Hardware Implementation of Multicast Communication for Interconnection Networks

[1]I. N. Cota, [2]J. S. Aude

Federal University of Rio de Janeiro, NCE/IM
PO Box 2324 – Rio de Janeiro – RJ – 20001-970 – Brazil
[1]iuri@nce.ufrj.br, [2]salek@nce.ufrj.br

Abstract—
This work is concerned with the implementation of hardware resources to support multicast communication on the Multiplus distributed shared memory multiprocessor. The multicast messages to be transmitted are those required for an efficient implementation of directory-based cache coherence protocols. To achieve this goal, a redesign of the current implementation of the Multiplus multistage interconnection network is proposed. This paper presents the main concepts of the overall redesign and describes the message transmission modes, the cache coherence protocols, the network switching element control logic and the overall multistage interconnection network architecture which have been conceived. An implementation of the network switching element on an Altera EPLD has been carried out from a VHDL description of the circuit functionality.

Keywords— **Interconnection Network, Message, Transmission Modes, Multicast and Broadcast Communication, Message Routing, Directory-Based Cache Coherence Protocols.**

I. INTRODUCTION

Multiplus [Aude96] is a distributed shared memory multiprocessor designed to support up to 128 processing elements (PE's) based on SPARC processors and 32 Gbytes of global memory space organized into 32 clusters. In each cluster, four PE's, one I/O Processor and a Network Interface are interconnected through two 64-bit bus systems, one for instruction and data access and the other for block transfer operations. Within the Multiplus NUMA architecture, shown in Fig. 1, the clusters are interconnected through an inverted n-cube Multistage Interconnection Network. In its larger configuration, the network consists of five stages of 2x2 crossbar switching elements [Bron96] and is able to interconnect 32 clusters. The switching elements perform the message routing through the network under the wormhole mode. The message header carries the information used by the switching elements to route the message to one of its two outputs. This information consists of 5 bits. Each bit controls the routing process through a switching element belonging to each network stage. Figure 2 illustrates an example of such networks consisting of 3 stages. In this case, if the message header at port 1 holds the value "011", the switching element at the first stage will

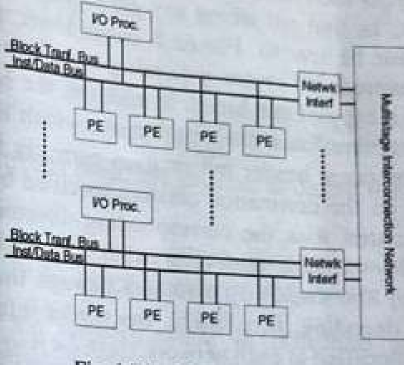

Fig. 1 The Multiplus Architecture

route the message to its output #0, according to the most significant header bit. Following a similar reasoning with the header second bit, the switching element receiving the message at the network second stage will route this message to its output #1. Finally, the switching element at the third stage will route the message to its output #1 which is connected to port 3 ("011"), the destination port specified by the header. With this simple routing scheme, the header format and its decoding are not complex and the message transmission process can be simple and fast. However, within distributed shared memory architectures, cache block invalidation messages are frequently required by the coherence protocols and this kind of message has to be sent to all processing elements which hold a copy of the cache block to be invalidated. Unfortunately, the current implementation of the Multiplus multistage interconnection network only supports point-to-point communication and, therefore, can only send multicast messages as a set of independent single-destination messages.

Due to its importance for the efficient implementation of collective communication primitives and cache coherence protocols, multicast communication has been extensively studied for direct networks such as meshes [Lin94, Moha96] and, in a more limited way, for indirect networks such as the multistage interconnection networks [Vara99].

This work proposes a redesign of the Multiplus multiprocessor multistage interconnection network which will support broadcast and multicast communication messages sent to the destination ports in a single network transmission. This facility greatly improves the efficiency of cache coherence protocol implementations.

Within the Multiplus multiprocessor, the cache coherence protocols to be implemented are based on the release consistency [Bers93] and on the entry consistency [Ghar90] memory models. Both protocols will benefit from the provision of an interconnection network with hardware support to multicast communication of cache block invalidation messages. These protocols require the use of distributed directory structures [Leno90], which store information on the clusters holding copies of each shared cache block, to find out where any particular invalidation message must be sent to. For each cache block, full-map directories store a bit vector indicating the set of clusters which have a copy of that block. In addition, the Multiplus directory structure has a header generator which is able to codify the message header information correctly from the information on the destination clusters specified by the bit vector associated with the corresponding directory entry. Basically, the message routing at each network stage is performed by the switching elements through the header information decoding. In addition, the header information needs to be modified at each network stage for it to hold the source cluster identification when it reaches the destination clusters. This information is required for these clusters to send back the acknowledgment reply messages.

Section 2 of this paper describes the proposed directory-based cache coherence protocols to be implemented on the Multiplus multiprocessor. Section 3 discusses in detail the operation modes of the network switching elements defined by the header models to be adopted. Section 4 describes the header generation procedure. Section 5 describes the switching-element design to incorporate the requirements of the new network functionality. Section 6 describes the behavioral simulation of the overall porposal, which has been performed to validate it. Finally, Section 7 summarizes the work and describes its next evolution steps.

II. DIRECTORY BASED CACHE COHERENCE PROTOCOLS

The cache coherence protocols to be adopted within the Multiplus multiprocessor are based on the use of distributed directories. Two memory consistency models are supported: entry consistency and release consistency. The caches operate under the write-through scheme and the cache block size is 32 bytes.

The entry consistency memory model requires the use of acquire and release operations at the beginning and end of each critical section, respectively. In addition, every shared variable must be associated with a synchronization variable. When an acquire operation is performed on a synchronization variable, only the shard variables guarded by that particular synchronization variable are made consistent.

With the use of the entry consistency memory model, a dirty block list (DBL) per acquire and per processor and a full-map directory per cluster are the main data structures which are held in hardware within the Multiplus architecture. In addition, a stack of active DBL pointers per processor is kept in software to cope with perfectly nested pairs of acquire-release operations. The DBL works as a direct mapped cache, which holds the list of cache blocks that have been written in between an acquire and the corresponding release operation. The directory holds, for every cluster local cache block, the list of remote clusters which may have a valid copy of that block in the private caches of their processors. One single bit is associated with each cluster for each directory entry.

To analyze the protocol operation it is necessary to look into the details of how the following four basic processor functions are performed: acquire, read, write and release. The acquire and release operations are implemented under software control with the assistance of dedicated hardware. When an acquire operation is performed, the processor obtains the identification of the last DBL in use before the last corresponding release operation. There are three possible situations. The first one takes place when the first acquire is performed and, as a consequence, there was no previous DBL in use. In this case, the processor obtains the identification of the next free DBL available and sets it as the current DBL in use. If there was a previous DBL in use associated with this processor, its identification is saved in the active DBL pointer stack. The second situation takes place when there was a previous release operation and the DBL in use belonged to the same cluster. In this case, this DBL simply becomes the DBL in use again. Finally, a third situation takes place when a remote DBL was the last one in use by the last corresponding release operation. In this case, the same procedure adopted in the first situation is repeated and the remote DBL is copied to the new local DBL using the DMA facility provided by the interconnection network interface. Then, an invalidation operation takes place to remove from all caches within the cluster the blocks stored in the DBL which are marked as having copies in private caches within that cluster.

Read and write cache coherence protocol operations are totally performed under hardware control. When a remote read operation is performed after an acquire and before the corresponding release operation, the target cluster updates, if necessary, the directory entry corresponding to the particular cache block which has been read by setting the bit associated with the source cluster.

In write operations, after an acquire and before the corresponding release operation, it is firstly necessary to identify if the block to be written is remote or local. If it is local and is already present in the DBL in use, nothing needs to be done. If it is local but not present in the DBL the information stored in the local directory corresponding entry is stored in the DBL entry assigned to the block to be written. If the corresponding DBL entry is not free, a flush operation is started by sending a multicast invalidatio

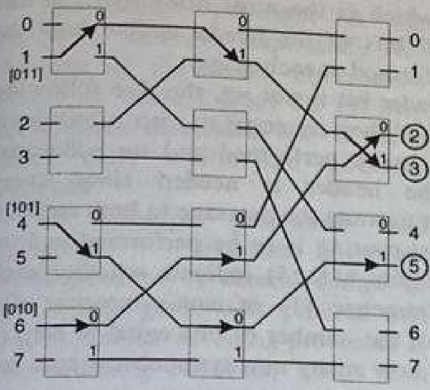

Fig. 2 N-cube Network

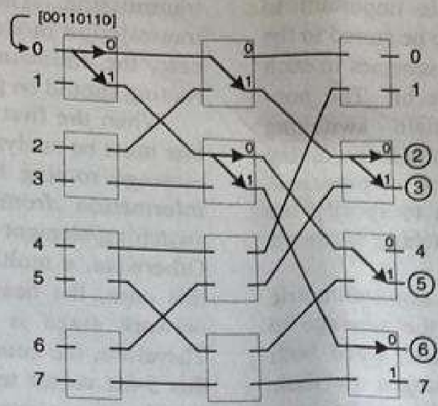

Fig. 3 Multicast Transmission

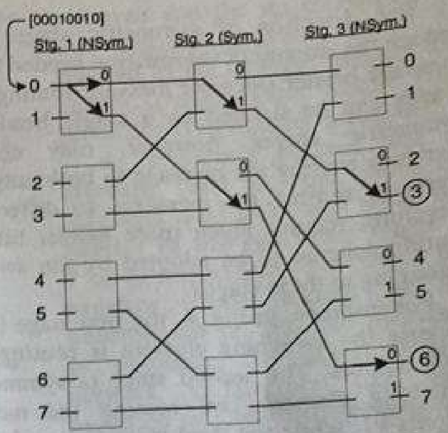

Fig. 4 Symmetric and Non-Symmetric Stages

message through the interconnection network to all clusters holding copies of that particular block.

When the block to be written is remote and not present in the DBL, the destination cluster sends back, as a reply to the write operation, the contents of the directory entry associated with the written block. This information is stored in the DBL at the source cluster. Once again, if the DBL entry is not free to store this information, a flush operation takes place with the use of a multicast message.

Finally, when a release operation is performed, all pending write operations have to be completed before releasing the synchronization variable. It is possible to check this information because all write operations in between an acquire and the corresponding release operation are required to have an acknowledge reply sent by the destination cluster. After the completion of the pending write operations, the DBL identification is stored in memory as the last one associated with a release operation and a pop operation is performed in the active DBL pointer stack associated with that particular processor in order to define the new DBL in use.

With the use of the release consistency memory model, basically the same data structures used with the entry consistency memory model are held in hardware. Read and write operations are performed exactly in the same way described for the entry consistency memory model. However the acquire and release operations have to be modified.

When an acquire operation is performed, a much simpler procedure takes place. The processor obtains the identification of the next free DBL available and sets this DBL as the current DBL in use. If there was a previous DBL in use associated with this processor, its identification is saved in its active DBL pointer stack.

For the release operation, after ensuring that all pending write operations have been performed in memory, the current DBL is scanned and invalidation multicast messages are sent to all clusters holding copies of data blocks which have been modified. Once this operation is completed, the DBL contents can be discarded and the DBL is made available for future use. A pop operation is then performed in the active DBL pointer stack associated with that particular processor in order to define the new DBL in use.

III. Multicast Communication within the Interconnection Network

For the implementation of broadcast and multicast communication within the interconnection network, it is necessary to modify the message routing procedure within the switching elements since, in some cases, the message will have to be routed to both outputs. The header must carry complex enough information to precisely describe the transmission mode (peer-to-peer, broadcast or multicast) and how the message should be routed through the network. For multicast communication, the header must indicate how each network switching element should route the message (Figure 3). However, the header size is limited by the number of bits of the word which is transmitted through the network. If this size is bigger than the word length, the message transmission gets slower and a more complex hardware is required to deal with the header. In the redesign of the Multiplus multistage interconnection network, the word length to be transmitted through the network is 16 bits.

With this limitation on the header size, it is not possible to adopt a simple representation based on the use of a single bit for every possible destination cluster of a message, as in the full-map directory structure, since there may be up to 32 clusters in the current definition of the Multiplus architecture.

The compression of the header information is performed by the header generator circuit, which is part of the directory structure. This circuit is able to recognize occurrences of *symmetric routing* (all the switching elements of a particular network stage route the incoming messages to either the output #0 or the output #1) along the complete path to be followed by the message in the network and use this information to reduce the header size. A network stage is said to be symmetric when the number of messages that reach the switching elements at that stage is the same as the number of messages leaving that stage, because all the messages have either been routed to the outputs #0 or to the outputs #1 of the switching elements.

The detection of symmetric stages is important to reduce the amount of information needed to be stored in the message header since the message routing schemes in such stages can be defined by a single header bit. The non-symmetric stages, however, may contain switching elements routing the message to both outputs or switching elements routing the messages to different outputs and, therefore, require much more header bits to specify the routing scheme to be adopted by the switching elements belonging to these stages.

For instance, in Fig. 4, the first stage is non-symmetric because the switching element is routing the message to both outputs. The second stage is symmetric, since both switching elements are routing the messages to their outputs #1, while the third stage is non-symmetric because the switching elements are routing the messages to different outputs.

To establish a classification of the transmission modes, it is necessary to have more than a single header model for the multicast messages because, depending on the number of the message destination clusters, the header information encoding may be simpler or more complex. For a network interconnecting 32 clusters, even when there are only 2 destination clusters for a message, it is possible to find 80 configurations with 4 symmetric stages and 1 non-symmetric stage, 160 configurations with 3 non-symmetric

TABLE I
THE HEADER MODELS

Symmetric Stages	Non-Sym. Stages	Header Model	# of bits
5	0	1SSSSS	6
4	1	0TTTTTSSSSNN	12
3	2	0TTTTTSSSNNNN	13
2	3	0TTTTTSSNNNNNNNN	16
1	4	01TTTTSSNNNNNNNN (x2)	16
0	5	011TTTSSNNNNNNNN (x4)	16
0	Broadcast	000000	6

stages, 80 configurations with 4 non-symmetric stages and 16 configurations in which every stage is non-symmetric. The adopted solution in this work classifies the header types as a function of the number of non-symmetric stages as shown in Table I.

When every stage is symmetric, a peer-to-peer message is transmitted. Therefore the header needs to specify only 5 routing bits, one for each network stage. When one or more stages are non-symmetric, a multicast message is transmitted, except in a broadcast message, when all the clusters are message destinations. When the first header bit, h(0), is set, it indicates that a peer-to-peer message is under transmission. Therefore, it is simple to detect this transmission mode, which is the most often used. In this case, the following 5 bits of the header define how the routing should be performed at each stage.

When the first header bit is not set, the five following bits must be analyzed. If they are equal to zero, a broadcast message routing has to be performed and no additional information from the header is needed since every switching element has to route the message to both outputs. Otherwise, a multicast routing is to be performed and, in this case, the header bits h(1.. 5) indicate whether each network stage is symmetric (1) or non-symmetric (0). Therefore, the count of the number of bits equal to zero in this 5-bit vector tells how many non-symmetric stages are used in the message transmission.

When the number of non-symmetric stages is equal to 1, the header bits h(6..9) indicate how the routing should be performed in the four symmetric stages and the bits h(10..11) indicate how the routing should be performed in the single non-symmetric stage. In this case, these bits will necessarily be set to "11" because the single non-symmetric stage can only be routing the message to both outputs of the switching element which is receiving the message. In a similar way, when the number of non-symmetric stages is 2 or 3, the header bits h(6..8) or h(6..7) indicate, respectively, how the routing should be performed in the symmetric stages. The header bits h(9..12) or h(8..15) indicate, respectively, how the routing should be performed in the non-symmetric stages.

When the number of non-symmetric stages is equal to 4, the message is transmitted twice with the header using the same format adopted when only 3 non-symmetric stages are present. In the first transmission, stage 1, if it is a non-symmetric stage, or stage 2, otherwise, is set as a symmetric stage which routes the message to the outputs #0 of its switching elements. In the second transmission, this stage is set to route the message to the outputs #1 of its switching elements. The 8-bit vector which defines the routing to be performed at each of the 3 non-symmetric stages is not the same in both transmissions, because each operation will reach a different group of destination clusters.

A similar approach is adopted when the number of non-symmetric stages is equal to 5. In this case, four message transmissions are performed and in each of them the header model is the same used when there are only 3 non-symmetric stages. The four transmissions refer to the 4 routing options in the first two network stages. For each one of these options, different patterns of the 8-bit vector are used to define the routing to be performed by the 3 non-symmetric stages.

IV. THE HEADER GENERATOR

The header generator is the hardware module responsible for building up the header information shown in Table 1 from the bit map information associated with a cache block entry within the directory structure.

The header generation and decoding processes are interdependent. A careful design of the header generation process can lead to simpler decoding processes at the switching elements, which is a necessary condition for reducing the message transmission delay through the network. The header generation process with the formats presented in Table 1 can be simple, when broadcast messages are considered, or complex, in the case of multicast messages. In both cases, however, it is firstly necessary to identify the header model which will have to be used.

If only a single bit is set in the corresponding directory bit vector entry, a peer-to-peer message transmission is to take place. So, the first header bit has to be set to 1 and the header S bits must indicate that a symmetric routing mode is to be used at each network stage. However, if more than one bit, but not all, is set in the directory entry, a multicast message transmission will have to take place and, therefore, the first header bit has to be set to 0. If all directory entry bits are set to 1, a broadcast message is to be transmitted and the first six header bits have to be set to 000000.

The header generation process for multicast messages is more complex because it needs to know the routing mode (symmetric or non-symmetric) at each network stage. In addition, the header non-symmetric vector (N bits) has to be defined. This vector can be 2, 4 or 8 bits long depending on whether there are 1, 2 or 3 non-symmetric stages, respectively. The header non-symmetric vector is only analyzed by the non-symmetric network stages during the message transmission. To obtain this vector it is firstly necessary to visualize the network paths along which the multicast message will flow. Then, the network symmetric stages are removed and the set of outputs in this reduced network, which is reached by the message, defines the header non-symmetric vector. This procedure is illustrated in Figure 5.

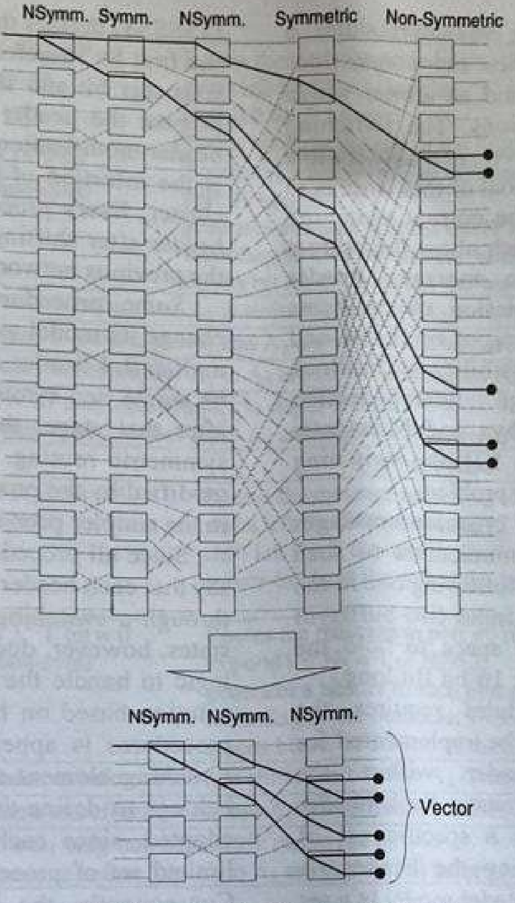

Fig. 5 Header Non-Symmetric Vector

As previously pointed out, in principle, two or four multicast messages need to be sent when the number of non-symmetric stages is 4 or 5, respectively. To reduce the negative impact caused by the transmission of multiple messages in sequence, the interconnection network architecture has been modified to allow two messages to be sent at the same time without any alteration in the header configuration and in the switching element logic design. This is achieved with the duplication of the first two stages of the network as shown in Figure 6. The merge of the two sub-network outputs can be implemented very easily with simple tri-state buffers because only one of the ouputs in each pair to be merged can be in use at any time instant. This is ensured by the header codification when more than a single multicast message has to be transmitted. When the first network stage is non-symmetric, the header generator builds two headers, each one specifying that the first stage is symmetric. In the first header, S(1) = 0, and in the second one, S(1) = 1. The first header can lead messages to the destination ports 0 to 15 while the second one can reach the ports 16 to 31. When the multicast message requires the first two stages to be non-symmetric, 4 different headers are created and in each one of them the first two stages are defined as symmetric. Therefore, each header will lead its message through a different path in the first two network stages and, therefore, two messages cannot reach the same output port of the second network stage when they are sent by the duplicated stages at the same time.

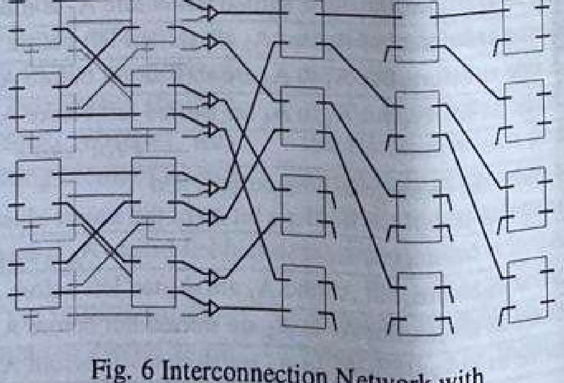
Fig. 6 Interconnection Network with Duplication of the First Two Stages

V. THE SWITCHING ELEMENT DESIGN

The Multiplus multistage interconnection network switching element [Bron96] consists of: fifo buffers at the input ports which temporarily store received packets; input controllers which are responsible for the synchronization and control of the remaining switching element modules; a 2x2 crossbar switch which establishes the connections between the input and output ports; and an arbiter which controls the crossbar switch connections. The switching element current implementation, however, can only route a message to a single output. The main goal of this work is to remove this limitation by adding to the current switching element design, the logic modules which allow the correct decoding and modification of the message header information required for the transmission of multicast messages. In addition, since the sending of multicast and broadcast messages introduces the possibility of deadlock situations in wormhole multistage interconnection networks, as pointed out by Varavithya and Mohapatra [Vara 99] and by Stunkel et al. [Stun 97], the switching element new design also deals with this problem.

Deadlock situations in multicast and broacast messsages are avoided by ensuring that no communication of such type is forwarded from one stage to the following one in the network without making sure that the input fifo buffer in the destination stage has enough free space to hold the complete message, which is always four 16-bit flit long.

The design of these logic modules requires the definition of the procedures that should be implemented for decoding and modifying the header within the interconnection network switching elements. Each header model presented in Table 1 suggests a specific set of procedures for decoding and modifying the header. A particular procedure associated with a header model is a set of tasks that the switching element has to perform when it receives a message with a particular configuration of that header model. These procedures can be applied by the switching elements regardless of the network stage to which they belong. Therefore, whenever a switching element receives a message, it must analyze the message header to select the procedure to be adopted.

Every procedure modifies the header information. For instance, the header model for peer-to-peer communication (1SSSSS) defines the message routing at each stage through the first S. Therefore, at each stage, the switching element tests this bit and shifts to the left the remaining S bits to prepare the header for the next stage decoding. Another header modification which is common to all header models is the insertion of one bit to identify the message source cluster. Every procedure inserts this bit at the end of the header after shifting to the left the bits already inserted by the previous network stages.

Some procedures, however, modify the header and change its model as well. For instance, a message with the multicast 1 header model (0TTTTTSSSSNN) may perform its single non-symmetric routing at any network stage. So, after that stage, the message transmission only requires symmetric routing. Therefore, the message header can be modified by the procedure evoked at that stage and encoded in the simpler peer-to-peer model.

Since all procedures modify the header, it is possible tc say that each header performs a state transition when it goes through a switching element. The large number of header states, however, does not imply that the switching elemen logic to handle the header is more complex. The adopte solution based on header models and their correspondin procedures is appealing because it does not require th switching element to know the network stage to which i belongs to define the header modification procedure to b adopted, since each header model requires the use of limietd set of procedures regardless of the network stag Consequently, the same not so complex logic design i applicable to all network switching elements. In additio this design is greatly simplified because it depends on th number of different procedures and not on the number possible header states. For a better understanding of th header management procedures which are involved, Figu 7 describes the set of tasks that a switching element has perform when a particular header model is present in t received message.

Peer-to-Peer Routing	*Procedure*
$1S_1S_2S_3S_4S_5$	Routes the message according to the 1st S bit, shifts left the S bits and inserts at end of the header a bit A_1 of the source cluster vector identification
$1S_2S_3S_4S_5$ . . . . A_1	Same as before, but A_1 is shifted left before A_2 is inserted
$1S_3S_4S_5$ . . . . A_1A_2	Same as before, but A_1 and A_2 are shifted left before A_3 is inserted
$1S_4S_5$ . . . . $A_1A_2A_3$	Same as before, but A_1 to A_3 are shifted left before A_4 is inserted
$1S_5$ . . . . $A_1A_2A_3A_4$	Same as before, but A_1 to A_4 are shifted left before A_5 is inserted

Broadcast Routing	*Procedure*
000000	Routes the message to both outputs and inserts at end of the header a bit A_1 of the source cluster vector identification
000000 . . . . A_1	Same as before, but A_1 is shifted left before A_2 is inserted
000000 . . . . A_1A_2	Same as before, but A_1 and A_2 are shifted left before A_3 is inserted
000000 . . . . $A_1A_2A_3$	Same as before, but A_1 to A_3 are shifted left before A_4 is inserted
000000 . . . . . . $A_1A_2A_3A_4$	Same as before, but A_1 to A_4 are shifted left before A_5 is inserted

Multicast 1 Routing		Procedure
$0\,T_1T_2T_3T_4T_5S_1S_2S_3S_4N_1N_2\,.\,.\,.\,.$	If 1st T bit = 1 (sym.)	Routes the message according to the 1st S bit, shifts left the T bits inserting "1" in the last position*, shift left the S bits and inserts bit A_1 at the end of the header
	If 1st T bit = 0 (non-sym)	Routes the message to port #0 based on N_1 and port #1 based on N_2, generates a header without the N vector, according to the Peer-to-Peer model and inserts bit A_1 at the end of the header
$0\,T_2T_3T_4T_5\,1\,S_2S_3S_4\,.\,N_1N_2\,.\,.\,.\,A_1$	Same as before, but A_1 is shifted left before A_2 is inserted	
$0\,T_3T_4T_5\,1\,1\,S_3S_4\,.\,.\,N_1N_2\,.\,.\,A_1A_2$	Same as before, but A_1 and A_2 are shifted left before A_3 is inserted	
$0\,T_4T_5\,1\,1\,1\,S_4\,.\,.\,.\,N_1N_2\,.\,A_1A_2A_2$	Same as before, but A_1 to A_3 are shifted left before A_4 is inserted	
$0\,T_5\,1\,1\,1\,1\,.\,.\,.\,.\,N_1N_2A_1A_2A_3A_4$	T = 0 (only non-symmetric stages are left) Same as before, but A_1 to A_4 are shifted left before A_5 is inserted	

Multicast 2 Routing		Procedure
$0\,T_1T_2T_3T_4T_5S_1S_2S_3N_1N_2N_3N_4\,.\,.\,.$	If 1st T bit = 1 (sym.)	Routes the message according to the 1st S bit, shifts left the T bits inserting "1" in the last position*, shift left the S bits and inserts bit A_1 at the end of the header
	If 1st T bit = 0 (non-sym)	Routes the message to port #0 if $N_1 = 1$ or $N_2 = 1$ and to port#1 if $N_3 = 1$ or $N_4 = 1$, generates a header to each port reducing the N vector size, according to the Multicast 1 model inserts bit A_1 at the end of the header
$0\,T_2T_3T_4T_5\,1\,S_2S_3\,.\,N_1N_2N_3N_4\,.\,.\,A_1$	Same as before, but A_1 is shifted left before A_2 is inserted	
$0\,T_3T_4T_5\,1\,1\,S_3\,.\,.\,N_1N_2N_3N_4\,.\,A_1A_2$	Same as before, but A_1 and A_2 are shifted left before A_3 is inserted	
$0\,T_4T_5\,1\,1\,1\,.\,.\,.\,N_1N_2N_3N_4A_1A_2A_2$	T = 0 (only non-symmetric stages are left) Same as before, but A_1 to A_4 are shifted left before A_5 is inserted	

Multicast 3 Routing		Procedure
$0\,T_1T_2T_3T_4T_5S_1S_2N_1N_2N_3N_4N_5N_6N_7N_8$	If 1st T bit = 1 (sym.)	Routes the message according to the 1st S bit, shifts left the T bits inserting "1" in the last position*, shift left the S bits and inserts bit A_1 at the end of the S bits
	If 1st T bit = 0 (non-sym)	Routes the message to port #0 if $N_1 = 1$ or $N_2 = 1$ or $N_3 = 1$ or $N_4 = 1$ and to port#1 if $N_5 = 1$ or $N_6 = 1$ or $N_7 = 1$ or $N_8 = 1$ generates a header to each port reducing the N vector size, according to the Multicast 2 model inserts bit A_1 at the end of the header
$0\,T_2T_3T_4T_5\,1\,S_2A_1N_1N_2N_3N_4N_5N_6N_7N_8$	Same as before, but A_1 is shifted left before A_2 is inserted	
$0\,T_3T_4T_5\,1\,1\,A_1A_2N_1N_2N_3N_4N_5N_6N_7N_8$	T = 0 (only non-symmetric stages are left) Same as before, but copies the A bits to the end of the header before A_3 is inserted	

Fig. 7 The Header Models and Corresponding Procedures

The insertion of bits equal to "1" in the "*" situations indicated in Figure 7 ensure that the count of zeroes in the T vector of bits (5 bits) is equivalent to the number of remaining non-symmetric stages which determines the size of the N vector of bits in the header. Without this information, the behavior of each switching element would depend on its position in the network.

VI. BEHAVIORAL SIMULATION

To validate the overall proposal for the implementation of multicast communication from information stored as bit vectors in the distributed directories, a behavioral simulation has been performed using a C program that implements the complete functionality of the proposed solution.

As shown by Fig. 8, the simulation starts by considering the function which generates all the possible combinations of bit vectors that define the network ouput ports which will receive the message. Each combination is used by the header generator to create the message header that defines how the message will be routed through the network. The message is then transmitted from the specified network input port. The message transmission involves the header decodification and alteration at each switch in order to direct the message to the next stage.

When a message is received, the Network Interface returns an Acknowledgement message, which is sent through the network to the original message source cluster. When the Acknowledgement message is recognized at a network output port, the number of the cluster which sent the message indicates the bit that should be enabled in the "acknowledgement bit vector". In the end, the acknowledgement bit vector must be identical to the bit vector which was previously used for the header generation.

This complete procedure has been repeated for all possible bit vector configurations and correct results have been produced in all cases, which validates the proposal.

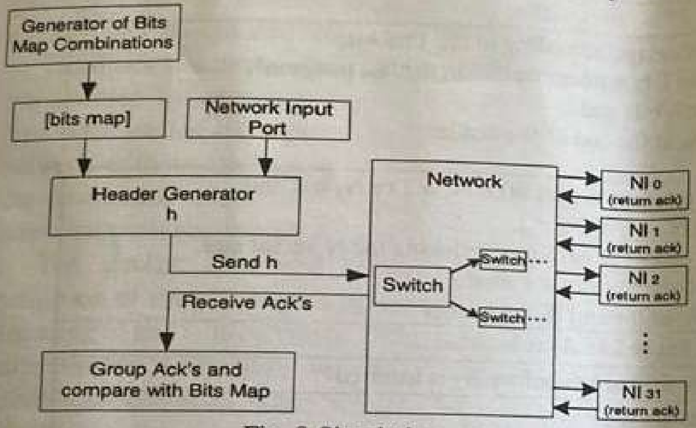

Fig. 8 Simulation

VII. Conclusions and Future Work

The redesign of the Multiplus multistage interconnection network to support multicast and broadcast messages required for an efficient implementation of directory based cache coherence protocols has been presented. In the proposed solution different message header models are used depending on the message type and on the number of symmetric network stages required to route multicast messages. The procedures for generating the message header from the directory bit vector information and for decoding and modifying the header at each switching element have also been described in detail. The network architecture can be configured with a duplication of its first two stages. With this expanded configuration, the maximum number of multicast messages to be sent in sequence to reach any combination of destination ports is reduced from 4 to 2. The additional hardware required for setting up this configuration for a network interconnecting 32 clusters represents 40% of the total hardware cost since two network stages are added to the previous five stages used in the original configuration.

A major benefit of the proposed solution for the implementation of hardware support to multicast in the Multiplus interconnection network is that the same logic design can be used for all the network switching elements regardless of its location in the network. The full switching element design has been described in VHDL and implemented with an Altera EPM7512B EPLD. The fifo buffers within the switching elements have been designed with six 16-bit inputs. The logic cell utilization rate within the EPLD has been 82% and the maximum bandwidth per network channel achieved has been around 30 Mbytes/s.

Future work will focus on the redesign of the Multiplus network interface to include the required hardware support to the proposed cache coherence protocols and the header generator circuit and on the overall performance evaluation of the cache coherence protocols with the addition of the hardware support to multicast invalidation messages.

Acknowledgements

The authors would like to thank CAPES, CNPq/RHAE and FINEP, Brazil, for the support given to the development of this research work.

References

[Aude96] "The Multiplus/Mulplix Parallel Processing Environment", J.S.Aude et al. – Proc. of the Int'l Symposium on Paralel Architectures, Algoritms and Networks (I-SPAN96) – Beijing, China, June 1996

[Bers93] "The Midway Distributed Shared memory System", B. Bershad, M. J. Zekauskas, W.A. Sawdon, Proc. of the IEEE COMPCON Conference, 1993, pp. 528-537

[Bron96] "Project and Implementation of a High-Performance Switching Element Using EPLD's", Bronstein, G., Proceedings of the XI Brazilian Microelectronics Society Congress, Águas de Lindóia, São Paulo, pp. 93-98, August 1996

[Ghar90] "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors", K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, J. Hennessy, Proc. of the 17th Int'l Symp. on Computer Architecture, May 1990, pp. 15-26

[Leno90] "The Directory-Based Cache Coherence Protocol for DASH Multiprocessor", D. Lenoski, et alii, Proc. of the 17th Int'l Symp. on Computer Architecture, 1990.

[Lin 94] "Deadlock-Free Multicast Wormhole Routing in 2-D Mesh Computers", X. Lin, P.K. McKinley, L.M. Ni, IEEE Transactions on Parallel and Distributed Systems, Vol. 5, N. 8, pp. 793-804, August 1994

[Moha96] "A Hardware Multicast Routing Algorithm for Two-Dimensional Meshes", P. Mohapatra, V. Varavithya, Proc. of the 8th Symp. on Parallel and Distributed Processing, pp. 198-205, October 1996

[Stun97] "Implementing Multidestination Worms in Switch Based Parallel Systems: Architectural Alternatives and their Impact" , C. B. Stunkel, R. Sivaram, D. K. Panda, Proceedings of the International Symposium on Computer Architecture, pp. 50-61, June 1997

[Vara99] "Asynchronous Tree-Based Multicasting in Wormhole-Switched MINs", V. Varavithya, P. Mohapatra, IEEE Transaction on Parallel and Distributed Systems, Vol. 10, No. 11, pp. 1159-1177, November 1999