
Hierarchical Directory-Based Shared Memory Architecture

M. J. KUMAR, S. VENKATESH, D. KIERONSKA AND L. M. PATNAIK¹

Department of Computer Science, Curtin University of Technology, Bentley, WA 6102, Australia

¹*Microprocessor Applications Laboratory, Department of Computer Science, Indian Institute of Science, Bangalore 560 012, India*

Email: kumar@cs.curtin.edu.au

We describe the design of a directory-based shared memory architecture on a hierarchical network of hypercubes. The distributed directory scheme comprises two separate hierarchical networks for handling cache requests and transfers. Further, the scheme assumes a single address space and each processing element views the entire network as contiguous memory space. The size of individual directories stored at each node of the network remains constant throughout the network. Although the size of the directory increases with the network size, the architecture is scalable. The results of the analytical studies demonstrate superior performance characteristics of our scheme compared with those of other schemes.

Received May 14, 1994; revised March 29, 1995

1. INTRODUCTION

Recently, distributed shared memory systems have been introduced to overcome some of the deficiencies of message passing and shared memory systems. Message passing systems have efficient communication networks and are scalable. However, message passing computers are hard to program and partitioning data in such systems is very difficult. On the other hand, shared memory systems are easy to program as they are based on single address space global memory concept. Shared memory systems suffer from very poor scalability. Distributed shared memory systems incorporate positive features of shared memory and message passing paradigms, and at the same time overcome some of the drawbacks of these systems. In other words the distributed shared memory paradigm enables shared memory view of a loosely coupled distributed memory system. The DASH prototype (Lenoski *et al.*, 1990) developed at Stanford University and the KSR-1 machine manufactured by Kendall Square Research (Ramanathan and Oren, 1993) are examples of such systems. The DASH prototype belongs to the non-uniform memory access (NUMA) class and makes use of the directory-based protocol for maintaining cache coherence, whereas the KSR-1 employs the cache-only memory access (COMA) scheme (Hwang, 1993).

The DASH is a high-performance machine with single address space and coherent caches (Lenoski *et al.*, 1992). In the DASH machine reduced memory latency is achieved by caching of memory including shared writable data (Lenoski *et al.*, 1990). Lenoski *et al.* (1992) have clearly identified the merits of directory-based cache-coherence schemes as against snooping schemes. The DASH architecture consists of a 2-D mesh network of clusters; each cluster consists of a set of processor elements (PEs) sharing a common

communication channel. The cache coherence protocol is based on a four-level memory hierarchy protocol. Lenoski *et al.* (1992) point out that even though the above topology is employed for the DASH prototype, the concepts used in DASH are topology independent. In this paper we present an improvised network architecture for implementing a directory-based NUMA distributed shared memory system. The network architecture employed in the proposed scheme is a variation of the extended hypercube (EH) architecture presented in Kumar and Patnaik (1992).

In recent years the hypercube has been one of the most popular topologies employed in building parallel computing systems as it enjoys desirable properties such as logarithmic diameter and connectivity, fault-tolerance, and good embedding capabilities. Several variations of the hypercube topology have been proposed in recent years to overcome some of its drawbacks. The EH proposed by Kumar and Patnaik (1992) is one such architecture which combines positive features of hypercube and n -ary tree topologies. The EH was proposed as a candidate architecture for building large-scale message passing systems as it has a hierarchical, modular and recursive structure, and possesses such good topological properties as constant connectivity, low diameter and low cost factor. The EH architecture consists of two types of nodes: processor elements (PEs) at level zero and network controllers (NCs) at all higher levels. In the past, the NCs of the EH have been successfully employed to perform such house keeping jobs as message passing, parallel I/O, algorithmic fault detection, task scheduling and load balancing (Kumar, 1992). Recent studies have revealed that the NCs can be successfully employed in implementing an efficient directory-based NUMA system based on a variation of the EH architecture. The novel feature of such a scheme is the use of dedicated

nodes (NCs) for performing directory maintenance and block transfer operations so that the PEs can concentrate on executing the computational tasks of the application problem. The high bandwidth network of the recently proposed Stanford FLASH multiprocessor is expected to consist of a large number of PEs (Kuskin *et al.*, 1994). Each PE node of the FLASH is augmented by a novel custom controller called MAGIC (Memory And General Interconnect Controller). The MAGIC architecture facilitates high-speed data transfers for intra-node and inter-node message passing, and contains a specialized data path that moves data in a pipelined fashion and enables the PE to update the protocol data and the associated data transfers in parallel. The FLASH prototype uses a dynamic allocation to maintain cache coherence. In effect the FLASH uses a NC to implement a directory-based NUMA system.

In this paper we describe implementation of a directory scheme in a hierarchical network to enable shared memory view of a message passing system. We also show that the size of each subdirectory is the same regardless of the size of the network. The paper is organized as follows. The second section describes a variation of the EH architecture that is used in the proposed scheme. Distributed directory control and cache coherence mechanisms are presented in Section 3. A comparison of our scheme with those of other systems is presented in Section 4. Section 5 concludes the paper with a discussion on the proposed work.

2. THE EH ARCHITECTURE

2.1. Topology of the EH and EHT

The EH architecture consists of two types of nodes, the PEs and the NCs. The basic module of the EH, represented by $EH(k,1)$ consists of a k -cube of PEs at level zero and one NC at level one. Each PE of the k -cube is connected to the NC by a dedicated link. In general, an

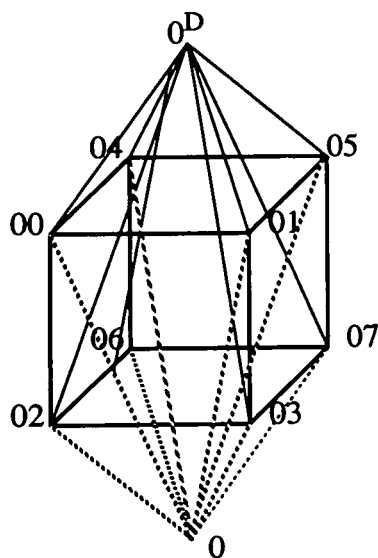


FIGURE 1. Basic module of the EHT [EHT(3,1)].

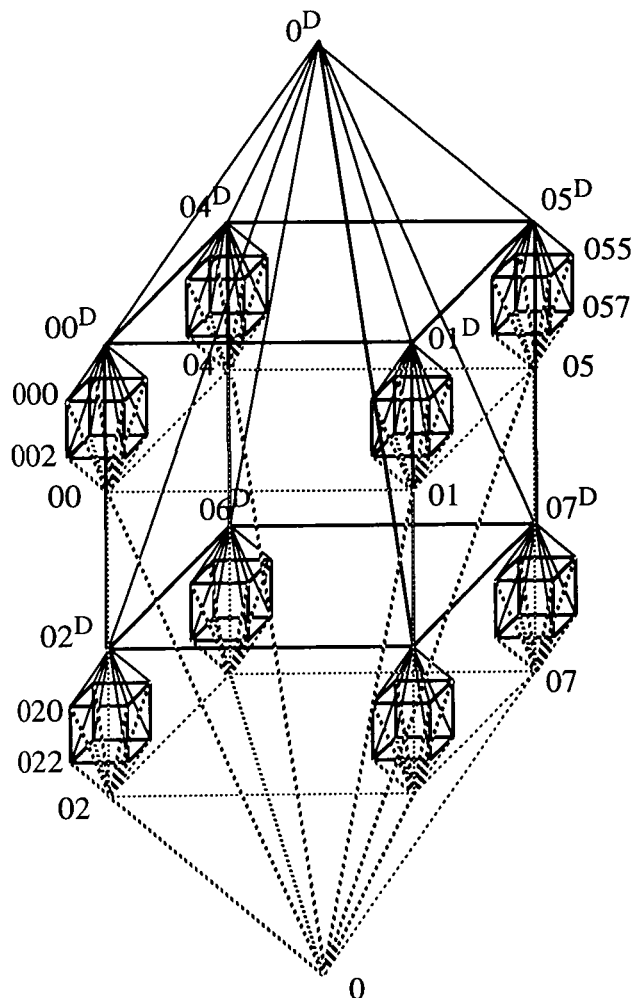


FIGURE 2. EHT(3,2).

$EH(k,l)$ is constructed by interconnecting 2^k number of $EH(k,l-1)$ s such that the NCs (of level $l-1$) of the $EH(k,l-1)$ s and the NC of level l form an $EH(k,l)$. For the implementation of the distributed directory scheme we make use of a variation of the EH architecture and represent it as EHT (Kumar, 1992). The basic module of the EHT consists of a k -cube of PEs and two NCs as shown in Figure 1. The 2^k number of PEs are at the zeroth level of hierarchy and the two NCs are at the first level of hierarchy. An EHT with $l+1$ levels of hierarchy is represented by $EHT(k,l)$. Each PE (child node) is connected to two NCs (parent nodes) by dedicated links. An $EHT(k,1)$ can be extended to an $EHT(k,2)$ by interconnecting 2^k number of $EH(k,1)$ s as shown in Figure 2. In general, an $EHT(k,l)$ can be constructed by interconnecting 2^k number of $EHT(k,l-1)$ s. The $EH(k,l)$ is a single rooted 2^k -ary tree of height l with additional horizontal links. Every node in an $EH(k,l)$ is connected to its neighbours: k siblings at level j ($0 \leq j < l$), 2^k children at level $(j-1)$, and one parent at level $(j+1)$. Topologically, the $EHT(k,l)$ differs from the $EH(k,l)$. The $EHT(k,l)$ is a 2-rooted 2^k -ary tree of height l with additional horizontal links. Every node at level j ($0 < j < l$) in an $EHT(k,l)$ is connected to its neighbours:

k siblings at level j , 2^k children at level $(j-1)$, and one parent at level $(j+1)$. Every PE (node at level 0) is connected to two parent nodes and k sibling nodes. An $\text{EHT}(k, j)$ for $j < l$, is part of an $\text{EHT}(k, l)$ and an $\text{EHT}(k, j)$ represents the rest of the $\text{EHT}(k, l)$. The $\text{EHT}(k, j)$ consists of two complementary NCs at level j and their descendants, whereas the $\text{EHT}(k, j)$ consists of all those NCs and PEs which are not in the $\text{EHT}(k, j)$. The NCs reside at hierarchical levels one and above, whereas the PEs reside at level zero. Topological and architectural properties of the $\text{EHT}(k, l)$ are similar to those of the $\text{EH}(k, l)$ discussed in Kumar and Patnaik (1992).

2.2. Addressing the nodes of the EHT

The PEs of the $\text{EHT}(k, l)$ are represented by k -digit numbers. Each digit is a modulo 2^k number. In the $\text{EHT}(k, 1)$, the NCs are addressed by 0 and 0^D and the PEs are addressed by two-digit numbers: each digit is a modulo 2^k number. The first digit indicates the address of the parent node and the second digit corresponds to the position of the PE in the k -cube; the addresses of all pairs of neighbouring PEs differ in one digit position. In general the nodes of a k -cube at level j are represented by an $(l+1-j)$ -digit address, where $0 \leq j \leq l$. Thus a node at level j is represented by $D_l D_{l-1} \dots D_j$, and a node at level 0 is represented by $D_l D_{l-1} \dots D_1 D_0$. It is to be noted that there are two NCs with the same address for all levels $j > 0$; $D_l D_{l-1} \dots D_j$ and $(D_l D_{l-1} D_{l-2} \dots D_j)^D$ or NC and NC^D .

3. DISTRIBUTED DIRECTORY SCHEME FOR THE EHT

Each PE of the proposed architecture comprises a processing unit, cache and memory. Any state of the art microprocessor could be used as the processing unit. In the EHT there are two hierarchical networks: one network is called the *request* network and the other is called the *transfer* network. An NC at level j of the *request* network is represented by $(D_l D_{l-1} D \dots D_j)^D$. Every NC of the *request* network maintains a directory of all cache block transfers related to its descendants; this has motivated us to represent all the NCs of the request network with the superscript 'D'. An NC at level j of the *transfer* network is represented by $D_l D_{l-1} \dots D_j$ and it performs all cache block transfers migrating to/from its

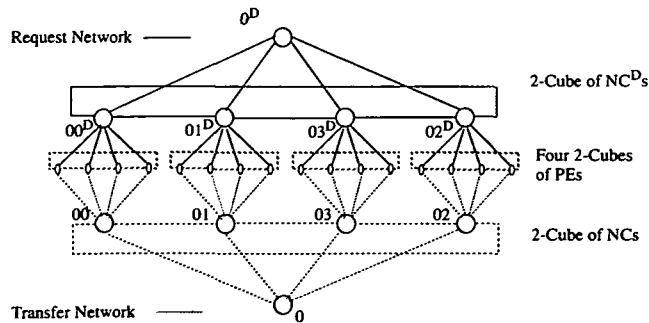


FIGURE 3. The request and transfer networks for an EH(2,2).

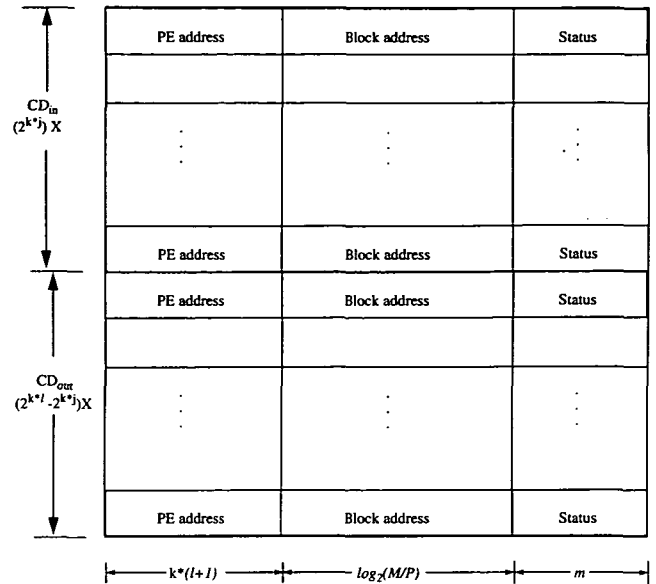


FIGURE 4. Format of cache directory at an NC of level j .

descendants. The *request* and *transfer* networks for an $\text{EHT}(2,2)$ are shown in Figure 3. The use of separate networks for transfer and request enhances the performance of the system. In Section 4 we present a qualitative analysis of the proposed network to demonstrate the positive features of the network.

3.1. Directory configuration

The entire memory area is treated as a single contiguous memory by each PE in the system. Physically, each PE of the $\text{EHT}(k, l)$ consists of a processor, memory and cache. Cache directories are maintained at all levels of hierarchy. Each PE maintains two directories: one maintains information about blocks migrating into the PE and another maintains information about blocks migrating out of the PE. These directories are called CD_{in} and CD_{out} , respectively. Each NC^D like the PEs maintains two cache directories: one for incoming and another for outgoing blocks represented by CD_{in} and CD_{out} , respectively (see Figure 4).

The CD_{out} of an NC^D at level j stores information about blocks migrating from each of its 2^{k*j} descendent PEs to one or more PEs in the $\text{EHT}(k, j)$. The PE address in this case (Figure 4) indicates the foreign PE address this block is migrating to. The maximum required size of CD_{out} is determined by the possibility that all cache blocks of $\text{EHT}(k, j)$ have migrated from $\text{EHT}(k, j)$. Hence the size of this directory is equal to the sum of all the cache sizes (number of blocks) in the $\text{EHT}(k, j)$. Each cache block in the system not only possesses a unique address, but also has a unique entry in each directory (CD_{in} or CD_{out}). As a result it takes $O(1)$ time to check the status of a block in any directory.

The CD_{in} of an NC^D at level j stores information about blocks migrating from the PEs of the $\text{EHT}(k, j)$ to one or more of the NC^D 's descendent PEs. The PE address in this case (Figure 4) indicates the local PE that

the cache block has migrated to. The maximum size of this directory is determined by the case when all cache blocks of the $EHT(k, j)$ have migrated from $EHT(k, j)$. As explained earlier it takes $O(1)$ time to check the status of a block in any CD_{in} as well.

The CD_{in} (CD_{out}) of an NC of level 1 is the concatenation of the CD_{in} s (CD_{out} s) of all its child PEs. Each NC^D at level 1 serves 2^k children. However, the NC^D of level 1 accepts one request at a time from each child PE. In general, the NC at any level cannot be pre-empted when it is servicing a request. At any given time the number of requests waiting to be serviced at any NC^D does not exceed 2^k . In other words, a PE can make one request at a time. This means that while the request can be blocked at the PE level, there is no further propagation of this blocking through the hierarchy of the EHT. This can be maintained by a simple flag called the *request flag* at the PE level. When a PE is required to make a request with its parent NC^D , it checks the *request flag*; a set flag indicates a pending request. If the flag is not set, the PE makes the request and sets the flag. The flag is reset upon receipt of an acknowledgement from the parent NC^D at level 1. The PE maintains a queue of the requests.

Division of the directory into CD_{in} and CD_{out} reduces the search area for each directory update. In the remainder of this section we analyse the properties of our directory scheme. The main memory capacity of each PE is assumed to be M bytes, which is logically divided into P blocks. The address of a given block in main memory of a PE (see Figure 4) is given by:

$$(\text{block})\text{page_address} = D_l D_{l-1} D_{l-2} D_{l-3} \dots D_0 D_p,$$

where each D_i ($0 \leq i \leq l$) corresponds to the address of the NC^D at level i , and is a k -bit mod 2^k number. D_p is the block number in the main memory and is $\log_2(M/P)$ bits long. Thus, a block address at level 0 is represented by $\{k(l+1) + \log_2(M/P)\}$ bits. If the *status* of a cache block is represented by m bits then an arbitrary directory entry at any PE in $EHT(k, l)$ is composed of

$$Q = \{k(l+1) + \log_2(M/P) + m\} \text{ bytes.}$$

The *status* keeps track of the status of the block with respect to read, write or validity. The size of the directory to address the cache of a PE is given by XQ , where X is the number of blocks in the cache. Prior to modifying a block, the active PE initiates a *block invalidation* command which is broadcast by the parent NC^D to all other PEs holding a copy of that block. Block invalidation involves changing the status of that block in all relevant directories. As opposed to the global shared memory model, individual blocks in the memory can be copied, read and written, but can never be relocated. An active PE requiring access to a block of memory, must copy the block into its cache. In section four we discuss the usage of the directory in more detail.

We use the following notations in the rest of this paper.

PE_{active}

PE that makes the request (to read, write or copy)

$NC_{active}, NC_{active}^D$
 PE_{host}
 NC_{host}, NC_{host}^D
 PE_{dirty}

parent nodes of PE_{active}
 PE that owns the requested block
 parent nodes of PE_{host}
 remote PE containing the dirty block

NC_{dirty}, NC_{dirty}^D

parent nodes of PE_{dirty}

Note: NC_x is in the *transfer* net and NC_x^D is in the *request* net.

3.2. Directory properties

In this section, we outline some properties of the directory architecture proposed. We provide proof of the fact that the distance travelled by a request message and the corresponding cache block is $O(k+l)$.

THEOREM 1. *The maximum number of hops a request message travels is $\{k + 2(l-1)\}$*

Proof. The request message travels from a PE_{active} to an NC_{host} , which is the parent node of PE_{host} . The distance between PE_{active} and PE_{host} is always greater than the distance between PE_{active} and parent of PE_{host} . The maximum distance between any pair of PEs in the $EH(k, l)$ is given by $\{k + 2(l-1)\}$, the diameter of the EH, hence the proof of the theorem. \square

COROLLARY 1. The maximum number of hops a block travels is given by $O(k + 2(l-1))$

Proof. The maximum distance a block travels is equal to the diameter of the EH. \square

COROLLARY 2. The time taken to perform a block invalidation is also given by $O(k + 2(l-1))$

Proof. Complexity of a broadcast operation is given by the diameter of the network. \square

THEOREM 2. *Given $EHT(k, l)$, for every i and j , $1 \leq i \leq l$, $1 \leq j \leq l$ the size of the cache directory at level i , is the same as the size of the cache directory at level j .*

Proof. The maximum size of a directory in an NC^D is the sum of the sizes of CD_{in} and CD_{out} . We consider the sizes of CD_{in} and CD_{out} separately, and then we will compute the sum. \square

The size of the directory is the same for all NCs. This is proved as follows.

3.2.1. Directory of incoming blocks (CD_{in})

The maximum size of the directory for an NC^D at level 1 to hold information on incoming blocks corresponds to the case when every cache block in the local cluster has migrated from elsewhere. For a cluster at level 0 containing 2^k PEs, each of which has a cache directory of size XQ , the directory size is given by

$$(2kXQ) \text{ bytes}$$

TABLE 1. Directory of sizes at different hierarchical levels

Level	CD_{in} (bytes)	CD_{out} (bytes)
1	128 K	8064 K
2	1 M	7 M
3	8 M	nil

In general, the directory size for an NC^D at an arbitrary level j in the hierarchy is given by

$$(2^{k^j} XQ) \quad (1)$$

3.2.2. Directory of outgoing blocks (CD_{out})

The maximum size of the directory at an NC^D at level 1 to hold information on outgoing blocks corresponds to the case when every cache block external to this cluster has migrated from this cluster. Thus, for an $EHT(k, l)$, the number of blocks external to the current cluster is given by

$$(2^{k^l} - 2^k)$$

The size of the directory containing addresses of $2^{k^l} - 2^k$ cache blocks is given by

$$(2^{k^l} - 2^k) XQ \text{ bytes}$$

Thus, the size of the directory at level j for the outgoing blocks is given by

$$(2^{k^l} - 2^{k^j}) XQ \text{ bytes} \quad (2)$$

The size of CD_{in} of a PE is XQ bytes and the size of CD_{out} is PQ bytes. For example if each PE has 16 Mbytes of memory, and each memory block is 128 bytes, CD_{out} will be 128 Kbytes. In Table 1 the sizes of the NC^D directories at successive hierarchical levels of an $EHT(3, 3)$ are shown when the size of each cache is 512 K bytes.

3.2.3. Total directory size

The total directory size is thus given by (1) and (2) as

$$2^{k^j} XQ + (2^{k^l} - 2^{k^j}) XQ = 2^{k^l} XQ \quad (3)$$

This result is independent of j , hence it holds for an arbitrary j , $1 \leq j \leq l$.

The size of CD_{in} for NC^D at level j is greater than the size of CD_{in} for NC^D at level $j-1$, $1 \leq j \leq l$ and size CD_{out} for NC^D at level j is greater than the size of CD_{out} for NC^D at level $j+1$. However, the total size of the directory is the same for all NCs. The size of each

directory increases with the network. For example, if an $EH(k, l)$ is extended to an $EH(k, l+1)$, the size of each directory increases 2^k -fold.

Ghose and Simhadri (1991) discuss a MIN (multistage interconnection network)-based scheme that uses switch directories to maintain cache coherence. In a MIN employing switches, each PE has $(m-1)$ sibling PEs connected by the switch. In contrast, the PEs of the $EHT(k, l)$ are directly connected to k neighbouring PEs. Further more, the EHT employs two separate networks for servicing *requests* and performing *transfers*. Andrews *et al.* (1991) propose notification and multicast networks for Synchronization and Coherence. Their scheme assumes use of networks as processor-memory interconnects and hence is differs from our scheme and those of Hagersten *et al.* (1992), Kuskin *et al.* (1994) and Lenoski *et al.* (1992), where the processor and its memory reside on the same node.

4. USING THE DIRECTORY

A PE requiring to read or write a block of memory needs to perform different steps depending on where the original block is stored and where its copies are being held.

To demonstrate the use of the directory we consider the following three cases:

1. PE_{active} wants to read a block which resides in its own memory.
2. PE_{active} wants to write to a block which resides in its own memory.
3. PE_{active} wants to copy a block from PE_{host} ; PE_{active} and PE_{host} may or may not be in the same cluster.

4.1. Case 1: PE_{active} wants to read a block b which resides in its own memory

In this case, PE_{active} and PE_{host} are the same. In our scheme, only valid blocks are retained in the caches. On a *cache hit*, it is not required to check the block's status. If the required block is not in its cache, that is on a *cache fault*, the PE_{active} probes its CD_{out} to see if the block has migrated out and if so check's the block's current status. If the block has neither migrated nor is dirty, the PE_{active} reads the block from its memory. On the other hand, if the block has migrated out to a remote PE and is dirty, then the NC^D_{active} in conjunction with the NC^D_{dirty} initiates transfer of the block. The following steps are carried out:

```

{
    if cache fault
         $PE_{active}$  probes its  $CD_{out}$  to check the status of block  $b$ 
        if  $b$  has migrated out and is dirty then {
             $NC^D_{active}$  requests  $NC^D_{dirty}$  to send a copy of block  $b$ 
             $NC_{active}$  copies  $b$  from  $NC_{dirty}$ 
             $PE_{active}$  copies  $b$  from  $NC_{active}$ 
        }
         $PE_{active}$  copies  $b$  from memory to cache
    read  $b$  from cache
}

```

4.2. Case 2: PE_{active} wants to write to a block *b* which resides in its own memory

In this case, PE_{active} copies the block into its cache after following the procedure for reading a block as outlined in Case 1. Then NC^D_{active} invalidates the status of this block in its directory and requests all remote NC^Ds to which this block has migrated to invalidate the block's status. The PE_{active} then sets a semaphore bit to lock the block prior to writing and releases the semaphore after write operation is completed. The PE_{active} then performs the write operation on this block. This process is detailed below:

```
{
  if cache fault
    PEactive probes its CDout to check the status of block b
    if b has migrated out and is dirty then {
      NCDactive requests NCDdirty to send a copy of block b
      NCactive copies b from NCdirty
      PEactive copies b from NCactive
    }
  PEactive copies b from memory to cache
  PEactive issues block invalidation command to NCDactive
  NCDactive invalidates all entries of block b in its directory
  NCDactive requests all remote NCDs holding copy of block b to invalidate its status
  PEactive locks the semaphore and performs write on b
  PEactive releases the semaphore.
}
```

4.3. Case 3: PE_{active} wants to read/write a block *b* whose address is in a remote PE_{host}

When copying from its own cluster (but another PE's memory), the following block transfer procedure is carried out:

1. PE_{active} sends a request to its parent NC^D.
2. NC^D sends a command to the PE_{host}, to initiate the block copy (the NC^D may also give the command without being prompted by the receiving PE).
3. The PE_{host} transfers the block to PE_{active} by employing the PE-to-PE links of the *k*-cube at the zeroth level. (Note: if the size of the network is restricted to only two levels, block transfer among non-neighbouring PEs is performed by using the PE-NC-PE path).

```
{
  if cache fault
    PEactive requests NCDactive to check b's status in its CDin
    if b is available on a sibling PE and not dirty then
      NCDactive directs the sibling PE to send a copy of b to PE active via PE-to-PE links
    else {
      NCDactive requests NCDhost to send a copy of b
      NCDhost checks its directory CDout
      if b has migrated out and is dirty
        NCDhost requests NCDdirty to send a copy of the updated block to PEhost
      NChost sends a copy of block b to the PEactive
    }
  PEactive copies block b to its cache
  if it is a write operation {
    NCDactive invalidates all local directory entries of this block, and requests all NCDs with an entry for b (including NCDhost) to invalidate b's status.
    PEactive locks the semaphore and performs write
    PEactive releases the semaphore.
  }
}
```

Block copy from a remote PE is performed as follows:

4. The PE_{active} sends a request to its parent NC^D at level 1.
5. The NC^D at level 1 sends a request to one of its siblings or its parent; this is repeated till the request reaches the parent NC^D of the PE_{host}.
6. The PE_{host} transfers the block to its parent NC which in turn transfers the block to one of its siblings or children; this is repeated until the block reaches the requesting PE.

When a PE_{active} wants to read or write a block whose address is in a remote PE and not currently in its cache, it probes through its NC^D_{active} to check if this block is available in the active cluster. If the block is available in the active cluster, and is not dirty, the block can be transferred to PE_{active}. If the block is not available in the active cluster at level 1, the request to find the block is transmitted to level *l* - 1 and once again the block can be transferred to PE_{active} if the block is not dirty. If the block is found at any level, but is found to be dirty, then the block must be transferred from the remote PE. If so, the remote PE, through its NC^D requests transfer of the dirty block both back to itself, and to the PE_{active} that is requesting this block. This process is described below:

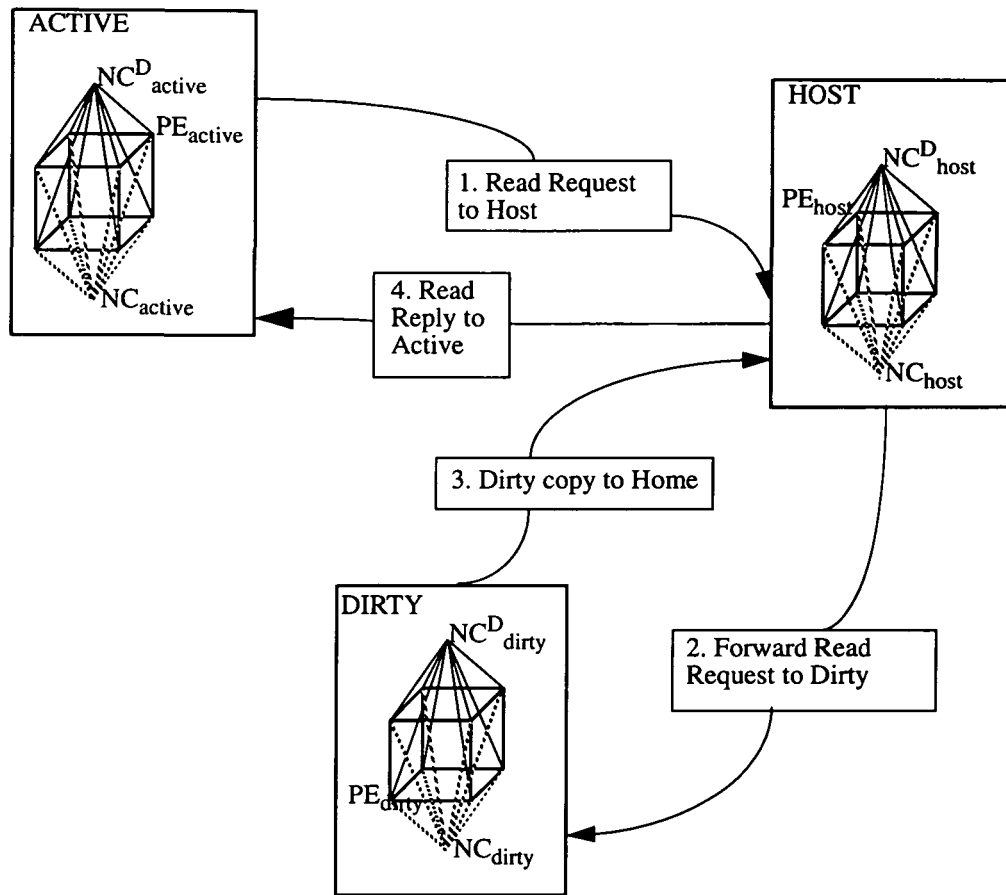


FIGURE 5. (a) Read of dirty remote cache.

The schematic diagrams shown in Figure 5 illustrate the operations carried out for reading remote dirty cache block and writing into a shared remote cache block.

The protocol mechanisms employed in the NUMA-based scheme presented in this paper is different from those used in COMA-based architectures such as the Data Diffusion Machine (DDM) proposed by Hagersten *et al.* (1992). The DDM employs a hierarchical network structure based on the COMA memory organization (Hagersten *et al.*, 1992). Due to lack of fixed home locations, additional hardware is used to keep track of data in the machine. The leaf nodes consist of PEs and attraction memories, whereas the higher level nodes consist of directories only.

However, both these architectures employ hierarchical directories. In the DDM machine, the directory size increases with the hierarchical level and the higher level memories are expensive and slow. In contrast, all NCs have directories of equal size and a typical search takes $O(1)$ time on each directory of our scheme. However, a typical read (or write) operation of our scheme requires the request message to be directed to the remote PE via the host PE, whereas in the DDM as there is no such thing as host PE, the request message is transmitted directly to the shared PE. In addition, in our scheme transfer of cache blocks (or items) involve the path via

the host PE, whereas in the DDM these transfers are from the shared PE to the requesting PE via the least common ancestor node.

5. ASSESSING THE LOAD AT EACH LEVEL

In this section we estimate the rate of service that is required at an NC^D of level j in order to service the requests coming in from the rest of the network. Consider the arrivals from the siblings and children at the NC^D at level l , the root of the network in a $EHT(k, l)$, to be represented as α_l (Figure 6). At the root, there are no siblings from which requests arrive. Let α_j be the arrival rate of requests at the NC^D of level j . We can assume that this traffic will on a statistical basis be equally divided among all its 2^k children and k siblings. That is $\alpha_l / (2^k + k)$ of the arrivals at NC^D of level l will be directed towards each NC^D (or PE) of level $l-1$. At the NC^D of level $l-1$, the total arrivals is given by $(\alpha_l / (2^k + k) + \alpha_{l-1})$. By proceeding in this manner, we can derive the arrival rate at the NC^D of level 1 as $(\alpha_1 + \alpha_2 / (2^k + k) + \dots + \alpha_l / (2^k + k)^{l-1})$. If m were to represent the rate of service of requests at each PE then

$$\mu = (\alpha_1 / (2^k + k) + \alpha_2 / (2^k + k)^2 + \dots + \alpha_l / (2^k + k)^{l-1}) \quad (4)$$

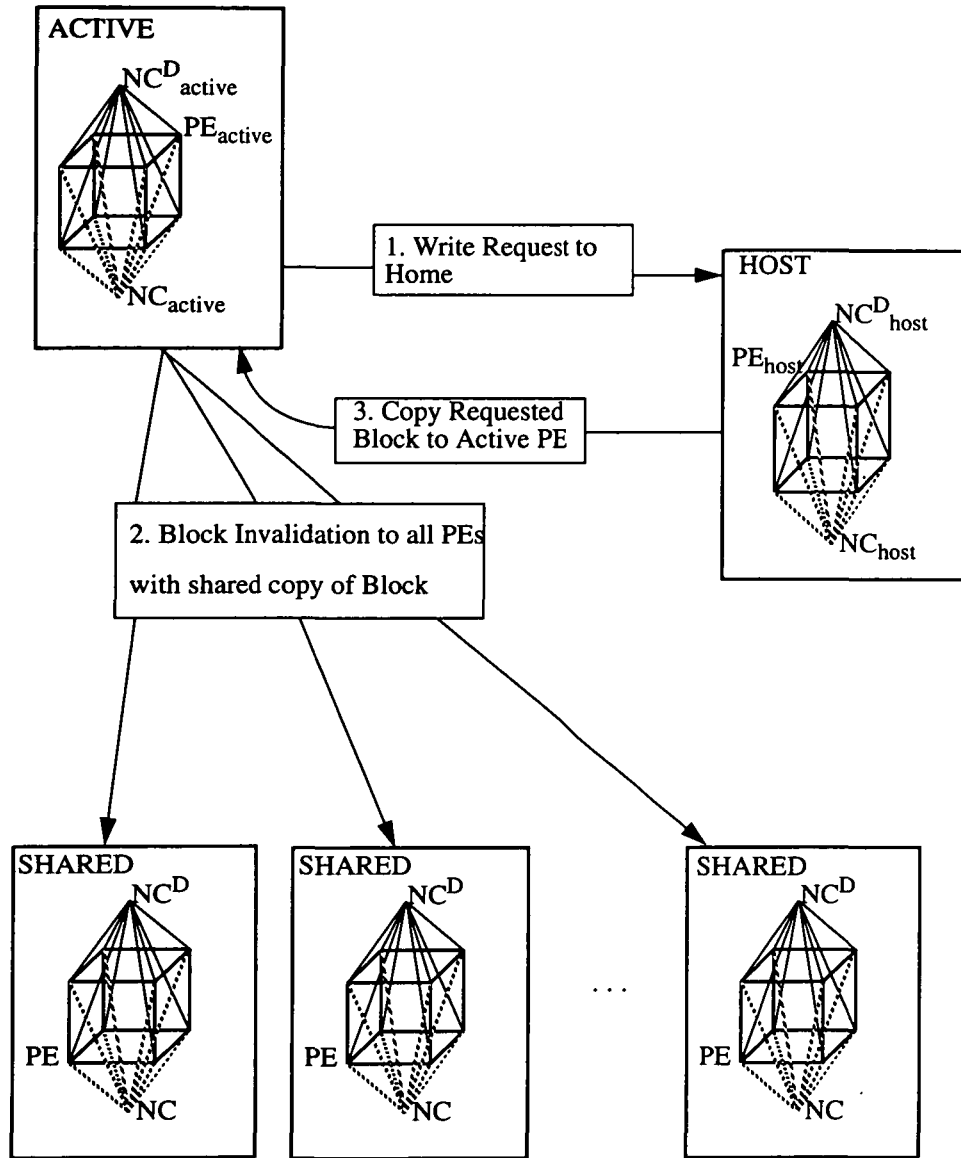


FIGURE 5. (b) Write to shared remote cache block.

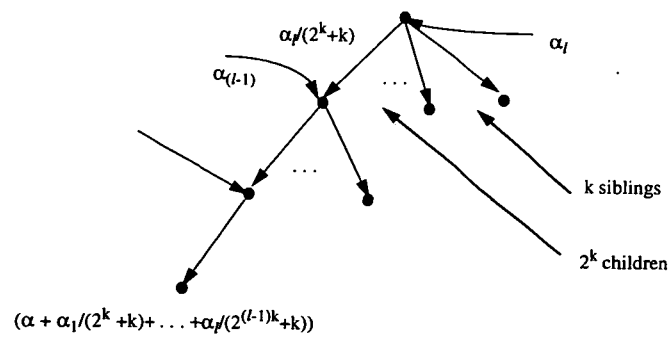


FIGURE 6. Arrival of requests at different levels.

TABLE 2

l	μ
2	$\mu = \alpha$
3	$\mu = (12/11)\alpha$
4	$\mu = (133/121)\alpha$

TABLE 3

k	μ
2	$\mu = 7/6\alpha$
3	$\mu = 12/11\alpha$
4	$\mu = 21/20\alpha$

If, $\alpha = \alpha_1 = \alpha_2 = \dots \alpha_l$, $\mu = \alpha(1 - (1/(2^k + k))^{l-1}) / (1 - (1/(2^k + k)))$. When l is large, $\mu = \alpha(1/(1 - (1/(2^k + k))))$. To consider the service rate, we examine equation (1), for $k = 3$. Then, for different values of l we have, Table 2 demonstrates the linear relationship between μ and α for varying hierarchical levels l . Similarly, for $l = 3$, Table 3 shows the relationship between μ and α for different k .

From Tables 2 and 3 it is clear that the service rate μ is dependent on the arrival rate and is independent of the size of the network. Furthermore, the service rate is approximately equal to the arrival rate.

6. COMPARISON WITH DASH DISTRIBUTED SHARED MEMORY

The DASH distributed memory prototype is constructed in four levels (Lenoski *et al.*, 1993):

1. The processor level.
2. The local cluster.
3. The home cluster level.
4. The remote cluster level.

Two aspects of the shared memory architecture in the DASH architecture and our scheme on the EH architecture are compared: memory bandwidth and the scalability of directory costs.

In the proposed scheme, as in the DASH system, there are two underlying networks: one for transferring the status and requests from point-to-point and another for transferring the blocks within the network. In the DASH architecture, this is possible at the local level by distributing physical memory among the clusters, and at the global level by having a scalable connectable network and point-to-point messages among the clusters. In the EHT architecture, the total memory bandwidth scales linearly with the number of processors as at a local level the physical memory is distributed in a hierarchy of EHs. On a global level, like the DASH architecture, the hierarchy of the EH lends itself to a scalable connectable network and the communication between remote points is also point-to-point between NCs. The topology of the network in effect determines

TABLE 4. Communications steps for typical operations such as memory read and block invalidation

Scheme	64 PEs	512 PEs	4096 PEs
DASH	9 ^a	16 ^b	28 ^c
EH-Based	5	7	9

^a 4 PEs per cluster and a 4×4 mesh of clusters.

^b 8 PEs of per cluster and a $4 \times 4 \times 4$ mesh of clusters.

^c 8 PEs per cluster and an $8 \times 8 \times 8$ mesh of clusters

TABLE 5. Total directory sizes

Scheme	512 PEs	4 K PEs
DASH	16 Gbytes	1 Tbyte
EH-Based	584 Mbytes	36.5625 Gbytes

the number of communication steps required to perform operations such as block invalidation and block transfers. It is desirable to have a small network diameter to reduce the time taken for performing these operations. As discussed in Section 2, the EH has a small diameter as compared with that of the DASH prototype whose topology is a combination of the bus and mesh topologies. In Table 4 we compare the maximum number of communication steps required to perform typical broadcast or block copy operations on the DASH prototype and the EH-based scheme proposed by us.

In the DASH architecture, the directory memory required is given by $C^2 M/L$, where C is the number of clusters, M is the number of bits per cluster and L is the cache line size in bits. The growth of the directory information is proportional to C/L . Although for small numbers of clusters, this overhead is not significant, for large number of clusters there is a significant overhead.

In the EH architecture, for a given EHT(k, l), the size of the directory at any network controller in the hierarchy is given by $2k \cdot l \cdot XQ$. Thus, if the dimension l of the hypercube is increased to say $l + 1$, the directory increases by a factor of 2^l . In Table 4 we compare the total directory sizes in the DASH prototype with those our scheme. We assume that each PE has a memory of size 16 Mbytes and cache of size 128 Kbytes. Further, the cache consists of 32 byte blocks or lines. We compare directory sizes for two cases:

1. DASH with 512 PEs, 64 clusters, eight PEs per cluster and an EH(3,3) with 512 PEs.
2. DASH with 4096 PEs, 512 clusters, eight PEs per cluster and an EH(3,4) with 4096 PEs.

Directory sizes for the above two cases are shown in Table 5.

7. CONCLUSION

In this paper we examine the design of a directory scheme for maintaining cache coherence in a distributed shared memory system based on the hierarchical network of

hypercubes. The architecture comprises two networks: one for request and one for transfers. We show that the entire shared memory can be viewed as a single address space, and that the directories required to maintain information of blocks migrating in and out of a cluster are of constant size. This distributed shared memory is compared with the DASH distributed memory with respect to memory bandwidth and scalability of directory cost. Our comparisons indicate that the proposed scheme requires a small memory space for directories and communication efficient. Further, we have completed the design aspects of a prototype system using Intel's 80 × 86 processors. The proposed scheme assumes one outstanding request per processor, whereas the DASH can support multiple requests per processor. It is planned to carry out a number of simulation studies of the proposed scheme and evaluate its performance while executing applications problems. The effect of multiple requests on the overall performance of the system needs to be evaluated. We are in the process of carrying out a simulation study of the proposed architecture.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referees whose suggestions have enhanced the technical content of this paper.

REFERENCES

- [1] Andrews, J. B., Beckmann, C. J. and Poulsen, D.K. (1991) *Notification and Multicast Networks for Synchronisation and Coherence*. Technical Report, University of Urbana Champaign.
- [2] Ghose, K. and Simhadri, S. (1991) A cache coherency mechanism with limited combining capabilities for MIN-based multiprocessors. In *Int. Conf. on Parallel Processing*, pp. I-296–I-300, Location??.
- [3] Hagersten, E., Landin, A. and Haridi, S. (1992) DDM—a cache-only memory architecture. *IEEE Computer*, **September**, 44–54.
- [4] Hwang, K., (1993) *Advanced Computer Architecture, Parallelism, Scalability, Programmability*. McGraw-Hill, New York.
- [5] Kuskin, J., Ofelt, D., Heinrich, M., *et al.* (1994) The Stanford FLASH multiprocessor. In *Int. Symp. on Computer Architecture*, pp. 000–000, Location???
- [6] Lenoski, D., Laudon, J., Gharchorloo, K., *et al.*, (1992) The Stanford Dash multiprocessor. *IEEE Computer*, **March**, 63–79.
- [7] Lenoski, D., Laudon, J., Gharchorloo, K., *et al.* (1990) The directory-based cache coherence protocol for the DASH multiprocessor. In *Proc. 17th Int. Symp. on Computer Architecture*, pp. 000–000, Location???
- [8] Lenoski, D., Laudon, J., Joe, T., *et al.* (1993) The DASH prototype: logic overhead and performance. *IEEE Trans. Parallel and Distributed Systems*, **4**, 41–60.
- [9] Kumar, M. J. (1992) *Architecture, Performance, and Applications of a Hierarchical Network of Hypercubes*. PhD Dissertation, Indian Institute of Science.
- [10] Kumar, M. J. and Patnaik, L. M. (1992) Extended hypercube: a hierarchical interconnection network of hypercubes. *IEEE Trans. Parallel and Distributed Systems*, **3**, 45–58.
- [11] Ramanathan, G. and Oren, J. (1993) Survey of commercial parallel machines. *ACM SIGARCH Computer Architecture News*, **21**, 13–33.