Champaign.

[22] Larry Wittie and Creve Maples. Merlin: Massively Parallel Heterogeneous Computing. In *Proceedings of the 1989 ICPP*, pages I:142–150, 1989.