CACHE COHERENCE IN CENTRALIZED SHARED MEMORY AND DISTRIBUTED SHARED MEMORY ARCHITECTURES

Sujit Deshpande[#1], Priya Ravale[*2], Sulabha Apte[#3]

[#] *Lecturer, CSE Department, Walchand Institute of Technology, Solapur, Solapur University, Solpaur, Maharashtra, India.*
[*] *Lecturer, IT Department, Walchand Institute of Technology, Solapur, Solapur University, Solpaur, Maharashtra, India.*
[#] *Professor, CSE Department, Walchand Institute of Technology, Solapur, Solapur University, Solpaur, Maharashtra, India.*

[1]sujit.sujitdeshpande@gmail.com, [*2]priyaravale@rediffmail.com, [3]aptesulabha@gmail.com

Abstract— **In the field of distributed and parallel computing the common term is multiprocessor architectures. The multiprocessor architectures may have centralized shared memory or distributed shared memory. These systems may not be useful when the number of processors is large as the bandwidth on the memory becomes excessive and this produces a bottleneck. Significant reduction with the problem of memory bandwidth can be resolved by inclusion of large caches with processors. But inclusion of caches with processors creates the problem of cache coherence. This paper presents cache coherence problem and solution on it, in multiprocessor architectures with various protocols of cache coherence. It compares and discuses benefits and limitations of protocols.**

Keywords—**Cache coherence, Distributed Shared Memory, Write Invalidate, Write Update**

I. INTRODUCTION

Distributed computing refers to the use of distributed systems to solve computational problems. In distributed computing, a problem is divided into many tasks, each of which is solved by one computer. Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel").

As power consumption (and consequently heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multicore processors.[5]

II. LITERATURE SURVEY

The terms "concurrent computing", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them. The same system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel. Parallel computing may be seen as a particular tightly-coupled form of distributed computing, and distributed computing may be seen as a loosely-coupled form of parallel computing. Nevertheless, it is possible to roughly classify concurrent systems as "parallel" or "distributed" using the following criteria:

- In parallel computing, all processors have access to a shared memory. Shared memory can be used to exchange information between processors.
- In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.

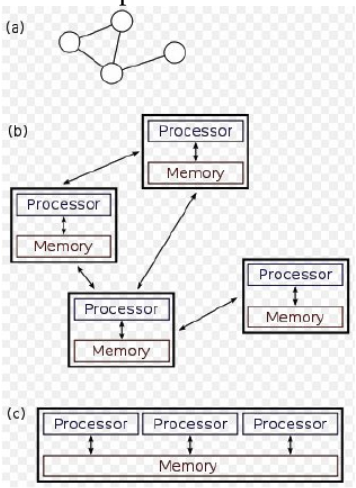

Fig. 1 – Distributed and Parallel Systems

The Fig. 1 illustrates the difference between distributed and parallel systems. Fig. 1 - (a) is a schematic view of a typical distributed system; as usual, the system is represented as a graph in which each node (vertex) is a computer and each edge (line between two nodes) is a communication link. Fig.1 - (b) shows the same distributed system in more detail: each computer has its own local memory, and information can be exchanged only by passing messages from one node to another by using the available communication links. Fig.1 - (c) shows a parallel system in which each processor has a direct access to a Shared memory. With the above discussion we can say distributed computing uses distributed memory while parallel computing uses shared memory concepts. Both of them has some advantages and disadvantages listed below

Distributed Memory

- Scales well
- Difficult to program
  - Message passing or RPC based

Shared Memory

- Easy to program
- Difficult to build
  - Tight coupling in hardware

Advantages of both distributed memory and shared memory can be merged and limitations of both can be overcome in Distributed shared memory with advantages listed below.

- Logically shared memory
- Physically distributed local memories
- Page based
  - Shared pages
  - Demand paging between nodes.

III. CACHE COHERENCE

In this paper we have discussed cache coherence in centralized shared memory and distributed shared memory architectures.There are two types of MIMD machine depending upon whether memory is locally assigned to each processor, or all processors communicate with the same central memory. This second option is called centralized shared-memory architectures, and is shown in Fig. 2.[1]

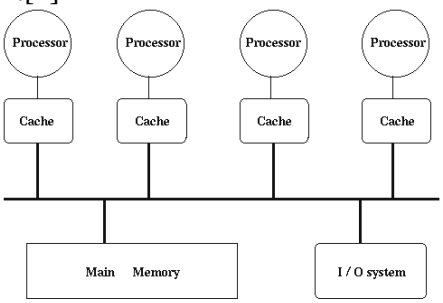

Fig. 2 – Centralized shared memory architecture

This kind of architecture is very useful for multiprocessor workstations such as Sun Ultra-SPARC workstations. It is not very useful when the number of processors becomes large as the bandwidth on the memory becomes excessive and this produces a large bottleneck. For large arrays of multiprocessors a distributed shared memory approach is more suitable as shown in FIG. 3: [6]

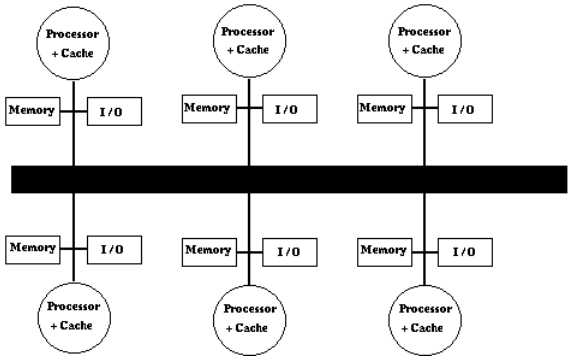

Fig. 3 – Distributed shared memory architecture

Significant reduction with the problem of memory bandwidth can be resolved by inclusion of large caches with processors. The architecture supports the caching of both shared and private data. Private data is used by a single processor, while shared data is used by multiple processors; essentially providing communication among the processors through reads and writes of shared data. When private data is cached, its location is migrated to the cache, reducing the average access time as

well as the memory bandwidth required. Since no other processor uses the data, the program behavior is identical to that in a uniprocessor. When shared data are cached, the shared value may be replicated in multiple caches. In addition to the reduction in access latency and required memory bandwidth, this replication also provides a reduction in contention that may exist for shared data items that are being read by multiple processors simultaneously. Caching of shared data, however, introduces a cache coherence problem.

In computing, cache coherence (also cache coherency) refers to the consistency of data stored in local caches of a shared resource. When clients in a system maintain caches of a common memory resource, problems may arise with inconsistent data. This is particularly true of CPUs in a multiprocessing system. Referring to the Fig. 4, if the top client has a copy of a memory block from a previous read and the bottom client changes that memory block, the top client could be left with an invalid cache of memory without any notification of the change.

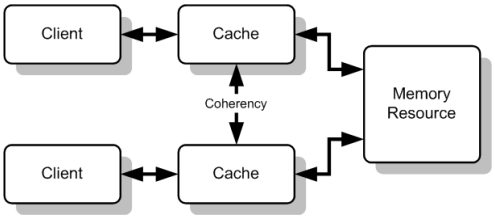

Fig. 4 – Multiple Caches of Shared Resource

TABLE I illustrates cache – coherence problem.

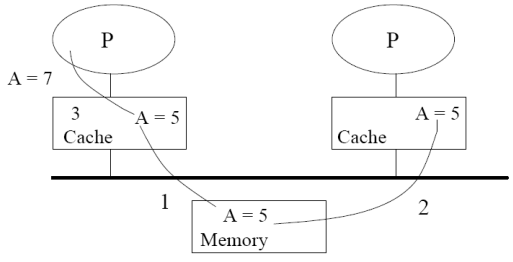

Fig. 5 – The cache – coherence problem for a single memory location, read and written by two processors.

TABLE I

The cache – coherence problem for a single memory location (X), read and written by two processors (A and B).

Time	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				1
1	CPU A reads X	1		1
2	CPU B reads X	1	1	1
3	CPU A stores 0 into X	0	1	0

Cache coherence is intended to manage such conflicts and maintain consistency between cache and memory.

Two policies are used to overcome the cache coherence problem

1. Write Back or Write Invalidate
2. Write through or Write Broadcast or Write Update

Write Invalidate vs. Write Update Strategies

1. *Write Invalidate* : On a write, all other caches with a copy are invalidated
2. *Write Update* : On a write, all other caches with a copy are updated

• *Write Invalidate is bad when :*

– single producer and many consumers of data.

• *Write Update is bad when :*

– multiple writes by one PE before data is read by another PE.

– Junk data accumulates in large caches (e.g. process migration).

– Efficiency of Interconnection network (ICN) decreases.

• Overall, Write invalidate scheme is more popular as the default.

The protocols to maintain coherence for multiple processors are called cache – coherence protocols. They are –

1. *Snooping Protocol* – for centralized shared memory architectures
2. *Directory Based Protocol* – for distributed shared memory architectures

1. *Snooping protocol* – Every cache that has a copy of the data from a block of physical memory also has a copy of the sharing status of the block, and no centralized state is kept. The caches are usually on a shared – memory bus, and all

cache controllers monitor or snoop on the bus to determine whether or not they have a copy of a block that is requested on the bus. This protocol is useful for small systems as shown in Fig. 2. [6]

TABLE II

Requests from processor & the bus and responds to these based on their type.

Request	Source	State of addressed cache block	Function
Read hit	Processor	Shard or exclusive	Read data in cache
Read miss	Processor	Invalid	Place read miss on bus
Read miss	Processor	Shared	Address conflict miss; place read miss on bus.
Read miss	Processor	Exclusive	Address conflict miss; write back block, then place read miss on bus
Write hit	Processor	Exclusive	Write data in cache
Write hit	Processor	Shared	Place write miss on bus
Write miss	Processor	Invalid	Place write miss on bus
Write miss	Processor	Shared	Address conflict miss; place write miss on bus
Write miss	Processor	Exclusive	Address conflict miss; write back block, then place write miss on bus
Read miss	Bus	Shared	No action, allow memory to service read miss
Read miss	Bus	Exclusive	Attempt to share data; place cache block on bus and change state to shared.
Write miss	Bus	Shared	Attempt to write shared block; invalidate the block.
Write miss	Bus	Exclusive	Attempt to write block that is exclusive elsewhere; write back the cache block and make its state invalid.

As write invalidate is preferable than write update snooping protocol is implemented using write invalidate policy.

In this protocol cache controllers (snoopers) continuously snoop (monitor) the bus, watching the addresses. It checks whether the address on the bus is in their cache and if so, it takes respective actions depending on the request either by processor or bus. The cache coherence mechanism receives requests from both the processors and the bus and responds to these, based on the type of request, whether it hits or misses in the cache, and the state of the cache block specified in the request. Fig.6 shows the finite state transition diagram for a single cache block with all above listed requests and functions for respective request.

2. Directory based protocol – The sharing status of a block of physical memory is kept in just one location, called the directory.[3]

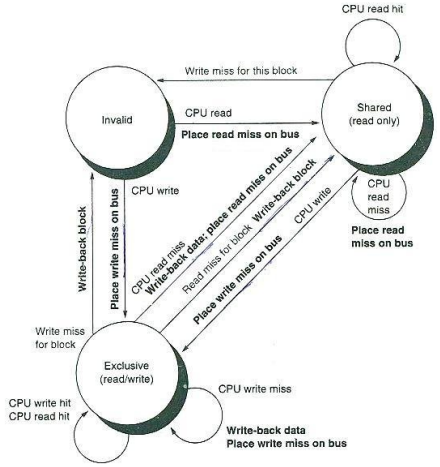

Fig. 6 – Cache – coherence state transition diagram with the state transitions induced by the local processor and the bus activities.

This protocol considers the system as shown below

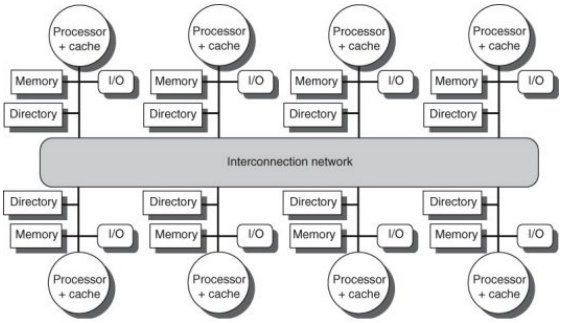

Fig. 7 – Distributed shared memory multiprocessor architecture with directories.

A directory is added to each node to implement cache coherence in a distributed memory multiprocessors. Each directory is responsible for tracking the caches that share the memory addresses of the portion of memory in the node. The directory may communicate with the processor and memory over ICN, as shown. Like snooping protocol, there are two primary operations that a directory protocol must implement : handling a read miss and handling a write to a shared, clean cache block. To implement these operations directory must track the state of each cache block. In a simple protocol, these states could be the following :

Shared – One or more processors have the block cached, and the value in memory is up to date.

Uncached – No processor has a copy of the cache block.

Exclusive – Exactly one processor has a copy of the cache block, and it has written the block, so the memory copy is out of date. The processor is called the owner of the block.

With these states of cache block, we must track the processors that have copies of the block when it is shared, since they will need to be invalidated on write. The simplest way to do this is to keep bit vector for each memory block as shown below. When the block is shared, each bit of the vector indicates whether the corresponding processor has a copy of that block as shown in Figure. We can also use the bit vector to keep track of the owner of the block when the block is in the exclusive state as shown in Figure.

C – Block is cached

S – Block is Shared

E – Block is Exclusive

TABLE III

Example of bit vector

Processors		P1		P2		Pn	
Blocks	Status in Memory	C/U	S/E	C/U	S/E	C/U	S/E
B1	Valid	C	S	C	S	U	
B2	Valid	C	S	U		U	
Bn	Valid	C	Invalid	C	E	U	

In this protocol, the communication between processors and directories can happen by sending the messages. Different messages are listed below which are sent among nodes. The nodes categorized as

Local node : It is the node where requests originates.

Home node : It is the node where the memory location and directory entry of an address reside.

Remote node : Copies exist at third node, called remote node.[4]

A remote node is the node that has a copy of a cache block, whether exclusive or shared. A remote node may be the same as either the local node or the home node. In such cases, the basis protocol does not change, but interprocessor messages may be replaced with intraprocessor messages.

The possible messages sent among nodes to maintain coherence, along with the source and destination node, the contents (where P = requesting processor number, A = requested address, and D = data contents), and the function of the message, listed in TABLE 4.State transition diagram for an individual cache block in a directory – based system is shown in Fig.8. In the state diagram requests by the local processor and from the home directory are shown. The states are identical to those in snooping protocol, and the transactions are very similar, with explicit invalidate and write – back requests replacing the write misses that were formerly broadcast on the bus. [3][4][6]

TABLE IV

Possible messages sent among nodes to maintain coherence

Message Type	Source	Destination	Function of this message
Read miss	Local Cache	Home directory	Processor P has a read miss at address A; request data and make P a read sharer.
Write miss	Local Cache	Home directory	Processor P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	Home directory	Remote Cache	Invalidate a shared copy of data at address A.
Fetch	Home directory	Remote Cache	Fetch the block at address A and send it to its home directory ; change the state of A in the remote cache to shared
Fetch / Invalidate	Home directory	Remote Cache	Fetch the block at address A and send it to its home directory ; invalidate the block in the cache.
Data Value Reply	Home directory	Local Cache	Return a data value from the home memory.
Data Write Back	Remote Cache	Home Directory	Write back a data value for address A.

In this protocol the directory implements the other half of the coherence protocol. A message sent to a directory causes two different types of actions: updates of the directory state and sending additional messages to satisfy the request. The states in the directory represent the three standard states for a block as uncached, shared , exclusive. The Figure shows the actions taken at the directory in response to message received. The directory receives three different requests: read miss, write miss, and data write back.

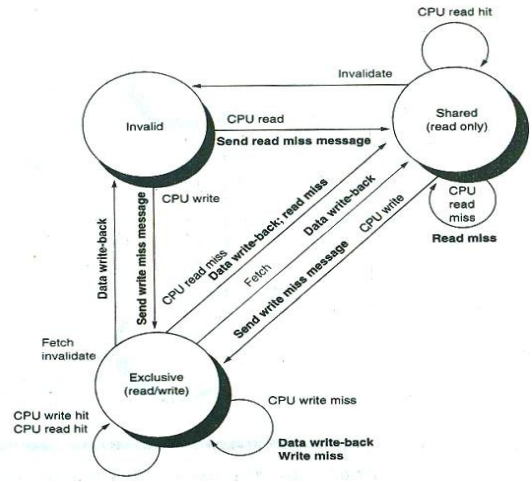

Fig.8 – Cache – coherence state transition diagram

The state transition diagram for the directory has the same states and structure as the transition diagram for an individual cache.

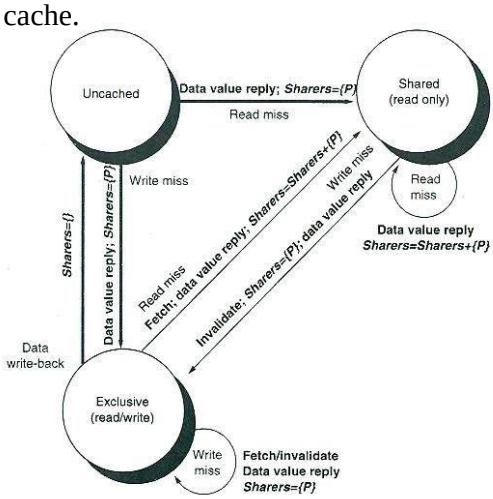

Fig.9 – Cache – coherence state transition diagram for directory

COMPARISON BETWEEN SNOOPING AND DIRECTORY BASED PROTOCOL

Snooping protocols tend to be faster, if enough bus bandwidth is available, since all transactions are a request/response seen by all processors. The drawback is that snooping isn't scalable. Every request must be broadcast to all nodes in a system, meaning that as the system gets larger, the size of the (logical or physical) bus and the bandwidth it provides must grow. Directories, on the other hand, tend to have longer latencies, but use much less bandwidth since messages are point to point and not broadcast. For this reason, many of the larger systems use this type of cache coherence.

IV. CONCLUSION

In this paper we have discussed about centralized shared memory and distributed shared memory architectures with cache coherence protocols. Cache coherence is an important part of multiprocessor systems without which the performance degrades. In general, the directory based protocol is more used for larger systems to enhance their performance; while snooping protocol is used for smaller systems.

FUTURE SCOPE

The concept of cooperative caching can be introduced instead of cache allotted to each processor. So that caches can be used more efficiently.[2]

REFERENCES

[1] Dubois, M. ; Briggs, F.A. ; - Effects of Cache Coherency in Multiprocessors Computers, IEEE Transactions on Volume: C-31 , Issue: 11 Digital Object Identifier: 10.1109/TC.1982.1675925 Publication Year: 1982 , Page(s): 1083 - 1099

[2] Jing Zhao, Student Member, IEEE, Ping Zhang, Guohong Cao, Senior Member, IEEE, and Chita R. Das, Fellow, IEEE - Cooperative Caching in Wireless P2P Networks: Design, Implementation, and Evaluation

[3] Lenoski, D. ; Laudon, J. ; Gharachorloo, K. ; Gupta, A. ; Hennessy, J. ; - The directory-based cache coherence protocol for the DASH multiprocessor; Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on Digital Object Identifier: 10.1109/ISCA.1990.134520 Publication Year: 1990 , Page(s): 148 - 159

[4] Agarwal, A. ; Simoni, R. ; Hennessy, J. ; Horowitz, M. ;- An evaluation of directory schemes for cache Coherence; Computer Architecture, 1988. Conference Proceedings. 15th Annual International Symposium on Digital Object Identifier: 10.1109/ISCA.1988.5238 Publication Year: 1988 , Page(s): 280 - 289

[5] Sinha, P. ;- Fundamentals of Distributed computing; Distributed Operating Systems:Concepts and Design Digital Object Identifier: 10.1109/9780470544419.ch1 Publication Year: 1996 , Page(s): 1 - 45 Copyright Year: 1997

[6] David A. Patterson, John L. Hennessy, – Computer Architecture – a quantitative approach ; 3rd edition published by Morgan Kaufman Publishers Publication date : January 2003.