ory. In this case, the entire system of TMC and TIC is considered. However, the TMC capacity limits the realizable throughput for small numbers of nodes. In this case, the interconnect capacity is practically infinite. As system size grows, the usable bandwidth per node drops below the TMC capacity. The bandwidth and latency for single and dual TIC configurations below is the effective performance as seen by an idealized application. This includes the traffic that is used to maintain cache coherency, which is not included in the shown bandwidth

S3.mp nodes have bulk memory copy facilities. These operate cache coherently and can achieve peak transmission rates of up to 66% of the raw bandwidth in one direction.

6 Project Status

S3.mp is a experimental prototype project and not a currently planned Sun product.

The TIC chip has been fabricated and is used in a message interface board that provides message passing capabilities similar to those of the IBM SP-2 system or Myrinet.

The TMC design is in the final stages of verification and is expected to be submitted for fabrication in June. Functional TMC chips should become available in mid-August, leading to first operational S3.mp nodes in the late Summer of ‘95.

7 Summary

The main innovation of the S3.mp project is the support for spatially distributed shared memory multiprocessing. Unlike other cache-coherent DSM multiprocessors, S3.mp system can be distributed over a local area while still maintaining competitive performance. S3.mp remote miss latencies compare favorably [21] to the reported internode latency for the recently introduced Convex SPP-1000 Parallel Computer and S3.mp bisection bandwidth is close to the Cray T3D realm. This performance level is achieved at a relatively low system cost: all S3.mp components can be integrated in one modestly sized ASIC with standard CMOS technology. Other innovations of the S3.mp project include the use of DSM to reduce communication overhead. S3.mp allows spacial distribution of computational resources while preserving the convenience and efficiency of using memory semantics to access them. Additionally this allows workstations and I/O resources to be viewed as building blocks that make up a scalable system rather than a distributed collection of independent entities. Finally S3.mp hides the artifacts of adaptive routing, namely out of order delivery, by integrating it with a order insensitive coherency mechanism that is based on memory models defined for the Sparc architecture.

7.1 References

[1]Bell, G. *Ultracomputers, A Teraflop Before Its Time*. Communications of the ACM 35,8 (August 1992), 27-47.

[2]Lenoski, D. *The Design and Analysis of DASH: A Scalable Directory-Based Multiprocessor.* PhD Dissertation, Stanford University, December 1991.

[3]Bisiani, R., and Ravishankar, M. K. *Design and Implementation of The Plus Prototype*. Technical Report, Carnegie Mellon University, August 1990.

[4]Thapar, M., Delagi, B., and Flynn, M., *Linked List Cache Coherence for Scalable Shared Memory Multiprocessors*, Proceedings of the 1993 International Conference on Parallel Processing, pages 34-43.

[5]Agarwal, A.,Kubiatowicz J., Kranz, D., Lim, B., Yeung, D., D’Souza, G., Parkin, M. *Sparcle: An Evolutionary Processor Design for Large-Scale Multiprocessors*. IEEE Micro, June 1993, pages 48-61.

[6]Nowatzyk, A. *Communications Architecture for Multiprocessor Networks.* PhD Dissertation, Carnegie Mellon University, December 1989.

[7]SPARC International, *The SPARC Architecture Manual: Version 9,* Prentice-Hall, 1994

[8]Kendall Square Research, *Technical Summary.* 1992. 170 Tracer Lane, Waltham, MA 02154-1379

[9]Hagersten, E., Landin, A., and Haridi, S. *DDM - A Cache-Only Memory Architecture.* IEEE Computer 25,9 (September 1992), 44-54.

[10]Li, K. *Shared Virtual Memory on Loosely Coupled Multiprocessors.* PhD Dissertation, Yale University, September 1986.

[11]Bershad, B. and Zekauskas, M. *Midway: Shared Memory Parallel Programming with Entry Consistency for Distributed Memory Multiprocessors*. Technical Report CMU-CS-91-170, Carnegie Mellon University, September 1991.

[12]Bal, H., Jeffifer, S., Tanembaum, A. *Programming Languages for Distributed Computing Systems.* ACM Computing Surveys 21,3 (September 1989), 261-322.

[13]Herlihy, M. *Wait-Free Synchronization*. ACM Transactions on Programming Languages and Systems 11,1 (January 1991), 124-149.

[14]Dill, D., Drexler, D., Hu, A., Yang, C. *Protocol Verification as a Hardware Design Aid.* 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors, (October 1992), 522-52

[15]McMillan, K., *Symbolic Model Checking,* 1992, Carnegie Mellon University PhD Thesis, Pittsburgh, PA 15213

[16]Eichelberger, C.W., Davidson, H., *Viking Supersparc MCM Developement Program*, 1993 IEEE Multichip Module Conference, Santa Cruz, p. 28-32

[17]Nowatzyk, A., Parkin, M., *The S3.mp Interconnect System and TIC Chip,* Hot Interconnects’93, Stanford CA, Aug. 1993

[18]Nowatzyk, A., Browne, M., Kelly, E., Parkin, M., *S-Connect: from Network of Workstations to Supercomputer Performance*, Proceedings of the 22nd Symposium on Compter Architecture, June 1994

[19]Saulbury, A., Nowatzyk, *A. A Simple COMA Implementaion on the S3.mp Multprocessor*, 5th Workshop on scalable shared memory multiprocessors, June 1995

[20]Pong, F., Dubois, M., Nowatzyk, A., Aybay, G., *Verifying Distributed Directory-based Cache Coherency Protocols: S3.mp, a Case Study,* In proceedings of EuroPar’95, Stockholm/Sweden, August 1995

[21]Sterling, T., Savarese, D., Merkey, P., *Evaluation of the SCI based Convex SPP-1000 Parallel Computer*, SCI Workshop, Santa Clara, CA, 1995

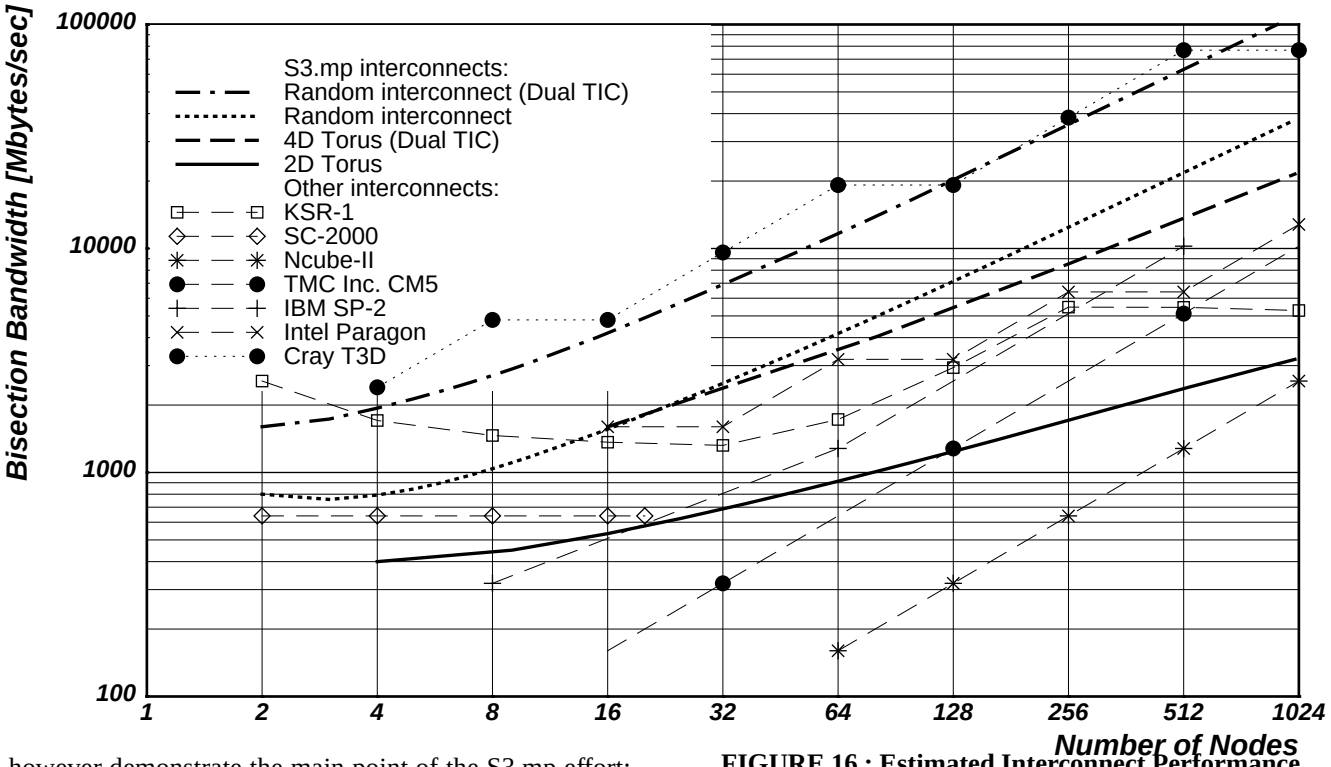


FIGURE 16 : Estimated Interconnect Performance.

however demonstrate the main point of the S3.mp effort: the software overhead of message passing dominates all hardware factors. Also, the internode bandwidth of a TMC based system is much higher because bulk data transfers do not need to traverse the Mbus.

Performance estimates for larger systems are based on simulations and analytical models. The TIC bisection bandwidth and latency for 2D meshes, 4D meshes and the average of random interconnect topologies with 1 and 2 TIC chips are given in Figure 14. The shown bandwidth includes only the data portion of single packets that are addressed to random destinations and does not include bandwidth used for flow control and CRC. To present a conservative estimate, the bandwidth is derated to 80%, to allow for congestion effects.

The reference points for other interconnect systems in Figure 16 use 100% of the raw physical bandwidth with no deductions for any communication overhead, congestion or router inefficiencies. They represent the upper performance limits which cannot be reached in real applications.

The raw bandwidth that is provided by the TIC interconnect system is not directly available to an application process, rather there is an average of 4 control messages for each data message. Since the control messages are 4 times smaller than the data message, this leaves still more than halve of the raw bandwidth.

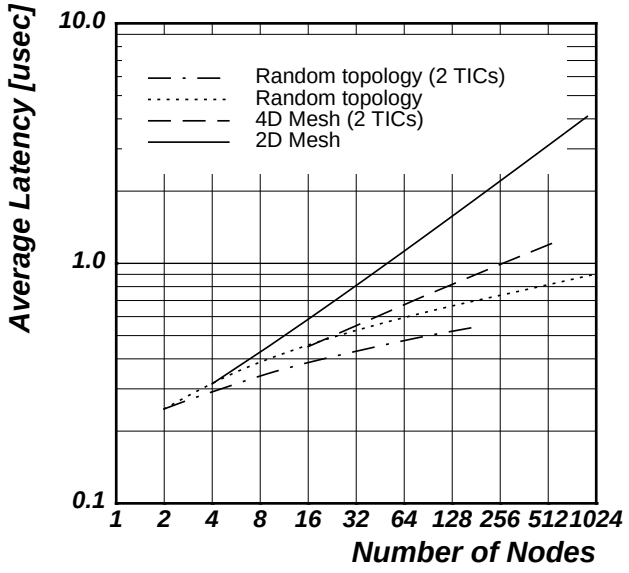


TABLE 1 : Interconnect System Performance.

Number of nodes	Bandwidth / node	Remote access latency
2	<406* (<812*) MB/s	0.84 (0.84) µs
8	112 (<313*) MB/s	1.8 (1.4) µs
32	63 (194) MB/s	2.8 (1.9) µs
128	43 (144) MB/s	3.8 (2.4) µs
512	32 (117) MB/s	5.0 (2.7) µs
1024	26 (96) MB/s	6.1 (3.1) µs

*: Limited by TMC performance

Table 1 summarizes the TIC performance for randomly generated topologies and random access to remote mem-

engine. The current implementation of the TMC chip uses 4 TSRF windows. More register windows will be beneficial in hiding the latency of remote accesses when used in conjunction with processors that have non-blocking caches and compilers which utilize aggressive prefetching schemes.

Transactions received by the protocol engines can be of two types: request and acknowledge. Request transactions require a new microcode thread to be generated. For transactions received from the Mbus controller, the transaction type relayed by the scheduler is used as an entry point into the microcode. For packets received from the input queue, type information from the packet is used as the entry point. The next available TSRF window is assigned to this transaction and a new thread is generated and marked as ready to execute. When the microcode sequencer detects that this thread is ready to run, it starts executing it. The TSRF entry corresponding to this thread becomes part of the state space of the microcode sequencer for the duration of this transaction. Until this thread is completed, all other subsequent transactions with the same ID will be suspended.

When an acknowledge packet is received by the RAS or the RMH, the ID field of this packet is compared to the ID fields of threads that are currently sleeping. If a sleeping thread waiting for the particular acknowledge packet is found. This thread is woken up, i.e., marked as ready to run, otherwise a new thread is forked to deal with an unsolicited packet. The microcode sequencer resumes processing a thread whenever it is available.

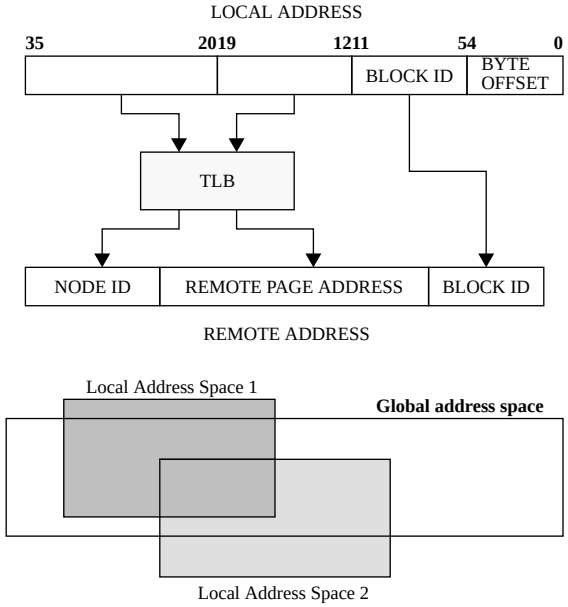
A set of timers, one for each TSRF thread, are used to deal with the case where an acknowledge is expected but never received from a remote node. Whenever a thread is suspended, its timer is programmed to generate an error acknowledge if an reply is not received within a certain amount of time.

In addition to providing a way of matching sleeping threads and reply messages, the ID field provides a convenient way to lock the directory and/or the INC while these data structures are in a transient state. Each engine can potentially lock *N* out of 128 slices of the directory/INC independently, where *N* is the number of entries in the TSRF. ID-locking is utilized exclusively by S3.mp directory protocols to implement message-delay independent operation.

4.4 Address translation

The TMC uses a TLB based address translation mechanism. The size of the segments being mapped from the global dress space to the local address space can be as small as 4Kbytes or as large as 1Mbyte. TLB misses are handled by the operating system. The S3.mp architecture defines a 64-bit address space. In the current implementation, modules have a 36-bit local address space, therefore, each node can only see a limited portion of the global address space at any given time.

FIGURE 15 : TMC Address Translation



5 Performance

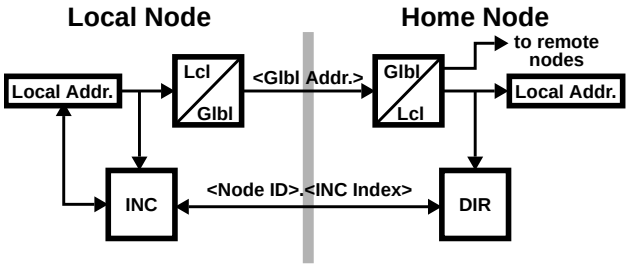
The first lot of TIC chips was received in November of 1994 and are used to interconnect a small network of workstation. The FPGA-based TIC demo boards only allow message passing and bulk memory-to-memory copy operations. In this capacity, the system is used to verify the correct operation of the TIC chip and to gain operational experience with the high-speed serial link technology. The performance of this system is largely limited by the Mbus-bandwidth, and the demonstrated transmission bandwidth was only slightly higher than 80 Mbytes/sec. Likewise, the rate at which messages were exchanged was limited by the software overhead to a sustained rate of 250K packets/sec with bursts to >1M packets/sec.

The TIC demo boards run at 51.840 Mhz (instead of 66 Mhz), which is due to some limitations in the analog section of the serial link core. The rest of the TIC chip has been tested at speed. The measured TIC traversal latency is 310ns, which includes 2m of cables and all serialization/deserialization and synchronization overhead. The distributed clock synchronization provided by the TIC kept all clocks stable to within 100ps.

The TIC demo system (currently 5 nodes as of March, ‘95) is too small to stress-test the TIC. It has essentially infinite bandwidth compared to interface bandwidth, which is limited by the Mbus. However, the measured performance is in line with the simulation results. It does

In the S3.mp architecture, the directory and the INC are closely related. Whenever a node accesses remote data, the INC will maintain a copy of it for the entire duration of the transaction. An active directory entry implies that there is at least one remote INC that has the corresponding data. Once an INC entry is reused, the directory entry corresponding to this block is updated. This relation allows the 64 bit global addresses to be abbreviated by only the node id and the INC index (Figure 12). As a result of this abbreviation scheme, S3.mp cache coherence protocols use very short messages (80-bits) for majority of the transactions.

FIGURE 12 : Address Abbreviation

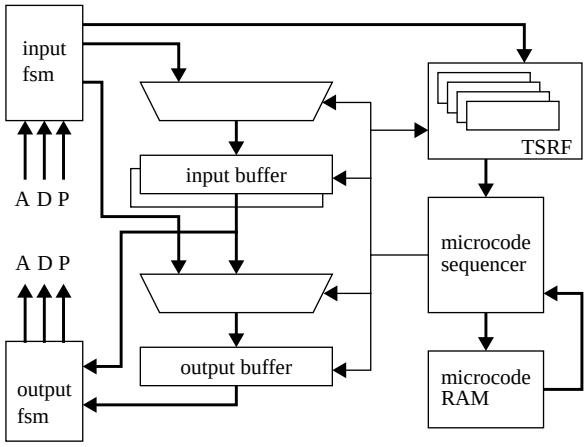


4.3 Protocol Engines - RAS and RMH

S3.mp directory protocols are implemented by two protocol engines on the TMC chip, the *Remote Memory Handler* and the *Remote Access Server*. The RMH is responsible for the transactions generated locally that refer to memory locations on a remote node. It also services invalidation and data forwarding requests originated by the home node and it maintains the INC. The RAS is responsible for all transactions that refer to local memory and that involve a remote cache. It services data request messages from remote nodes, maintains the directory and generates invalidation and data forwarding messages.

The structure of the RMH and RAS is shown in Figure 13:

FIGURE 13 : Protocol Engine Datapath



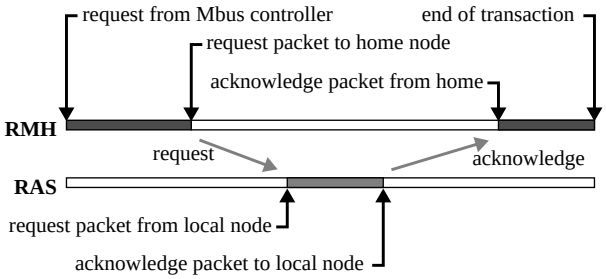
Directory protocols are separated into two parts, the server (RAS microcode) and the client (RMH microcode). Microcode is stored in on-chip RAM and appears as part of TMC.

A typical operation of the protocol engines is shown in Figure 14. Typically, there are two kinds of activity going on at the protocol engines:

1. Transactions that require an acknowledge. These kind of transactions typically have a short burst of local activity (i.e. Mbus cycles, memory cycles) which is terminated by sending a request packet. From this point on, the microcode waits for a reply from a remote node. This wait period is in the order of microseconds in the current implementation of the S3.mp system. When the reply is received, the transaction is terminated following another short burst of local activity.

2. Transactions that do not require an acknowledge. These are typically initiated by receiving a request packet and can be served using only the local resources of the receiving node. They are terminated by sending an acknowledge packet to the requestor.

FIGURE 14 : S3.mp Protocol Engine Activity



Due to the long latency associated with waiting for a reply, it would have been inefficient to run the protocol engines in a mode where the microcode sequencer was kept busy for the entire duration of the transaction while polling on the reply packet. To introduce a degree of parallelism in order to utilize the high bandwidth available from the interconnect network, the protocol engines were implemented as multithreaded state machines. RMH and RAS have Transaction Status Register Files (TSRF) with multiple contexts to keep track of multiple concurrent transactions. Each context includes all state that is needed to perform a memory operation (addresses, timers, state, pointers, etc.). Each TSRF entry consists of 121 bits, which exceeds the information that could be stored in either the directory or the INC. Hence the directory and INC can maintain only the stable states while active TSRF entries are required for all transient states that have outstanding messages.

The lowest 7 bits of the cache block address (bits [11:5] of the local address for 32-byte cache lines) are used as a tag for transactions. These 7 bits are preserved during address translation, thus, they effectively divide the global address space into 128 sets. Any protocol engine can be working on *N* of these sets concurrently where *N* is the number of register windows in the TSRF of that protocol

FIGURE 9 : Reading Data from a Remote Node

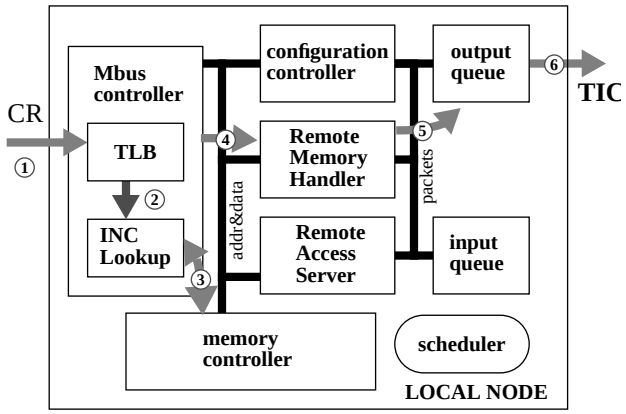


FIGURE 10 : Servicing Remote Read Request

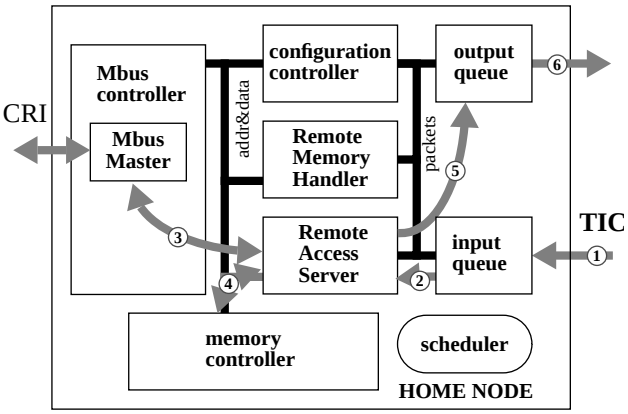


Figure 10 depicts the events of the TIC at the home node upon arrival of the request packet:

1. The request packet is removed from the interconnect by the input queue (like the TIC and the output queue, the input queue supports 4 priority levels that are guaranteed not to block each other in order to avoid deadlocks of the CC-protocol).

2. The input queue decodes the type of the packet to determine the destination unit and sends the packet to the RAS.

3. The RAS requests a snoop cycle on the home Mbus[1]. If one of the home caches have an exclusive copy of this cache line, data is read from this cache. This snoop action is necessary since the current S3.mp protocols do not distinguish between the resident directory state and states where a cache line is shared or owned by one of the CPUs at the home node.

4. If the snoop cycle has failed, the RAS reads the cache line and the associated directory information from the main memory.

5. If the directory is valid (i.e. the directory is not in *Exclusive_Remote* state), an acknowledge packet including the data is constructed and sent to the output queue. The RAS updates the directory information in the main memory if necessary. Assuming that no other node had a copy of the cache line involved in this transaction, directory state will be changed from *Resident* to *Shared_Remote_1* and the directory pointer will be updated to point to the local node.

6. The acknowledge packet including the data is sent back to the local node through the interconnect.

After the acknowledge packet is received by the remote node (Figure 11):

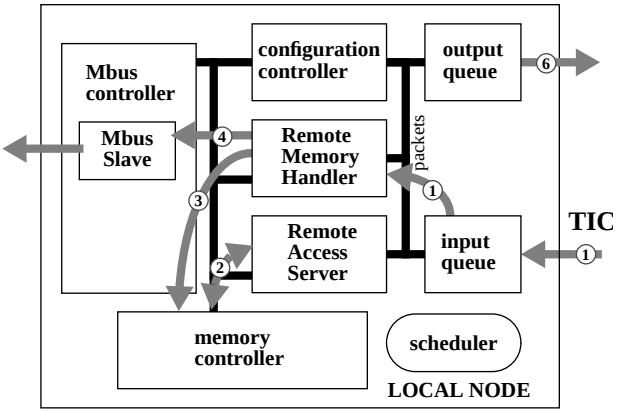
1. The acknowledge packet is removed from the interconnect by the input queue and sent to the RMH. This packet wakes up the sleeping RMH process which had initiated the request for this transaction.

2. The RMH reads the INC tags corresponding to the address of the cache line from the memory and allocates an INC entry. If there are no free INC entries available, RMH victimizes an existing INC entry in LRU fashion.

3. Data from the packet is written to the INC and INC tags are updated.

4. The Mbus controller is call to complete the transaction. This is done by enabling arbitration for the original requestor which will retry the transaction which will be served from a preload buffer.

FIGURE 11 : Completion of a Remote Read



The basic flow of transactions outlined above illustrates the operation of the TMC and the interactions of its sub-component. The complete microcode for the RAS/RMH requires 400/432 instructions, most of which are needed to deal with corner cases. The details of this protocol exceeds the scope of this paper. This protocol was successfully verified in a collaboration with the University of Southern California [20].

1. The directory may store the shared local and the exclusive local states of the memory block, thus eliminating the need for snoop cycles on the home node. However, since the directory is kept in the main memory, this would convert many ordinary local memory read operations on the home node into Read/Modify/Write cycles. Being able to service read cycles going to local memory as fast as possible at the expense of having to do snoop cycles on the home Mbus for remote operations seemed to be a good trade-off for most applications. In future implementations of s3.MP systems, TMC will most likely be integrated into the CPU. In this case, TMC will be able to snoop the CPU cache much faster.

handle two simultaneous transactions with a 4 or 8 bank memory subsystem. The split transaction communication protocol used in the TMC reduces the contention on internal busses by reserving the busses only when data transfer is taking place. It also introduces a level of parallelism *and* pipelining. It is possible to send the address for a transaction on the address bus while an independent data transfer is taking place on the data bus. Moreover, since acknowledges are generated by the target module as new transactions, communication does not depend on the number of cycles needed to process a request.

The concept is similar to object oriented programming. Each module has a small set of externally visible data structures and a set of transactions (methods) that operate on these structures. Modules can be tested independently of the whole system by constructing simple test environment that exercise all supported transactions. Retargeting the TMC design is important for this project so that the developed technology can be used in different platforms and/or technologies. For example, the memory controller module has a very simple functional definition: it either reads or writes a 32-byte block of memory (a cache line) of memory. This requires that some of the logic to handle byte insertion, etc. is moved into other modules, but it makes the memory controller extremely modular. A Rambus or E-DRAM based version of the memory controller can be integrated with a future version of the TMC design with minimal interface redesign overhead.

The Modular design methodology also decreases the complexity of the verification. Of particular importance is to formally verify the correctness of the inter-module communication protocol so that the problem of verifying the entire TMC decomposes into verifying each module independently. Simple module interfaces and the small set of transactions supported by each module allow to deal with the communication protocol at a high level of abstraction where formal verification tools like *Mur*φ [13] are applicable. With the simple interface approach, integration of modules at different levels of abstraction is also possible. The TMC uses synthesizable Verilog HDL descriptions that are interchangeable with C++ objects that form the S3.mp architectural model.

4.2 TMC Directory Operation

TMC has a 128-bit datapath to the memory. It uses a SDRAM memory subsystem with 144-bit data interface. 9 bits are used to maintain ECC over 128 bit data words and 7 bits are left unused. This gives us 14-bits of extra storage for every 32-bytes of main memory. TMC uses these 14 bits to maintain the directory information for cache-coherency protocols. Two bits are used to maintain the state of the directory and 12 bits are used to maintain a pointer to a remote node or to a set of extension node pointers.

The TMC reserves a programmable fraction of the main memory and uses this storage as a cache for remote references. This internode cache (INC) is required to include all locally cached blocks of remote memory and is critical for an address compression scheme that conserves bandwidth for control messages. Since remote references have at least 3x higher latency than local memory references, even a relatively slow (= operating at main memory speed) INC is beneficial. The INC is programmable in size and may occupy up to 50% of the total memory. It is implemented as a 3-way set associative, write-allocate cache with LRU replacement policy. Although S3.mp in this configuration is still fundamentally a CC-NUMA architecture, it does have many of the properties of a cache only memory architecture (COMA). Hence S3.mp offers a variable degree of COMA behavior that can be used to fine tune the system for specific application.

Since the protocol engines are micro-programmable, it is possible to use a different cache-coherency protocol that supports Simple-COMA, which is a hybrid COMA scheme, where storage allocation is maintained in software. S-COMA microcode is being developed for S3.mp in collaboration with the Swedish Institute of Computer Science [19].

The S3.mp directory uses a multiple linked list scheme to keep track of nodes having copies of a cache line. We will use the following popular naming convention introduced in [2] to explain the basics of the S3.mp directory operation:

1. *Home* is the node which has the original copy of cache line in its main memory
2. *Local node* is the node which gets a remote copy of the cache line
3. *Remote nodes* are any other nodes having a copy of the same cache line

Initially, a cache line is only resident at the memory of the home node and directory is in the *resident* state. When a CPU on another node reads a cache line from a remote node for the first time, the following sequence of events takes place:

1. A CPU initiates a Coherent Read (CR) transaction on the Mbus.
2. The local TMC detects that this transaction is within its remote address range and it performs an address translation to convert the local address to a global address.
3. The Mbus controller checks the local INC to see if this cache line has been retrieved previously.
4. Upon missing in the INC, the Mbus controller sends a *LOAD_LINE* request to the RMH.
5. The RMH constructs a request packet for the cache line and sends it to the output queue. The process responsible for handling this transactions is suspended until a reply arrives.
6. The request packet traverses the output queue and is sent to the home node via the TIC interconnect system.

The TMC is structurally divided into the following set of modules (Figure 7):

1. A bus controller to interface to the Mbus. Both passive (a local CPU access the TMC) and active (the TMC issues a bus transaction on behave of a remote node) are supported.

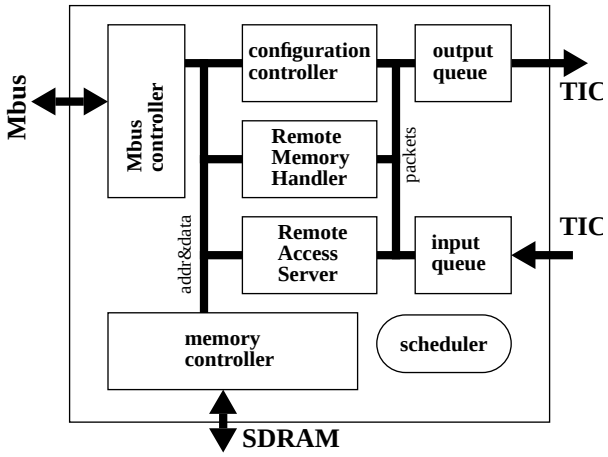
2. A memory controller that directly generates all signals necessary to drives multiple banks of synchronous dynamic memory (SDRAM). ECC is maintained over 128 bits, using 9 bits of redundancy codes. Given a standard 144 bit organization, this leaves 7 bits for each half of a cache line to store directory state information.

3. Two identical protocol engines to implement distributed cache coherency protocols (RAS and RMH).

4. A control unit to take care of configuration management, errors, diagnostics, timers, interrupts, etc.

5. Input and output queues for interfacing to the TIC (or to a general purpose point-to-point interconnect).

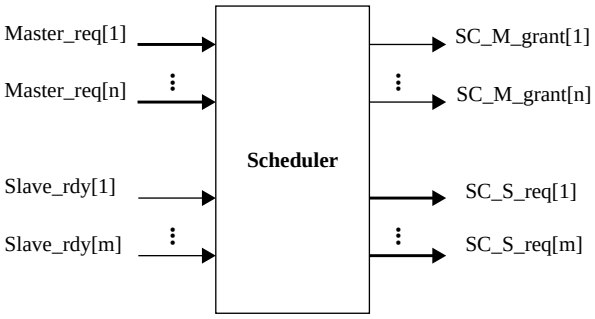
FIGURE 6 : TMC Structure



4.1 Modular Design Methodology

The TMC is designed as a system on a chip, where modules interact only through a small set of well defined transactions. Modules are designed to be as functionally self-sufficient as possible. The number of inter-module signals are kept to an absolute minimum. Functional units communicate with each other through an address bus, a data bus and a packet bus. Access to these busses are managed by a central controller, the *scheduler*. The scheduler receives dedicated request lines from each master module and ready lines from each slave module (Figure 7) and sends dedicated grant lines to each master and dedicated request lines to each slave. Requests requiring data or acknowledgments to be returned are handled as split transactions. Most slaves are required to be able to also function as a master and initiate transactions to return data and/or acknowledgments. This methodology allows independent design and verification of the modules. It also aids design reuse: modules can be replaced easily to interface to different system bus, different memory devices, etc. On the downside, the inter-module interfaces require some extra pipeline stages that a monolithic design could avoid.

FIGURE 7 : TMC Resource Scheduler



A typical transaction proceeds as shown in Figure 8:

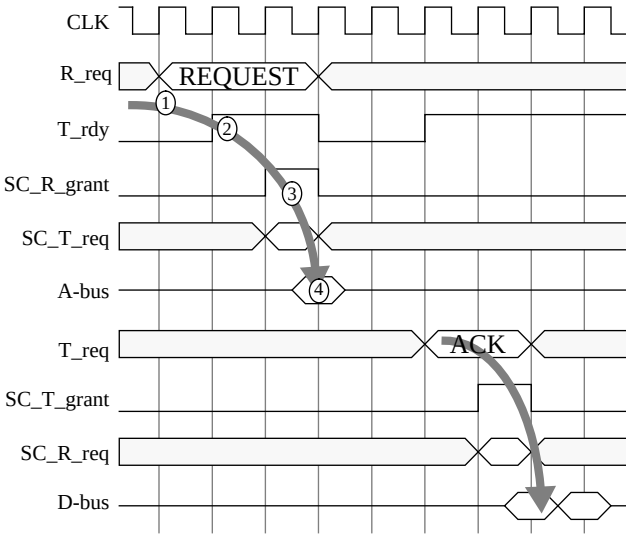
1. The initiator asserts its dedicated request line (R_req) to the scheduler. R_req is a multi-bit signal including information about the resources to be used (i.e. A-bus, D-bus, P-bus), the number of cycles required for the transaction, module ID of the target unit and the type of the transaction. Each unit has its own dedicated R_req line going to the scheduler.

2. Slave modules assert a dedicated ready line (T_rdy) to the scheduler whenever they are ready to accept transactions.

3. Every cycle, the scheduler examines the request inputs from all master modules and from all slave unit ready lines. When the scheduler determines that all resources required for a request are available, it schedules a transaction for that request. Necessary communication resources are reserved for the duration of the transaction. A grant signal (SC_R_grant) is sent to the requestor and the request type and requestor ID is relayed to the target module (with SC_T_req signal).

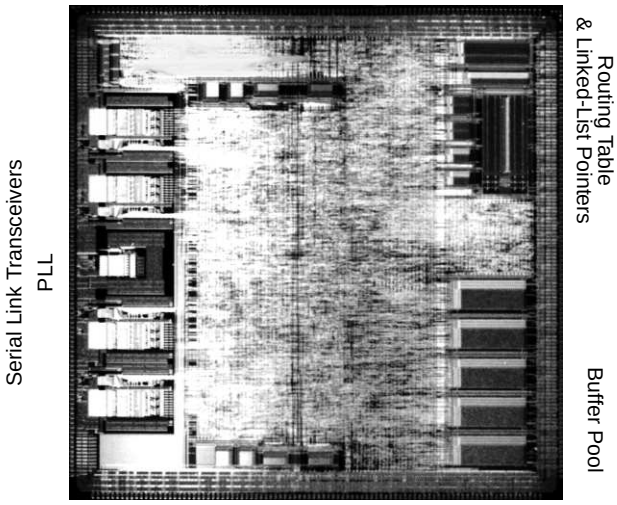
4. Requestors are responsible for driving data on the bus as soon as they receive the grant signal from the scheduler. Typically, requestors latch the data into their output registers in the cycle they assert their request. The grant signal from the scheduler is used as an enable to drive data from these registers to the busses in the same cycle the transaction is scheduled.

FIGURE 8 : Internal Bus Protocol of TMC



Three major transaction initiators on the TMC, *the Mbus Controller, the Remote Memory Handler* and *the Remote Access Server,* compete for access to the memory controller. The memory controller is designed to be able to

FIGURE 4 : TIC Chip Microphotograph



Vital Statistics

Chip Size:	11.8 x 11.8 mm
Technology:	0.65 µm CMOS, 2 Metal Layers
Power Supply:	4W @ 5V
# of Logic Gates:	48,000
Design Style:	Custom + Gate Array
On-chip Memory:	7520 bits
External Clock:	66 Mhz
Test Support:	Full ATPG Scan

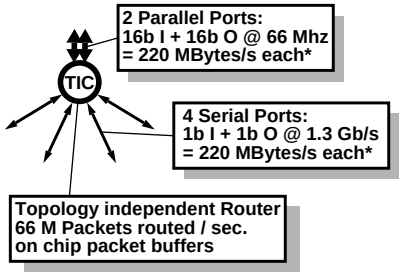
3 The S3.mp interconnect system

The S3.mp interconnect system is built out of single chip packet switches (TICs, Figure 4). Each TIC is a six ported router that can receive and send one packet on each port during each routing cycle (= 6 system clock cycles). Fixed length packets are used, which allowed the use of a statically scheduled pipeline. Two of the TIC ports use a 16 bit wide, parallel, full duplex interface (Figure 5). The other four ports use high speed serializer and deserializer pairs that are an integral part of the TIC chip. Hence the TIC does not require any costly external serializers, such as Glink (HP), HotRod (Gazelle) or Taxi (AMD) chips.

Unlike the interconnect networks of other scalable systems that use a specific topology (meshes, rings, fat-trees, etc.), S3.mp has no preferred topology, rather it explores the actual interconnect network whenever nodes are added or removed from the system. An off-line process computes the routing tables which are then loaded into each TIC. It is possible to add or remove nodes and change the interconnect topology while other parts of the systems are operating normally.

The details of the TIC chip and its technology are described in [17, 18]. The TIC interconnect system has been operational since November 1994. For the remainder of this paper, the TIC based interconnect system should be regarded as a black box that provides reliable packet delivery between arbitrary nodes.

FIGURE 5 : The TIC chip[1]



4 The Memory Controller

The TMC is responsible for handling accesses to local and remote memory and for implementing directory based cache coherence protocols. Besides acting as a normal memory controller, the TMC performs the following functions:

1. Maintaining the directory information for the local memory under its control
2. Constructing and sending messages to remote nodes on the network initiated by local Mbus transactions that require remote access or in response to messages received from other nodes on the network
3. Performing memory operations and Mbus cycles on the local node in response to messages received from remote nodes
4. Maintaining an Inter-Node Cache (INC) which is used to store a copy of every cache line retrieved from remote nodes
5. Sending and receiving diagnostic messages to/from other nodes to program configuration parameters, handle errors, etc.

From the CPU's point of view, the TMC operates as a virtual bus extender. Except for the extra latency in accessing remote locations, details of retrieving data from remote nodes and of maintaining a distributed cache coherence protocol, are completely hidden from these processors. Parallel application software targeted for a shared-bus cache-coherent multiprocessor system, written in accordance with SPARC memory models [7], executes correctly on a S3.mp system without any modification. However, performance tuning may be necessary to achieve good performance. This property of the S3.mp architecture is particularly important in being able to utilize existing SMP based applications. Besides supporting existing parallel code based on shared memory programming model, the TMC, acting as a virtual bus extender, enables single-threaded programs to utilize the collective physical memory in a network of workstations for memory intensive applications (e.g. CAD synthesis, formal verification, image processing, database applications).

1. The bandwidth figures are the usable packet throughput. The parallel ports transfer data only on 5 out of 6 cycles. The serial links carry an additional 16 CRC and handshake bits, that are not included.

enables multiple, cooperating operating systems to run on one S3.mp system, each of which is isolated and can tolerate the loss of other OS domains.

- S3.mp must have facilities to dynamically change the system configuration, i.e. add or remove nodes.
- S3.mp components must be remotely controllable, so that the entire system can be maintained from a single point.

The S3.mp architecture supports real-time and multimedia applications by providing support for global synchronization, precise clock distribution, multicasting, and prioritized bandwidth allocation.

1.1 Motivation and project goals

S3.mp based multiprocessors will not compete with Supercomputers in terms of absolute performance. However the S3.mp project tries to build the most cost-effective and the easiest to use scalable computing environment.

The first goal, cost-effective computing, leads to resource sharing. Currently, most installed workstation class systems waste significant amounts of memory, disk space and other hardware components because they cannot be shared efficiently over conventional networks. The 64+ Mbytes of main memory in the idle system next-door cannot be used to aid a memory intensive application on another machine. Gbit/sec fiber technology provides the means to change this situation. Other ingredients for cost-effective computing include advances in caching, latency hiding, system integration, high density VLSI circuits, and new packaging methods.

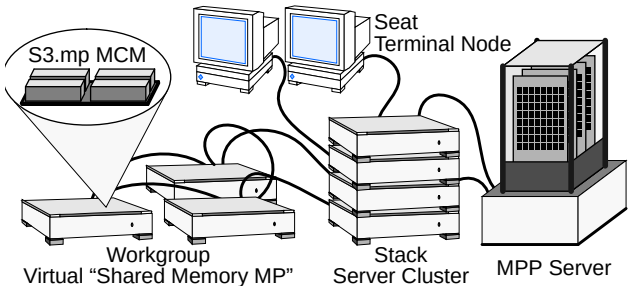
Software for parallel systems does not come easy, but evidence is mounting that the shared memory paradigm leads to higher programmer productivity than explicit message passing. This comes at a modest increase in hardware complexity and some increase in bandwidth demand. This situation is not unlike virtual memory vs. program controlled overlays. While the latter can be more efficient in some situations and does require less hardware, virtual memory became ubiquitous because of its ease of use.

1.2 S3.mp system configurations

The S3.mp prototype system will use multichip modules (MCM) to integrate the interface, 64 Mbytes memory and two processors into one compact node (less than 0.9 in^3). Prototypes of this MCM technology have been successfully tested [16]. The processor memory bus (Mbus) is exposed so that the S3.mp processing element can be added to several existing workstation platforms. Interconnecting a number of workstations in this manner leads to a workgroup that can run shared memory applications.

Low end implementations may omit the processors and use the memory as a frame buffer that is integrated into the monitor. Such terminal nodes can be connected to work groups or centralized high end MPP servers. High performance servers could be realized by densely packing many modules into one system.

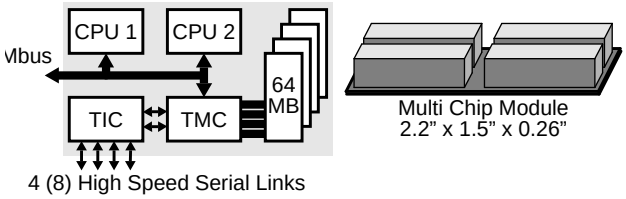
FIGURE 2 : S3.mp Configuration Examples



2 The S3.mp processing element

Each S3.mp node consists of 2 gate arrays that are part of one multi chip module (Figure 3). The memory controller (TMC, 172K gates) is in charge of maintaining memory coherency while the interconnect controller (TIC, 48K gates) is the building block for a scalable interconnect network. The TMC translates memory transactions from the processor bus (Mbus) to a set of messages exchanged among remote nodes. TMC can also initiate transactions on the Mbus as a result of messages coming from remote nodes. In the default configuration, two processors are connected to each TMC through the Mbus.

FIGURE 3 : A S3.mp Node



The TMC is designed to directly drive a 32-128 Mbyte memory array that uses 16 Mbit SDRAM devices with a high performance interface. Coherency is maintained on memory blocks of 32 bytes, which is the size of one cache line on Mbus based systems. The architecture is not tied to a particular cache line size because remote data is cached in the TMC and the TMC could maintain subblocks or multi-line objects.

TMC uses 18 bits of ECC overhead and 14 bits of directory overhead for every 32-bytes of memory. 14 bits provide sufficient storage to keep a 12-bit node pointer and 2-bit state for each cache line.

The Mbus of each S3.mp node can be connected to other devices, which is the primary facility to attach I/O devices to the system. In particular, it is possible to plug a S3.mp module into any Mbus slot (for example into a Sparc Station 10 or 20 Workstation).

The S3.mp Scalable Shared Memory Multiprocessor

Andreas Nowatzyk, Gunes Aybay, Michael Browne,
Edmund Kelly, Michael Parkin, Bill Radke, Sanjay Vishin
Sun Microsystems Computer Corporporation
2550 Garcia Ave., Mountain View, CA 94043
E-mail contact: agn@acm.org

Abstract

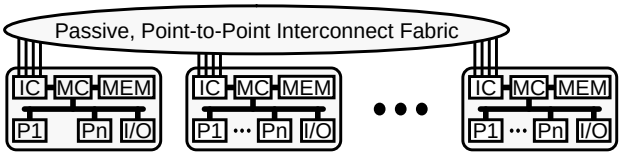
S3.mp (Sun's Scalable Shared memory MultiProcessor) is a research project to demonstrate a low overhead, high throughput communication system that is based on cache coherent distributed shared memory (DSM). S3.mp uses distributed directories and point-to-point messages that are sent over a packet switched interconnect fabric to achieve scalability over a wide range of configurations. S3.mp uses a new CMOS serial link technology that achieves transmission rates >1Gbit/sec and that is directly integrated into a packet router chip. Unlike other DSM systems, S3.mp can be spatially distributed over a local area via fiber optic links. This capability allows S3.mp to interconnect clusters of workstations to form multiprocessor workgroups that efficiently share memory, processors and I/O devices. Multichip module technology, the integrated arbitrary topology router, fast serial links, and a cache coherent DSM system that is integrated into the memory controller allow compact, massively parallel S3.mp systems.

1 Introduction

The S3.mp scalable multiprocessor system is an experimental research project that is being implemented by SMCC Technology Development group (TD) to demonstrate a low overhead, high throughput communication system that is based on cache coherent distributed shared memory (DSM). Advances in fiber optics and CMOS circuit technology allow the construction of low cost, high speed interconnect fabrics with bisection bandwidths in excess of 100 Gbytes/sec. Such systems cannot be used efficiently via conventional networking protocols. Instead S3.mp uses the notion of shared memory, where communication happens as a side-effect of accessing memory: a single store or load instruction is sufficient to send or receive data. The set of transactions that are required to support the DSM paradigm is small and well defined so that the S3.mp protocols were amenable to formal verification methods and could be implemented directly in hardware.

The S3.mp architecture is similar to ALEWILE, DASH, PLUS [2,3,5] and other cache coherent, nonuniform memory access (CC-NUMA) multiprocessors. However unlike these conventional CC-NUMA MP's, S3.mp is optimized for a large collection of independent and cooperating parallel applications that share common computing resources which may be spatially distributed. Consequently, S3.mp nodes may be separated by up to 200m, which means that a S3.mp system could be distributed over an entire building. S3.mp systems can be built by adding a specialized interconnect controller to the memory subsystem of an ordinary workstation.

FIGURE 1 : S3.mp System Overview



Workstations suitable for S3.mp interfaces must have a cache coherent processor / memory interconnect system. The first implementation of the S3.mp architecture is based on the *Mbus*, which is used in the SparcStation-10/20 series. Each node may have several processors, each equipped with caches that are kept consistent via snooping. When a processor accesses a remote memory location, the S3.mp memory controller translates the bus transaction into a message that is sent to the remote memory controller, which replies with the data and maintains a directory of blocks cached by external nodes.

S3.mp's interconnect controller (TIC) is part of a distributed switch that is optimized for the fine grained, irregular traffic of a DSM system. There is no need for a centralized switch, rather each node comes with a number of high speed links to connect to other nodes. Hence the interconnect fabric grows incrementally, as nodes are added to the system. Because the S3.mp architecture allows spatial distribution it must address a number of issues that are of lesser concern to conventional CC-NUMA machines:

- The interconnect topology is subject to external constraints, hence, the interconnect controller must be able to deal with arbitrary interconnect topologies.
- Fault tolerance is important to some S3.mp configurations. This requires hardware support to manage and protect multiple address spaces. This address translation mechanism