Introducing Memory into the Switch Elements of Multiprocessor Interconnection Networks

Haim E. Mizrahi, Jean-Loup Baer, Edward D. Lazowska, and John Zahorjan

Department of Computer Science
University of Washington
Seattle, Washington 98195

Abstract

As VLSI technology continues to improve, circuit area is gradually being replaced by pin restrictions as the limiting factor in design. Thus, it is reasonable to anticipate that on-chip memory will become increasingly inexpensive since it is a simple, regular structure than can easily take advantage of higher densities.

In this paper we examine one way in which this trend can be exploited to improve the performance of multistage interconnection networks (MINs). In particular, we consider the performance benefits of placing significant memory in each MIN switch. This memory is used in two ways: to store (the unique copies of) data items and to maintain directories. The data storage function allows data to be placed nearer processors that reference it relatively frequently, at the cost of increased distance to other processors. The directory function allows data items to migrate in reaction to changes in program locality. We call our MIN architecture the Memory Hierarchy Network (MHN).

In a preliminary investigation of the merits of this design [8] we examined the performance of MHNs under the simplifying assumption that an unlimited amount of memory was available in each switch. We found that despite the longer switch processing times of the MHN, system performance is improved over simpler, conventional schemes based on caching.

In this paper we refine the earlier model to account for practical storage limitations. We study ways to reduce the amount of directory storage required by keeping only partial information regarding the current location of data items. The price paid for this reduction in memory requirement is more complicated (and in some circumstances slower) protocols. We obtain comparative performance estimates in an environment containing a single global memory module and a tree-structured MIN. Our results indicate that the MHN organization can have substantial performance benefits and so should be of increasing interest as the enabling technology becomes available.

1 Introduction

In this study a new architecture for shared memory multiprocessors, the Memory Hierarchy Network (MHN), is introduced and analyzed. The systems under study are medium scale multiprocessors where processors are connected to memory modules by a multistage interconnection network. The approach advocated here extends the memory hierarchy into the interconnection network, tailoring it to the specific needs of accessing shared variables. A hierarchy of memory elements is built into the switches of the interconnection network, and dynamic data positioning and routing protocols are introduced. The study focuses on the performance and cost of various realistic implementations of this idea.

The motivation for this study is that there may be a considerable performance penalty in existing systems when shared variables are heavily referenced. This problem will be aggravated by the need for bigger systems. Trends in VLSI technology will allow faster and denser CPUs and memory designs. The inter-chip communication times, however, will not be sped up in the same proportion. Because of these developments, traditional measures of cost, such as total memory space, will be replaced by design constraints such as interconnection costs and pin count. Thus, adding memory and some logic to the interconnection switches will become an attractive way to enhance system performance.

A key observation is that keeping coherent multiple copies of shared variables, in the context of a multistage interconnection network, is very traffic intensive. Therefore, the proposed scheme is based on keeping only a single copy of each shared writable variable in the system, and dynamically moving it in the extended memory hierarchy to adapt to changing access patterns. As there is only one copy of shared writable data, there cannot be any inconsistent data, i.e., the coherency problem does not exist. Therefore, this study does not provide "yet another coherency protocol". Rather, the main questions addressed here are when and where to migrate a data object, as opposed to when to create a new copy or when to invalidate or update an existing one.

The material presented here has two major parts. The first part reviews the background and recent relevant studies, discusses the motivation for a new architecture, and briefly presents the main ideas. In the second part, the performance of a tree network with practical amounts of

memory in switches is analyzed in relation to the implementation cost.

2 Architecture Models

In this section the models of the parallel systems used in our study are presented. First, for the purposes of our performance comparisons, we define a baseline system in which shared writable data is not cached. We then present a "standard improvement" to that system that allows caching and uses a single, central directory to maintain coherency. Finally, we present an overview of the general MHN architecture and identify four specific versions of particular interest.

2.1 The baseline Processors-Caches model

The baseline architecture will be referred to as the Processors-Caches (PC) model. It contains N (a power of two) identical processors and associated local caches, a global memory, and a multistage interconnection network of depth log N.

In our model the caches are used exclusively for private and read-only shared data, as well as (non-writable) code. Accesses to local caches always result in hits satisfied in a single processor cycle. In other words, we assume a fast, conflict-free conventional multistage network that is transparent to the architecture.

Global memory is used for shared writable data. It is accessed through the network, and a cache is placed between each memory module and the network to speed accesses. (Note that there is no coherency problem associated with a single cache located at this port.)

To simplify our feasibility study, we decompose the model by assuming that there is only a single memory module. Therefore, our interconnection network is a complete binary tree with the global memory (and its cache) at the root and processors at the leaves. Because the single module case is not preferential to either the MHN or conventional network designs, the comparative results obtained for this model should be applicable to systems with larger numbers of memory modules.

We assume that read requests to global memory are synchronous, that is, the issuing processor waits for the result. The use of the interconnection network, memory access latency time, and potential memory contention cause the processor to remain idle for some period of time during this request. In contrast, write requests are assumed to be asynchronous: the issuing processor immediately resumes computing after the request packet is placed on the network.

We define the processor cycle time to be equal to one. The relatively simple switches in our PC model are also assumed to have unit cycle times. The global memory module has an access time of four.

2.2 The directory-based scheme

A clear weakness of the PC architecture is that all accesses to writable shared data must be sent to the global memory module. Performance may be improved through the use of "directory schemes" [1]. This involves caching data items at the processors and using a single, central directory located at the root to maintain coherence. In the simplest directory scheme, which we call DIR, only a single copy of each shared data block is kept in the system. (Note that this policy corresponds to policy "Dir1NB" from [1].) As will be seen shortly, the MHN also maintains only a single copy of each data item. Thus, the comparisons between DIR and the MHN serve to isolate the contribution of the dynamic routing capabilities of the latter.

Because the switches required by the DIR scheme are not substantially more complicated than those required by PC, we assume the same cycle times for them. This is a slightly optimistic assumption for DIR, and so serves to understate somewhat the performance advantages of the MHN design demonstrated in Section 4.

2.3 The MHN architecture

As in the DIR scheme, data items in the MHN architecture may move dynamically from one memory to another (although only a single copy of a data item can be present in the system at a time). However, the MHN extends the DIR scheme in two ways. Firstly, switches of the MHN can hold data. Thus, data can be present not only in the global memory module and the caches located at the processors, but also at intermediate stages of the interconnection network. Secondly, MHN switches contain directories indicating the location of items stored in the subtrees for which they are the roots. Thus, the information in the single directory of the DIR scheme is partially replicated and distributed among the interconnection network switches of the MHN.

The exact manner in which data movement takes place in an MHN is controlled by a data migration policy. In the selection of an appropriate migration policy for use in the MHN there are two dimensions to be considered: "when should data be migrated?" and "how far toward the referencing processor should it be migrated?". A family of answers to the question of "when" is given by "each time the last j references are from the same processor" for differing values of j. For example, for j=1 data items are moved on every reference, while for j=2 two successive references to the item must come from the same processor before migration takes place. Similarly, a family of answers to the question of "how far" is given by "k steps" for various values of k. Here obvious choices for k are 1 (one step toward the referencing processor) and ∞ (all the way to the referencing processor). We introduce the notation MHN/j/k to denote the MHN policy that moves a data item k positions after j consecutive references by the same processor.

Intuitively, appropriate values for parameters j and k of the migration policy relate to the assumptions made about the "burstiness" of workload. A workload is considered bursty if it exhibits alternating periods of high and low frequency of access to individual data items. (In contrast, the workload behavior is considered to be "random" if the frequency of access to an individual data item is relatively constant over time.) A workload becomes increasingly bursty when, other factors (in particular, overall average reference rate) held fixed, either the length of the high frequency peri-

ods increases or the access rate during the low frequency periods decreases.

Parameter k of the migration policy relates to the assumed length of a burst. The longer a burst is likely to be, the more advantageous it is to move data towards the referencing processor despite the fact that this moves it away from many other processors. Thus, parameter k should be large for bursty workloads and small for random workloads.

Parameter j relates to the low frequency reference rate. It is used to detect when a burst has begun. Some references to a data item are made even during periods of overall low frequency. It is counter-productive to migrate a data item in response to these accesses. For a very bursty workload, however, these low frequency period accesses are rare. Thus, for a bursty workload j can be small, that is, it is safe to assume that (nearly) all references to the data item indicate the beginning of a high access frequency period.

In the work presented here, we have chosen to evaluate four specific policies:

1. MHN/1/1: Move the data one step on each reference. Here a step is one edge in the path from the current location towards the processor that issued the request.

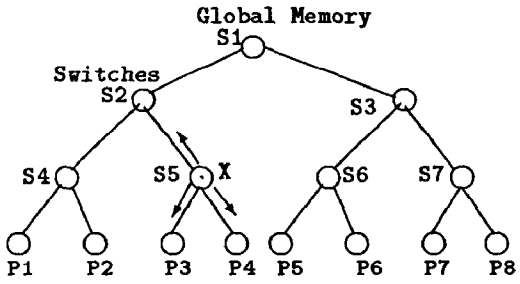

Figure 1: Data Migration in a Tree MHN/1/1 Architecture

   Consider the example shown in Figure 1. The data block X, located in switch $S5$, will move to switch $S2$ if the next reference to it is made by a processor in the set $\{P1, P2, P5, P6, P7, P8\}$. If the reference is made from $P3$ or $P4$, it will move to the local cache of the referencing processor.
2. MHN/2/2: Move the data two steps after two successive references from the same processor. This policy has almost the same "speed of migration" as the previous one but is more conservative in its estimate of when a burst has begun.
3. MHN/1/∞: Move the data all the way to the requesting processor on each reference. Note that while a requested data item is never migrated into a switch memory other than at the processors, all switches contain some data storage. This storage is used to "bubble up" replaced data items that arise when a lower level memory is full and a new item must be migrated there.
4. MHN/2/∞: Move the data all the way to the requesting processor on each two successive references from the same processor.

The performance evaluation of these alternatives is presented in Section 4. For brevity in what follows, whenever a detailed description is needed we will demonstrate the operation of the MHN/1/1 policy.

As will be seen in the next section, implementation of an MHN requires switches that are more complicated than those needed for the PC or DIR designs. Thus, in our performance evaluation of MHN we assume a switch cycle time of two, i.e., twice the cycle time of the switches in the simpler networks. (The one exception is that the MHN switches connected directly to the processors are relatively simple, acting as normal caches. Thus, we keep the unit cycle time assumption for those switches.)

3 The Design of MHN Networks

3.1 Switch protocols

In this section possible designs for MHN switches are discussed and their relative complexity and performance are compared. The complexity of these switches can be substantial, mainly as a result of the dynamic routing capabilities. Therefore, attention is focused on cost/performance tradeoffs in the implementation of the directories.

The basic operation of a switch consists of three tasks. First, as part of the global but distributed shared *memory*, the switch controls accesses to the data currently held in its local store. The switch data memory acts as a conventional memory module in a global memory system. However, because the position of data changes dynamically, the local data memory is accessed associatively, like a cache.

Second, as part of a global but distributed *directory*, the switch performs routing of requests from processors to memory. Arriving request packets are routed according to information on the current location of the requested data. Since routing decisions are made locally, a switch needs to hold enough information to uniquely select an appropriate port to forward the request. In contrast to a global directory approach, the routing information in a switch need not be updated on every movement of the data, but only when there is a change in the port through which the data is accessible, i.e., when the data passes through the switch.

Finally, as part of a conventional multistage interconnection network, the switch performs "static" routing of packets on the return path from memory to processors.

3.2 Switch structure

Figure 2 depicts a schematic block diagram of an MHN switch. The major components are:

1. Input and output buffers, implemented as First In First Out (FIFO) queues. The external inter-switch links can carry only one packet at a time. Whenever a destination buffer at a switch is full, transfers to it are stalled.
2. I/O ports. Each switch has six unidirectional ports, two for each of its neighbors ("Up", "Left" and "Right"), allowing full-duplex communication. The data paths of the inter-switch links are wide enough to handle one packet in a cycle.

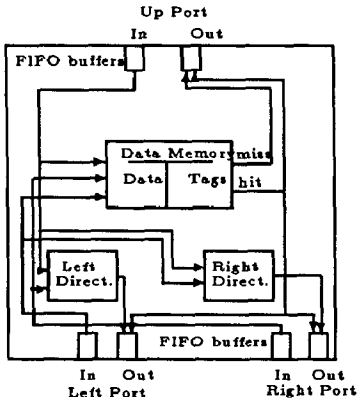

Figure 2: Basic MHN Switch Structure

3. Internal busses. The internal busses and logic can transfer a packet from each input port to any output queue in a switch cycle. Separate parts of the output queue allow insertion of multiple packets into it, thus avoiding contention when more than one packet is routed to the same port. The head of the output queue is one of the heads of its sub-queues, selected randomly by arbitration logic. (Similar assumptions have been made for almost all interconnection network performance evaluations, and a switch design that achieves this degree of parallelism was designed for the NYU architecture [3].)
4. Data memory. This is the part of the global shared memory that is currently located in the switch. It is organized like a cache. The size of the data memory depends on the size of the global memory, the technology used, and the switch's level in the network. The appropriate amount of memory at each level depends on the migration policy. For the MHN/(1 or 2)/∞ policies, for example, it will be reasonable to allocate a substantial portion of the memory to the leaves (processors) and to the levels near the root.
5. Routing directory. This part of the switch holds dynamically updated routing information. As routing is performed locally in each switch, the relative position ("Up", "Left", or "Right") of the data is sufficient. An important aspect of this work is to assess the complexity and the performance implications of various directory implementations. Two possible approaches are valid: (1) a single directory, which is accessed for routing all the packets through the switch, or, (2) separate subdirectories, each holding routing information for the data accessible through a particular external port. (Figure 2 depicts the latter organization.) Note that a separate directory for data stored locally is not necessary, as "hits" or "misses" on the local data store provide this information explicitly.

3.3 Basic switch operation and protocol under optimistic assumptions

The simplest switch protocol is based on two assumptions: that the data memory in each switch is large enough to contain all the shared data, and that the switches' directories hold "full knowledge" (i.e., they include routing information for *all* data blocks that are located in all descendants of the switch). Under these optimistic conditions: (1) data entries are never replaced, (2) routing information is never lost because of lack of space in a directory, (3) broadcasting is not needed, and (4) requests that cannot be serviced locally are forwarded through one port only. The protocol assures that whenever a data block is stored in a switch or transferred through it, the appropriate entry in the directory at the switch is updated. For data blocks that have never been transferred through a switch, the default information ("Up", as initialized) correctly indicates that packets referring to this data should be forwarded upwards.

The "full knowledge directory" switch protocol distinguishes among the following five types of packets: Read (issued when a processor loads a shared writable data item), Write (issued when a processor stores a shared data item), Answer (issued by a switch to forward data in response to a Read request), Answer Migrate (like Answer, but carrying a migration counter to support MHN/j/k policies), and Write Migrate (like Write but carrying a migration counter).

The assumption of unlimited memory space on which this straightforward protocol is based is not tractable for large systems. In practice, only limited memories are available and therefore misses on both the data store and the directory must be addressed. Replacement policies for data and directory entries, and routing in cases when information on the location of the data is not available, need to be specified. These modifications to the basic protocol are discussed in the next section.

3.4 Alternative directory organizations

In this section we discuss possible directory organizations. The possibilities fall into two groups. In the first, the "full knowledge" approach, each directory keeps information sufficient to forward on the correct port an arriving request for any data item. These organizations admit relatively simple routing protocols but require extensive directory memory.

The second approach requires that directories keep information on only some subset of the global data items. Here memory requirements are reduced at the price of more complicated protocols in the event that no information on the requested data item exists in the directory.

3.4.1 Full Knowledge Bit Map directories (FKBM)

A simple way to keep full routing information is to hold in each switch a full bit-map table, with an entry for each block in the shared memory subsystem. For the dynamic routing protocol in a tree network two bits are enough to encode the three possible relative positions of each data block ("Up", "Left", or "Right"). A fourth logical location information, "Local", is not kept explicitly in the directory, but is available to the router by checking the local data memory.

The obvious advantages of this approach are the simplicity of the protocol and the overall performance that all full knowledge schemes exhibit. By keeping a full bit map, the directory entry corresponding to each data block can be directly accessed and there is no need for associative search. By keeping full routing information for every block, no broadcasting is ever needed. The disadvantages of this approach are the size of the directories needed and the lack of scalability. The directory size in each switch grows linearly with the size of the global memory for shared data. Bigger directories will not only increase the cost of the implementation but will also mean slower switches, as the time to perform a memory access is a function of the memory size.

3.4.2 Full Knowledge Set Associative directories (FKSA)

An alternative full knowledge organization can be obtained by observing that each switch need keep track only of those data items that currently reside in the memories of descendant switches. Given this "inclusion property" [2], a very simple protocol suffices to effect correct routing: on a directory "hit", the packet is routed either "Left" or "Right" according to the routing information in the directory; on a "miss", the request is forwarded "Up".

As compared with FKBM, under FKSA the directory memory requirement of each switch is reduced from linear in the total amount of global memory to linear in the amount of memory contained in its descendants. Nonetheless, this still represents a substantial number of directory items in at least some switches (those near the root). To make manageable the task of building hardware to support these directories, a set associative scheme is used.

Let us denote by $A_{dir}(i)$ the associativity of the directories of switch i at the l_i^{th} level in the tree, by $A_{data}(i)$ the associativity of the data memory, and by $S_{dir}(i)$ and $S_{data}(i)$ the number of sets in the directory and data memory respectively. Then it can be proven ([2], [7]) that the inclusion condition requires

$$A_{dir}(i) \geq \sum_{j \in Des(i)} A_{data}(j) * \max(1, \lceil \frac{S_{data}(j)}{S_{dir}(i)} \rceil) \qquad (1)$$

Note that the number of sets in the data memory and the directories need not be fixed over the entire network. The last factor in the equation above takes into account the ratio of the number of data sets to the number of directory sets. If the number of sets in the (parent) directory is bigger than the number of sets in the (descendant) data memory, the ratio is less than 1, and the associativity of the directory "includes" room for all the entries in the data memory. If the directory has fewer sets than a descendant data memory, several sets of the descendant will fall into the same set in the parent directory, and the associativity is increased to reserve enough room for all the data blocks. Note that the maximum in the equation is taken for each level independently, guaranteeing that there is enough room for the routing information of each data block in each descendant level independently, regardless of the organization in other descendant levels.

3.4.3 The full knowledge data replacement protocol

Both the FKBM and FKSA directory organization are based on finite data memories in the switches. This finiteness causes a "data replacement" whenever a migration packet tries to write into a filled data set. Some modifications to the basic protocol are therefore necessary to handle the cases of data migration. The only cases when a directory entry is replaced is when a *local data store* in a switch has no place in the appropriate set.

The protocol makes use of an additional type of packet, labeled "DataMigrate" packets, that will be used to migrate a data block when a replacement occurs. Writing new routing information in an unfilled set of a directory poses no problems. Writing new routing information into a filled set requires more care. To avoid any loss of routing information when the location information of a migrating packet needs to replace an existing entry in the directory, the protocol limits the migration of data blocks into its sub-trees. Since there is always enough room in each directory for all data blocks in its descendant switches, a full directory set implies that the corresponding sets in the data memories in the descendant switches are also full, and local data replacement will take place. In these cases, the data replacement protocol will free a line (an entry in the local data set), making room for the new data entry. The protocol makes use of an "overflow" buffer to temporarily store routing information for added entries. During this time, additional data migrations whose directory entries fall in the same set are disallowed. The overflow buffer holds the routing information for data blocks that are currently being inserted into the sub-tree, until a data block is migrated out of the sub-tree to free a directory entry. When the "overflow" buffer is full, no additional migrations are allowed. (Overflow buffer entries are removed when the replaced data arrives at the switch).

3.4.4 Partial Knowledge Set Associative directories (PKSA)

In this section, the full knowledge conditions are relaxed, allowing switch directories to be smaller at the cost of more complicated request routing protocols. This leads to the design of limited set associative directories, holding only partial routing information.

The organization of PKSA follows that of FKSA, with the difference that the associativity is not required to increase with the level in the tree. This means that a PKSA switch is incapable of keeping directory entries for all data items in descendant switches. When a switch decides to add a new directory entry and the set into which that item falls is full, the switch simply discards one of the existing entries. There is no need for the switch to notify either the parent or descendant nodes of this action.

Clearly, PKSA does not satisfy the inclusion or full knowledge conditions. However, a modified form of the inclusion property does hold: although blocks can be located in the sub-tree and not have a valid entry in the directory, all the (valid) entries in the directories refer to data blocks located in the sub-tree. Therefore, no blocks that are not located in the sub-tree can have a valid entry in the directory.

An additional requirement of PKSA networks needed to make the data location protocol work is that the root directory *is* inclusive, that is, it contains routing information for all data items held by its descendants. In this sense, PKSA resembles the DIR scheme. However, there are two important differences. First, while the number of directory entries at the root under PKSA is the same as that for DIR, the amount of information per item under PKSA is substantially smaller. This is because PKSA stores only *routing* information (i.e., "Left", "Right", or "Up"), while DIR must store *location* information (i.e., a processor address). Second, because DIR stores location information, the root must be notified *every* time a data item changes location. In contrast, the root of a PKSA network needs to update its routing information only when the data item is migrated through the root (from one subtree to the other).

The protocol used to handle directory misses takes advantage of the modified inclusion property of PKSA networks. The protocol has two phases: an initial search up the tree until a directory is located that contains information on the data item, followed by a search down the tree to locate the item itself. The search up the tree is performed using SearchUpRead and SearchUpWrite packets. A switch receiving such a packet and neither storing the data item nor having a directory entry for it simply forwards the packet to its parent. We are guaranteed to reach eventually a directory holding routing information for the data item because of the pure inclusion property of the root directory.

Once routing information has been encountered, a downward search for the data item begins. If the switch with the routing information does not actually contain the data item, it sends a Read or Write packet as appropriate down the port indicated by its directory entry. Each subsequent switch receiving a Read or Write packet (and not containing the data item) interrogates its directory for further routing information. If such information is present, the packet is simply forwarded. If there is no directory entry present for the data item, we cannot know which subtree the data item is in. Thus, in this case the switch sends a BroadcastRead or BroadcastWrite packet down both the Left and the Right port.

The broadcast mechanism assures that the data referred to in the packet will be located so long as it is resident in the sub-tree where the broadcast takes place. A problem may arise when a broadcast packet (*bp*) references a data block that currently is being migrated out of the sub-tree to which *bp* is being sent. This can be solved by searching through a buffer holding recently sent broadcasts (or migration packets). Each broadcast (migrating packet) is placed in the buffer as soon as it is inserted in the output queue, and is removed from this buffer only after an acknowledgement message (BroadcastAck) is received.

4 The Performance of MHN Networks

In this section, the performance of systems built with limited data memories and partial directories is examined. A controlled set of experiments, in which the original assumptions concerning the memories in the systems are gradually relaxed, is conducted. By comparing the performance of the different designs, an assessment of the major consequences of the various organizations is possible:

1. The effect of *data* misses and replacements is evaluated by comparing the performance of the full knowledge MHN network with unlimited memory space to the performance of full knowledge MHN networks with limited data memory.
2. The effect of limited *directory* memory is evaluated by comparing the performance of FKSA to PKSA.

4.1 Workload characterization

It is clear that the workload used in comparing interconnection network architectures can have a strong influence on the results. For example, a (perhaps artificial) workload exhibiting little or no locality of reference will tend to favor a very simple network built out of fast, dumb switches over a network with smarter, slower switches.

Unfortunately, measurement data about the behavior of real workloads is scarce [4], and so it is not possible to make performance comparisons using "a typical, live workload". Because of this, a flexible abstract model of reference patterns is adopted that allows, through varying parameterizations of a single basic workload model, the exploration of the interconnection network performance over a wide spectrum of possible program behaviors.

The analytical workload model is specified by five parameters:

$$< Shared, AverBurst, NoObjects, Contention, Write >$$

The first parameter, *Shared*, is defined as the fraction of a processor's references that are to shared writable data out of the total number of memory references (both private and shared) that it makes. As only shared references are placed on the MHN network, this parameter controls the overall load on the network.

In comparing conventional interconnection networks and the MHN, we are particularly interested in the impact of locality. Our locality measure, *AverBurst*, reflects the tendency of a processor to reference repeatedly shared variables during a relatively short period of time. It specifies the average length of a burst of references (that is, the average number of consecutive shared references to the same data item.) When one burst ends, the next burst to begin is for a data item that is chosen at random among all the shared writable data items. The number of such items is given by *NoObjects*.

The third parameter, *Contention*, is defined as the fraction of memory references generated by a processor to randomly chosen shared data objects. Each processor is considered to have at all times a set of current "burst variables" which it references with probability $(1 - Contention)$ on each shared variable reference. With probability *Contention* a shared variable reference is made to some other data item, chosen uniformly. These accesses are not included in the burst produced by this processor, but rather represent occasional references to other global data items which are accessed from within a burst.p

Parameter *Write* is defined as the fraction of accesses to shared data that change the value of the accessed data item.

While it is possible to choose different values for these parameters for each data item and processor (as appropriate), a homogeneous system, in which a single parameter value applies to all processors and data items, is assumed. This simplifies the interpretation of results, as well as the construction and manipulation of the models.

To summarize, we briefly describe the processor behavior in terms of workload model parameters. On each cycle, each processor generates a single reference (provided that it is not waiting for a pending memory request). This reference is either to a private or to a shared data object, in the proportion specified by the *Shared* parameter. Each shared request is either a read or a write, with proportions that are specified by the *Write* parameter. To generate the address of the request, a data item is first selected following the *Contention*, and *AverBurst* parameters. When generating a *Contention* request, this item is picked at random uniformly from the appropriate set of data items. When generating a burst request, the likelihood of continuing the one of the current bursts from the given processor is specified by *AverBurst*.

4.2 Performance evaluation methodology

We have used simulation to obtain performance estimates because of the complexity of the mechanisms being investigated. However, there is a basic limitation to this approach in this domain. So long as we have infinite memory in the switches, the simulation results are insensitive to the number of data objects in the global shared memory. For the infinite memory case, different data objects do not interact and the presence of a data item in a given switch is not affected by references made to other data items. In the finite memory case this is no longer true, since the migration of one data item can cause the replacement of another one. Because of a practical limitation on the simulation run's CPU time, we could not simulate systems with realistic memory sizes. However, it is common practice in studies of this type to run simulations with a limited number of data objects to efficiently produce approximate results. A "scaled down" system is used in the simulation in which the size of the data memory and directory in the simulated system is proportional to the size of real systems. We have checked the sensitivity of our results to the number of data objects in the global shared memory over the range in which it was still feasible to simulate a 128 processors system. The changes in the performance metrics were minor. Unless otherwise stated, the results reported below are a system with 128 processors and 512 shared writable data items.

4.3 The effect of limited data memory

In this section we evaluate the effect of limited data memory. The question to be answered is the extent to which the limitation of the data memory degrades performance due to longer access times (since less data can be stored close to the processors) and the increased network traffic generated by data migrations needed for replacements. We assess this impact by comparing limited data memory versions to unlimited data memory versions, with unlimited directory memory in each case. (The performance of MHN with both limited data and directory memories, as well as a comparison of MHN performance with the simpler PC and DIR designs, is considered subsequently.)

Figure 3 displays the effect of finite data memories on the four MHN architectures for two locality patterns (*AverBurst*= 2, 20) as a function of memory size. (A memory size of '1' indicates that a total storage equivalent to 1024 data objects is distributed among all the switches in the network.) In our examples there are 512 shared writable data objects (i.e., *NoObjects*=512). Directories are organized as FKSA. The total data memory has been allocated among the MHN switches in an attempt to equalize their utilizations, that is, the ratio of the number of valid entries to the capacity.

The Y-axis of Figure 3 is the ratio of the effective processing power of the system with limited data memory to that of the same system with unlimited data memory. We call this ratio a "speedup gain".

It is interesting to compare the sensitivity of different MHN architectures to the limitation of memory space, as a function of the locality in the access pattern. As can be seen, the MHN/1/1 architecture is the most sensitive to the allocation of memory in the switches, while MHN/2/∞ and MHN/1/∞ are the less sensitive . This is expected, as these latter policies do not allocate data in the memories of the network switches. Thus, data objects which are no longer in use percolate slowly towards the root, becoming more accessible to a new user (processor). In fact, because of this phenomenon the performance of more limited memory space systems can be better than that of less limited ones. This is observed for the unit memory configurations, which outperform configurations richer in memory space. Generally, the degradation in performance due to limited memory exhibits a sharp knee. While the 6 memory unit configuration has essentially the same performance as the unlimited memory case, and the degradation due to a 2 memory unit limitation is between about 40% of speedup in the MHN/1/1 case, reducing the data memory further, as 1.0 or 0.1 units, causes a steep degradation.

4.4 The effect of limited directory memory

In this section the effect of limited associativity in the directories is studied.

To assess the relative effect that the restricted directory organization has on performance, we checked the performance under the PKSA protocol, which includes the search, broadcast and broadcast-acknowledgement phases. We measured the degradation relative to the FKSA case. We also recorded the number of broadcast packets, the level at which they were introduced, and the level at which they were consumed.

Figure 4 displays the expected performance of MHN/1/1 and MHN/2/∞ in their PKSA implementation. (Other MHN policies behave similarly.) The speedup is compared to that of the same architectures with full knowledge directories and the same data memory capacity. We note that for small localities (small burst lengths), performance is almost unaffected by limited directory space. This insensitivity is the result of the fact that for both policies data items never migrate very far from the root.

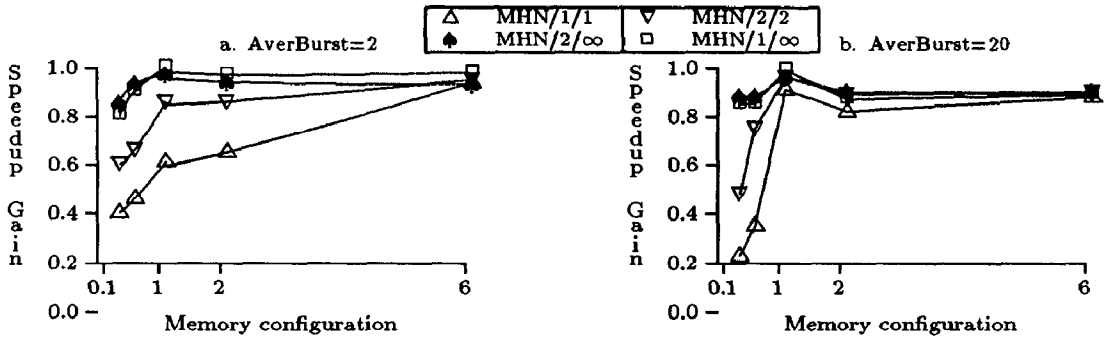

Figure 3: The Effect of Limited Data Memory

For higher localities (longer burst lengths), data is more spread out throughout the MHN. Thus, directory information is more important, resulting in some noticeable differences between the limited and unlimited directory memory schemes. The degradation caused is quite modest, however, never exceeding 20%.

Comparing the different migration policies, we observe that the MHN/1/1 architecture is much more sensitive to the organization of the directories than MHN/2/∞. This is intuitive, as MHN/x/∞ policies make much less usage of the data memories of the network switches. The MHN/2/∞ policy performs well because it is selective about which data objects should be placed in the (limited) data memories. Data blocks that are randomly accessed are not migrated, and therefore, the data memory holds only those objects that have a high probability of being accessed in bursts. So long as the capacity is sufficient to contain this active working set the performance is good.

4.5 The bottom line

We conclude this section by exploring the performance of MHN architectures with limited memory relative to the performance of PC and DIR architectures. This will be the "bottom line" of the expected performance, incorporating the degradation due to all the restrictions implied by limited data memories and practical partial directories. All the sources of overhead discussed above (data replacements, searching and broadcasting) are included in this evaluation.

Figure 5 shows the ratio of the effective processing power under a number of organizations to that obtained by the basic PC scheme. As can be seen, except for workloads exhibiting essentially no locality, the expected performance of MHN/1/1, MHN/2/2, and MHN/2/∞ is very good compared to both PC and DIR. All of the MHN architectures exhibit substantial performance gains, but the MHN/x/∞ schemes are clearly superior. This is an indication that much of the performance benefit of the MHN comes from the dynamic routing capability provided by the switch directories rather than from the ability to store data in intermediate levels of the network. Quantitatively, MHN/2/2 exhibits a factor of 2.2 to 3.2 improvement over PC, and MHN/2/∞ a factor of 2.2 to 4.0 (for the longer burst length). However, the MHN/1/1 scheme presents little advantage (and even a potential degradation at light loads) over the much simpler DIR scheme, making the cost effectiveness of this policy questionable.

5 Conclusions

The main conclusion of the feasibility study is that the MHN approach, based on the inclusion of data memories and dynamic routing capabilities in the switching elements of the interconnection network, is a promising one. The use of sophisticated switches in the implementation of the network for shared data was demonstrated to offer a significant performance improvement.

The price for this performance gain is more complex switches, with a substantial amount of memory. The implementation of such switches requires an ambitious VLSI design. In particular, large and fast memories are required in each switch for directory (and possibly data) storage. Based on the trends in VLSI technology, we believe that such switches will become feasible in the near future.

Specifically for the MHN, an obvious observation is that most of the complexity of the switch structure stems from the directories used to locate the data items. Recall that the dynamic routing capability is the major reason for observed performance gain. Thus, the price paid for implementing the dynamic routing cannot be avoided. The simulation results show that, under the wide set of assumptions made in the workload model, the dynamic routing capability of the switches contributed most of the performance gain, while the introduction of memory in the switches was found to be beneficial in only a limited domain.

Another important result can be inferred from the performance of the deferred migration policies (MHN/2/x), which performs substantially better than the commonly used "copy on the first reference". We claim that this has general applicability in the domain of multistage networks, where the cost of indiscriminate copying is high. Whenever there is only a single copy for each shared variable, its location becomes very important. Therefore, the decisions of whether to migrate a data item, and where to migrate it, should correlate with workload behavior. The workload used in our study exhibits bursts that are reliably indicated by two successive references from the same processor. Mapping this as a decision rule into the migration logic of each switch has little ramification on the hardware, but requires additional memory space for state information. The results show that this additional cost is well rewarded, in terms of improved performance.

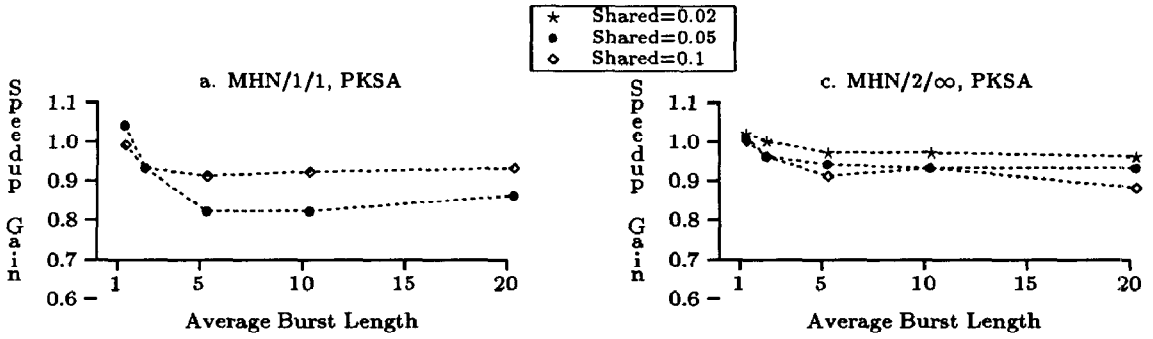

Figure 4: The Effect of Limited Directory Memory

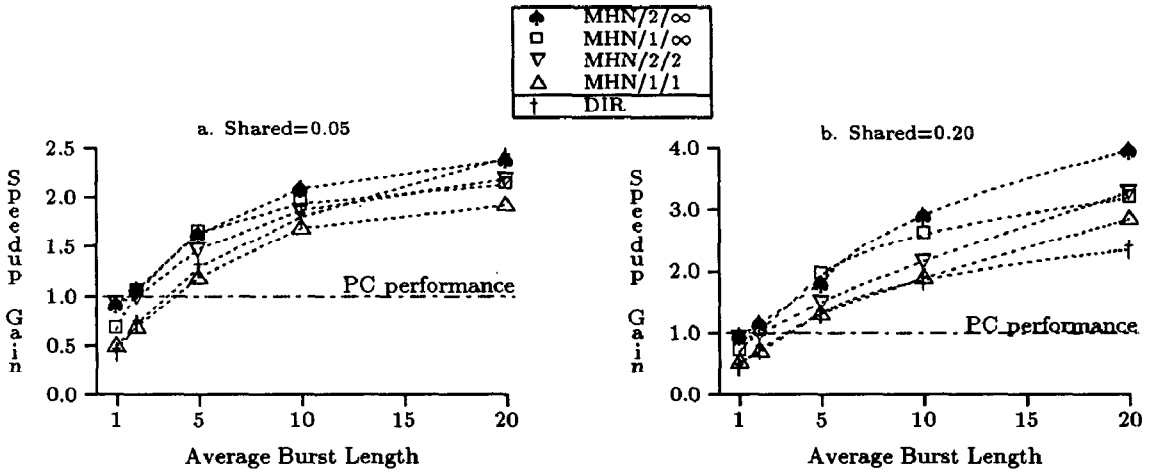

Figure 5: "Bottom line" (PKSA) Performance

Acknowledgements

This material is based on work supported by the National Science Foundation (Grants DCR-8352098, CCR-8619663, CCR-8702915, and CCR-8703049), the Naval Ocean Systems Center, U S WEST Advanced Technologies , The Washington Technology Center, and Digital Equipment Corporation (the External Research Program and the System Research Center). Additional partial support for this work was generously provided by Bell Communications Research, Boeing Computer Services, Tektronix, Inc., the Xerox Corporation, and the Weyerhauser Company. The Centre National de la Recherche Scientifique, France, and Laboratoire MASI, University of Paris 6, provided generous support and resources for Zahorjan for the year sabbatical leave during which this work was performed.

References

[1] Agarwal, A., Simoni, R., Hennessy J. and Horowitz, M. "An Evaluation of Directory Schemes for Cache Coherence". In *Proc. 15th Int. Symp. on Computer Architecture*, pages 280–289, 1988.

[2] Baer,, J.-L. and Wang, W.-H. "On the Inclusion Property for Multi-Level Cache Hierarchies". In *Proc. 15th Int. Symp. on Computer Architecture*, pages 73–80, 1988.

[3] Dickey, S., Kenner, R., Snir, M. and Solworth, J. "A VLSI Combining Network for the NYU Ultracomputer". *Ultracomputer Note 85*, June 1985.

[4] Eggers, J., and Katz, Randy H. "A Charaterization of Sharing in Parallel Programs and its Applicability to Coherency Protocol Evaluation". In *Proc. 15th Int. Symp. on Computer Architecture*, pages 373–382, 1988.

[5] Goodman, J.R., and Woset, P.J. "The Wisconsin Multicube: A New Large-Scale Data-Coherent Multiprocessor". In *Proc. 15th Int. Symp. on Computer Architecture*, pages 422–433, 1988.

[6] Kruskal, C.P. and Snir, M. "The Performance of Multistage Interconnection Networks for Multiprocessors". *IEEE Trans. on Computers*, pages 1091–1098, December 1983.

[7] Mizrahi, E. Haim. *"Extending the Memory Hierarchy into Multiprocessor Interconnection Networks"*. University of Washington, Dept. of Comp. Sci., Ph.D. Dissertation, Technical Report 88-11-03, November 1988.

[8] Mizrahi, H.E., Baer, J.-L., Lazowska, E.D., and Zahorjan, J. *"Extending the Memory Hierarchy into Multiprocessor Interconnection Networks: A Performance Analysis"*. University of Washington, Dept. of Comp. Sci., Tech. Report 88-11-10, November 1988.