Parallel Computer Architecture

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

Week - 09
Lecture - 47

Lec 47: SGI Origin Architecture

Hello everyone. We are starting module 6 today. The theme of the module 6 is we are going to discuss directory coherence case studies. And in this module we are going to do two case studies. One is on the SGI origin architecture and the second one is in the sequent NUMA-Q architecture. The first one is a flat memory based directory protocol and the second is a flat cache based directory protocol.

In each of these designs we are going to look at the architecture, how the protocol works and the correctness aspects. So, we will start with the first lecture which covers the SGI origin architecture. Okay. So, this is a directory protocol therefore, there is no snooping among the processors. So, even if there is a bus available the processors are not going to snoop on the bus because there is going to be no transmission of snoop messages onto the bus.

And there is going to be a separate directory controller which is going to handle actual one to one messages from the controller to the cache coherence controller. So, there is going to be a coherence controller or with every node there is going to be a coherence controller or the cache controller which is going to handle these messages coming from the directory controller. And the SGI origin is a slightly older machine, but it gives you a flavor of how a directory protocol will be implemented in a real system. More advanced versions of this are implemented in today's processors. So, we will start with the overview of the architecture or how the system looks like and then in the next lecture we will look at how the protocol works.

So, this SGI architecture is made up of one board on which we have two processors on it. So, every processor is the R10000 MIPS architecture. These processors are connected to each other through a system bus, but they do not implement a snooping protocol. Overall these are connected to the outside world through a module called the hub. Okay. So, we look at the architecture now.

So, this is one board I would say. So, one PCB board is this one rectangle and you can have several such connected to the interconnection network. Every board consists of two processors. Every processor is a R10000 MIPS architecture processor. These are connected to each other through a system address and data bus.

This bus is only used for communication to other modules, but not inter processor communication. Both of them are connected to the module called hub which is the main module implementing the directory protocol. Every PCB board or every node of a 2000 processor consists of a slice of the global memory. It has from 1 to 4 GB of global memory and a directory associated with it. Now you can recollect that we when we discussed directory protocols, we said that the directory information is stored with the home node and the home node's information is kept with the main memory. Okay.

So, these are some more details of the architecture. Every processor has got a 4 MB cache with 128 byte blocks and 2-way set associative. The processors are 195 MHz processors and they give a peak performance of either 390 MFLOPS or 780 MIPS. Overall, if I am going to connect such several boards to an interconnection network, we are going to get a improved system level performance and the max number of processors which I can connect to Origin 2000 system is 1024. So, if I have 1024 processors, I am going to have 512 such boards connected to the network.

You can imagine the scale of the system, this is very big and if your application is going to run on several of these processors or several of these nodes sharing data across each other, you can imagine the scale of coherence management which we have to handle. The max system performance is 500 GFLOPS. The SysAD bus that is the system address and data bus is a 780 MB per second link. It is mainly used for inter-board communication and not inter-processor communication. The overall memory bandwidth is this and the hub, which is the main directory coherence controller module, it connects to the outside router at a 1.56 Gbps link.

Okay. So, that was a quick overview. We are going to see more details of how we expand the small diagram to a bigger network, but before that what happens to the directory? I mean is the directory going to consider every processor within this board as a single processor, that is, is this board, is this one board, one node or is it consisting of two processors and are they independent to each other? Okay. So, this every processing node has got two MIPS processors, but they would be treated independently by the coherence controller. Overall, we have seen that we have a main memory module, an IO interface, a communication assist which is the hub. So, hub is called the coherence controller or the communication assist. This hub has to do all the job for coherence and hence it should be capable of accessing the memory system as well as the caches.

So, it has to be tightly integrated with the memory system of every processing node. It is going to see all the cache misses because it handles coherence whenever there is a cache miss it has to go to the hub. So, hub has to be capable of handling all the cache misses. It should also be able to receive transactions coming from the outside network and also capable of retrieving data from the caches. Okay. So, this hub which you can see here, it should be able to see all the cache misses coming out from the processor node.

It should definitely be able to read and write to this memory. It should be able to go and retrieve data from these caches and also send and receive messages to the interconnect. Okay. So, this is a zoomed out picture of that same processing node. Now let us look into more details of this particular one node. So, this is the hub and the hub is now connected to the main memory here and it is also connected to several other I/O or peripherals.

So, in the peripherals we have the network and the I/O ports and this hub is called the heart of the chip because it is going to do all the job for us related to the coherence. It connects this one node with other nodes in the system through the Xbow switch. So, we are going to look at this switch. So, one hub goes outside connects to the XbOw switch and from there it connects to other nodes. And hub implements the coherence protocol. Okay. So, you can see in this picture we have a hub connected to this Xbow module.

Every Xbow module is an 8 port crossbar because it has got 8 ports, 1, 2 going to the hubs, 2 ports going to a graphics processing unit and another 4 connected to different bridges. Now these bridges are used for other I/O or connecting to the disks and other peripherals. So, the export switch is going to connect 2 hubs to the outside network. So, all the other I/O devices connect through the bridge to the Xbow whereas the graphics devices and the hubs connect directly to the Xbow. Every processor in this whole system right now I have got 2 hubs.

So, you will have 4 processors if you have multiple such nodes connected, you will have multiple such processors in the system. So, when I am saying I connect this one node to several such through a scalable interconnect, every I/O device is capable of accessing the memory location from any board in the system. Here I have 2 hubs, so I have connected 2 boards. If I connect 4 or 16, I am able to access the data from any memory and from any I/O device. Okay. So, that is the capability of the system because the Xbow and the hub together help us to enable accessing data from any module in the system. Okay. So, when I want to grow my network from a 2 hub system to multiple. In the previous diagram we had just 2 hubs. Now I am going to connect it to more.

So, this Xbow switch should be able to help me connect to other nodes. How does it connect? Now it has got 8 ports and I am going to connect one, one Xbow which is the router to another router. Okay. So, here I have shown you a bigger network. If we have a

4 node system then this is the Xbow router and that router connects to 2 hubs which are the 2 nodes apart from other peripherals it also connects to the next Xbow. So, I have got 2 routers and each router connects to 2 nodes. This way I have a 4 node system constructed. If you want to construct a 8 node system then every Xbow should now connect to each other in this format and apart from connecting to 2 nodes of its own, that is the 8 node system.

Beyond that when you go you have to construct a hypercube topology, that is it forms a cube like this to connect to a 16 node system. Now I have a cube of 8 router,that is 8 XBbows, every router is going to connect to 2 nodes. Hence I have a 16 node system which is equal to a 32 processor system. When I want to construct or double the size we will have 2 such 16 node systems where every router is now connected to 3 other routers. So 4 other routers rather. So, here I have 1, 2, 3 in my own 16 node plane and the fourth link will go to a different cluster. So, this is my within a 16 node cluster and this goes to another 16 node. Okay.

So, you can see this is the link which connects from this 16 node to the other 16 node. Expanding it further, we can have a 64 node system, but for that you will need a meta router to connect because we have got only 4 links. Hence instead of this link connecting to this cluster, it connects itself to the meta router and through the meta router it is going to the next 16 node cluster. Okay. So, these are some latency figures how the physical wires are connected and overall we have got 4 virtual channels here, one for request to reply and other 2 for priority and input/output. You recollect from the previous lecture that virtual channels are used for avoiding deadlocks and sending logically different pathways for different types of messages. Okay.

So, this origin architecture is has to implement the directory protocol and what type of a directory protocol it implements. We had seen 3 varieties, one was a strict request response, one was intervention forwarding and third was reply forwarding. So, Origin architecture is implementing the reply forwarding type of directory protocol. So, when a request comes to the directory, it is not going to send a reply back to the requester, but it is going to send an intervention to the owner if there is one and this owner will then send a reply back to the home as well as to the requester. So, that is the reply forwarding method.

Apart from that, to optimize on the latency the Origin architecture says that, if the directory needs to fetch the data from the owner, in parallel it also sends a data to the requester, just in case the data is useful. So, this is called a speculative reply and for this it has to do a parallel look up of its own memory module to forward the data to the requester. This speculative data which reaches to the requester may or may not be used by the requester because in case the owner had a dirty copy, the dirty copy will be forwarded to the requester which will then be utilized. For the protocol the two processors in the node are treated separately. So, it is not a single processor, but or a single node.

Every node has got two processors. So, when I am talking to one processor, we can refer directly to the processor or to the node and when I refer to a node, I am referring to the hub because then the hub has to broadcast the request to both the processors. So, the unit of visibility is either the processor or the hub and this unit will be more clear when we look at the directory structure in the next slide. Okay. So, we have reply forwarding and speculative memory operations. Every processor is treated as a separate processor and not part of the node because they do not do snooping with each other. Because they do not use snooping and this will also mean that we cannot do cache to cache sharing between the two processors within the same node.

And we have already understood how the hub works, it is shared by both the processors and the directory apart from using reply forwarding, it is going to use the busy state to indicate or to help in the serialization and correctness aspects. So, when the directory is servicing a request, it goes into the busy state, hence future request will be NACKed to resolve any race conditions and provide serialization with respect to the home node and these NACKs will also help us in deadlock and livelock solutions. Okay. So, now we have to think about the different states of the protocol. We have two processors in the node which have caches. So, cache is going to implement the MESI protocol in this particular architecture.

So, you can recollect how the MESI protocol works. Now apart from these MESI four states with the cache we also need to keep states with the directory. Now you would say, why do I need to keep states in the directory because we never did this in the snooping protocol. What happened in the snooping, when there was a cache miss we sent it on to the bus every cache snooped on to it and then using the shared wired-OR signals, it was indicating the status of the cache block across different nodes. So, the state was given implicitly using snooping, but in a director,y we have to give an explicit information about the states.

So, if a cache is in a M or E or S state, the directory should have the correct information about this. Hence, the directory also has to maintain state of these shared cache blocks. Now what could be the different states? In Origin protocol we have got seven states. One is an unowned, unowned as the word says, there is no sharer of the state of block, the block is there in the memory, but nobody is sharing it. Then the second state is shared, where there are multiple one or more copies where the data is in read only manner.

We have a third state called exclusive, exclusive as the word says exactly one sharer is there in the system. Now, here the difference could be that is this sharer the read only copy or a dirty copy. Well, the directory does not keep this information, it only says I have exactly one sharer for this block and it leaves it to the sharer to identify whether the copy was dirty or clean. So, these are the three states and related to various actions of the directory we would have three pending states or busy states. So, when the directory is busy serving a read reply it will go into a busy state for reading, busy state for writing and busy state for uncached reads because if there is a DMA or an I/O data transfer these are not necessarily related to coherence and hence these are those blocks which will not be cached. They will be simply sent from the memory to the other node.

So, we will have three pending states for that. So, that covers six states and the seventh state is a special case with this directory when I will have a page migration from the memory of this node to another node. So, when the data moves from my memory to another memory, the directory entry related to my memory bank has to be deleted. So, when I am doing this, the directory also goes into a busy state, but that is not related to coherence, hence it is called a poisoned state where the directory stops all the requests, moves the page and then restarts the accesses. Okay.

So, we will look at all these states now. So, we have seven states: unowned, shared, exclusive and three pending states. In the shared, the memory copy is always valid, but in the exclusive we cannot guarantee that the memory copy will be valid or not because when a block is in exclusive state with one cache that cache will have the permission to either modify the block or not. So, that is the E state in the MESI protocol which immediately moves to the M state. Okay. And then we have busy states to indicate that the directory is currently servicing a previous request. It has not yet finished it and hence to guarantee serialization, it would go into the busy state and not service future transactions until the ongoing request is complete. So, these are the three busy states, busy for an ongoing read, busy for an ongoing write.

Now write requests could be write that is a pure write that is read exclusive or if you already have a copy, it could be an upgrade request. And the third is when the block is being accessed by I/O or by DMA for an uncached read. So, we will have to consider the first two cases related to the coherence protocol. The seventh state is the poisoned state which is used when the page gets migrated from one node to the other and the TLB has to be updated. So, during this the directory goes into busy state. Again we are not going to deal with this particular seventh state. Okay.

Understanding the states, now we have to look inside the directory structure, how is the directory information stored. So, directory is stored as a flat memory based. When you say flat memory based, we have a bit vector. What is the size of the bit vector? We have a 64 bit vector. If I have 64 bits, I can address 64 nodes because if you can recollect a bit vector essentially has a one to one correspondence between the index position and the processor ID.

So, if this bit is 0 the cache block is not shared in the processor, if this bit is 1 then the block is shared in that particular processor. Now with 64 bits I can only address 64 nodes. But in the first few slides we saw that there are 1024 processors possible to be connected in an Origin 2000 system. So, we have to now see how can I manage a 64 bit entry to cater to a 1024 node system. Okay. So, with this 64 nodes I can have three methods of three meanings associated with it.

One is the 64 bit entry is an exclusive entry. What does exclusive mean? It is not the exclusive state. This exclusive entry means that there is exactly one processor holding the block and when I have just one sharer why to maintain the bit vector with several zeros and a one, but I would simply store a pointer to the processor ID using this 64 bit. Now it is sufficient for any system because if I have a 64 bit entry I can have 2 power 64 coverage. So, that solves my problem when I have exclusive pointer. What happens when I have different sharers that is it is actually a bit vector.

So, the presence bits will indicate the sharer identity but now here I cannot go beyond 64 node system. So, we will see how to solve that using our overflow schemes which we studied that is the coarse vector format where every bit in the bit vector will now represent multiple nodes. Okay. So, in a shared bit vector the bits correspond to the nodes and not the processors. Now you need to remember this that I have got 64 nodes which I can connect and if I can connect 64 nodes a 64 bit vector will cover 128 processors. Okay. So, a bit vector is not covering the processors, but it covers the nodes.

So, if I am saying my bit is equal to 1 and if I send some message to this particular node then it is the job of the hub in the system to transmit or replicate our request, our coherence request to both the processors. Because the two processors are not snoop coherent. In case they were snoop coherent then the hub would simply put the message onto the bus and the both processors or any number of processors within that system could pick up that message, but now they are not. We actually have a processor P1 another P2 both of them connected to the hub. So, any coherence message which comes from outside, hub has to broadcast it to both of them, okay. So, invalidation or any request which is coming will be broadcasted by the hub, okay.

How are the bit vectors stored? The bit vectors are actually divided into two partitions. We have a 16 bit format in case you have lesser processors you are okay with having a 16 bit format and do not need more storage, but if you want a full 64 bit vector which can cover more processors then these extra bits that is the other 48 bits will be stored in a external directory memory, okay. So, we will see it here. So, this was the one node with two processors, the hub. Now that main memory had the data and the 16 bit directory entry, only 16 bits are stored with the DRAM.

These are the memory bank controllers and these are the second level caches. So, these 16 bits are with the memory. If you want a 64 bit vector the extra bits will be kept in this extended directory, all right. So, these are with the DRAM and these are in the external storage. This is how we will maintain those 64 bits. In case I am storing a pointer, now this pointer is to a processor and not to a node.

Bit vector points to nodes, pointers that is exclusive pointer points to the processor IDs. So, if I have a smaller system, I can accommodate using 16 bits. So, we will directly point to a processor within this node. So, this node has got two processors and a pointer will now point to a particular processor and not to the node. Again if you have a bigger system you can go for a 64 bit exclusive pointer which can point to any processor within this.

So, this router has got two nodes, each node has got two processors, okay. So, that pointer is going to point to this particular processor and not to the node. Second one is when I have a shared state, it is a bit vector, but bit vectors point to nodes. So, here I have two nodes and the pointer will be pointing to the node, any request from this will be then broadcast by the hub to the two processors.

So, bits are pointing to the nodes. Now what happens if I have more nodes in the system? With 64 bit format I can go only up to how many nodes in the system? 64 nodes, which is 128 processors, but I have 1024 processors possible in my system. Hence I have to group them together. So, if I have a P-node system, we can have $P/64$ as the group size. And for our example with 1024 processors, there are two processors, in every node which makes it 512 nodes and which further makes it 8 nodes in one set. So, when I have a bit vector every bit is going to point to a cluster of 8 nodes. The system will then dynamically choose between the coarse vector and the bit vector. Why because if the cache block is shared between limited number of nodes, then I can accommodate using a plain bit vector. I will go for a plain bit vector.

If the sharing goes beyond the capacity of a plain vector, then we will shift to a coarse vector. So, this is done at runtime by the directory controller. So, same thing here, for 1024, this becomes a 64 bit vector where every bit, every bit in this is now going to point to 8 nodes and when I say 8 nodes up to 16 processors. So, if I am going to send an invalidation to this, this invalidation will be received by 8 hubs and each hub will send it to, broadcast it to the two internal processors. Okay. So, we have seen the architecture, the directory states and overall how the storage is managed with the directory.

In the next lecture we are going to see how the protocol actually works. So, we will stop this lecture. Thank you so much.