Parallel Computer Architecture

Prof. Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati

Week - 07
Lecture - 41

Lec 41: Basic Operation of a Directory

Hello everyone. We are doing module 5 on scalable shared memory systems. This is lecture number 2 where we are going to discuss about the basic operation of a directory. So, before we go into understanding how a directory works, some definitions or terms we need to understand. We will discuss them one by one, but a quick overview. So, we have a concept of a home node.

Home node as the word says, the data block is in this particular, particular node of the system. Now what is my system? My system consists of several nodes connected to a scalable network. Every node has a processor, a cache and a memory. Right. This is the picture you have to keep in mind.

When I say home node, a processor when it accesses a block and if that block is available in the memory slice with its node, it is called a home node. Home node is also the node which is going to initiate a request. Okay. Then the concept of a dirty node. Dirty node is that node which has the block in the dirty state. Dirty state meaning it will be the provider of the node.

It will have to write back this data to the particular memory location as and when it evicts the data block. Then we have a concept of a owner node. Owner node, word says ownership. So, it has the data, it has to provide the data when asked for. Exclusive node is in addition to the E state which we were discussing, exclusive says that this is the only copy of the node in the complete system. Right.

This block is only held by, exclusively held by a particular node in the system. Local node is this node where the data is available and remote node is all the data blocks which come from other nodes and not this particular node. So, more details, we will see one by one. Home node was easy whose main memory has the block. So, you will have that slice of memory, in that this particular memory block is allocated.

Dirty node has a copy of the data block in a modified state. Owner node, it currently holds a valid copy of the data item and it is owner, so it has to supply the data whenever asked for. So, this owner node could be the home node. When will it be the home node? When the data is clean because the owner is the home because that clean data block is there, sitting in the memory of that particular node. And the owner node will be the dirty node when this particular block is in a cache which has changed the data item.

So, owner node can be of two types. It can be the home node for clean blocks and it can be the dirty node for dirty blocks. Exclusive node is the node which has a copy of the block in an exclusive state. So, here exclusive has got an additional meaning that it is either a clean or dirty. Here in the protocol when we had the E state, it meant that we had the clean data block and memory was up to date.

But if you have a node in a, housing a dirty block even that we can say that exclusive because it has an exclusive copy. So, that is the single copy of the block in the system. So, that is the concept of exclusive node. Then we have a local node. Local node is that node containing the processor which issues the request.

So, if P1 is issuing the request the node which houses this P1 is called the local node. Local blocks, block whose home node is local. Right. So, all the blocks which are accessed by the processor, if all those addresses are in your own memory slice then all these blocks are called local blocks, locally allocated blocks. Right. They are sitting in that particular memory bank. Apart from all this, all the other addresses which are not in your memory bank, they are elsewhere are called remotely allocated blocks or remote blocks. Okay.

So, we will take an example to further imbibe this concept. The different types of nodes. That is my system processor, cache, network interface. So, here I am going to see if each processor reads suppose data item A. Now this A is sitting here. So, this is the first node and that is the local address of A is node 1.

If I just say this is N1, this can be N2 and N3. If I say 3 nodes are there. A is a local node of N1. If N P2 reads A, it has to read A from N1. It does not have A in its memory block. So, for P2, A is a remote node.

For P3, A is again a remote node. Next suppose P1 want to read B. Now where is B? B is with N2. So, for P1, B is a remote node. For P2, for this, B is a local node and for P3, B is a remote node.

Now D, D is with P3. So, this D for P2 is a remote node. Okay. So, with this I hope you understand the concept of a local node and a remote node. Okay. We will continue the example. Here when P1 reads A, A it is a home node for A, it is a locally allocated block and it is a local node.

So, with respect to block A, N1, load N1 is the local node. When P2 reads A, A is remotely allocated. Right. So, it has to go to N1 to read through the network. So, it is remotely allocated. Hence, it is a remote node for P2.

When P2 reads D, D is also a remote node because D is sitting in the third node. And now here, D is a dirty node because this particular node has modified the data item. Hence, if I ask for block D which is the dirty node. So, when I say, for block D, the dirty node is N3. Here both the local node and the dirty node were same, but it may be possible.

Suppose I take another data item say read E, sorry F say. Suppose this does a read F, F is allocated here in N2. Yeah, we can have F here and let N1 change F. Right, this is possible. So, N3 wants to read F, F's local node is P2, that is N2 and F is modified by N1.

So, if I ask you about F, so you can say, F is a local node for N2, it is a remote node for N1 and N3 and F is dirty with N1. So, we can say that F is a dirty node with N1. So, this way think, the terminology will be clear to you. Okay, and then the owner, last point. Owner of block D, who is the owner of block D? It is N3 because it has the block in dirty node.

If I ask you who is the owner of block F? Block F, although it is local to N2, but because N1 has modified it, I can say owner of block F will be N1. Right, so with those terminologies which we will continue to use, we will quickly see a basic operation of a directory. So, how does a directory work? As we said, the directory is going to maintain information of all the nodes. How does it maintain that information? Easiest way is to maintain a bit vector. For K processors, I can have a K bit vector.

Every bit in this vector will say whether that particular processor has cached this data item or not. Apart from that, I need to know the dirty bit, that is has any of these processors modified the data item? With every cache, you will have the state of the cache block. Again, the coherence FSM is also associated with it. Then you have the dirty bit and any state bits associated with the cache.

So, cache had the valid dirty and the state bit. Directory only has the presence bit vector, that is the K bits vector and one dirty bit. So, that is the information. Okay. So, now every node will have to orchestrate with the directory. So, it goes through its cache controller and coherence controllers to the directory to get the work done.

So, when the directory receives that information from the local processor, if it is a locally allocated node or it will get a request from a remote node with the remote processor is sending a request to this directory. So, now what type of request we have to handle? A read request, a write request and a write back or a block replacement idea. Okay. So, we will do these two one by one. All right. So, we have this network where we have the directory.

Each directory has got a presence bit vector with these dots saying that these are, these nodes have cached the data item and who has cached and who has not cached the data item. So, we have a K bit vector denoting the presence of the block across the nodes. Read by a processor I, we are going to see next. And on block replacement, we will quickly handle this first. On block replacement, the request comes to the directory controller.

It updates the local memory, right. The memory will be updated and the dirty bit will be set off. So, this is the dirty bit with the directory, you will just turn that bit off and also now remove the, this bit from that particular node. Okay. Let us do the read operation. If the dirty bit is off, there is a dirty bit, this bit is off and this processor sends a request to the directory for reading a block.

The block is not dirty. So, processor says, the directory identifies who all are the sharers. It simply puts a dot in the index for this particular processor. So, P[i] will become 1 that is this cell will be turned into 1 from a 0 and the data, it sends the data. Right. So, sorry, data will go from the memory.

So, data is sent. So, read the memory and then turn the P[i] to 1, that is when the dirty bit is off. If the dirty bit is on and you want to do a read, so what will you do? Dirty bit on means the data is elsewhere, it is not in the memory. So, you need to recall the block from the dirty node. Now you understand the concept of a dirty node. Directory goes to the dirty node, brings the data.

For that dirty node, it changes the state to share because that dirty node had the data in modified state, it now makes that state to share. So, this particular dirty node will go from M to S, then the dirty bit will be turned off. We will provide the data to the processor and turn P[i] equal to 1. Suppose P[j] was the block with the dirty node, it still has its bit 1. We are not going to remove this because it is a read request.

On the write request, if the dirty bit is off, that is there is no dirty block in the system, but there are sharers. So, I have to send an invalidation to all the j's where P[j] is equal to 1. So, all these dots which you can see, these are the presence bit vector which says that all these 3, 4 processors have the data blocks. So, I have to send invalidation to them. The directory has to wait for an acknowledgement of invalidation. Subsequent to this, because it is a write operation, we need to make P[j], 0 for all these j who are earlier caching the block.

Then we have to make P[i] equal to 1 because now P[i] is going to modify the data block and turn the dirty bit on because P[i] will change the data item. So, when a write operation happens, if the dirty bit is off, you simply send invalidation to all the nodes which were sharing this data block. Once you get an acknowledgement saying that those nodes have deleted the block, you turn their bit to 0 and then give the block to P[i] making P[i] equal to 1 and turning the data dirty bit on. Second case is when dirty bit is on. So, when dirty bit is on, the data block is not with the memory, but there are no other sharers as well.

There are not multiple sharers but there is just exactly one sharer. So, I need to go to the dirty block, that is the processor having the block in the dirty state, go to that block and this block was in M state, change it to i, that is make invalid that particular data block, bring the data, give the data to the requester and then keep the dirty bit on because now you are still going to modify the data item but the P[j] becomes 0 but P[i] becomes 1. Right. So, if P[j] was the original requester, suppose this was the jth processor, then you will make P[j] 0 and you will make P[i] equal to 1. Because now P[i] will start changing the data item. Dirty bit remains on. Okay.

So, this is how read and write will be handled. This is an overall picture, there are many intricacies which we will discuss as we study the topic further. So, we have seen an overall operation of the directory. Now why are these directories important? They are good for my read miss because when a read miss comes, I can provide the data from the memory and the directory simply keeps track of that but on the write miss normally we had to send invalidations. If I was using a snow ping, I would end up sending a broadcast invalidation to all those nodes. Suppose 5000 nodes will receive invalidations whereas in the directory you have a bit vector and you know which bits are on, only those processors will receive an invalidation, others will not receive.

Hence, your bandwidth advantage comes there. Again the number of nodes which share the data item are again not too many. They do not scale with P. Suppose you have 500 processors, if you make them to 5000, it is not that your sharers will become 600 or 1000. Sharers will remain, say if there were 80 to 100, they will still remain 80 to 100. Number of sharers are limited, hence the global broadcast is unnecessary. Okay.

So, directories are valuable on a read because I can simply satisfy it by the memory.

And on a write we do not need to broadcast, because the number of sharers are not too many. So, number of sharers do not scale quickly with the number of processor nodes, hence on a write miss, we save on the broadcast disadvantages. We have seen that empirically number of sharers is small, it does not go up very quickly and hence I do not need to send invalidations to many processors. So, broadcast of invalidation is easily nullified or not required in this case. Okay. So, with these observations we can say that directories can become a scalable approach and also help in organizing the directory in a cost effective manner.

So, although directories are helping us, we need to still learn how do I organize them in a best possible way, so that as my system scales, does my directory scale and how do I manage the protocol more efficiently, even after it is already efficient, how to make it more efficient. So, we are going to discuss all this and look at details of how a directory works in future lectures. So, with this we with an overview of how a directory functions, we will stop this lecture. Thank you so much.