Parallel Computer Architecture

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

Week - 08
Lecture - 44

Lec 44: Directory Protocol optimisations

Hello everyone. We are doing module 5 on Scalable Shared Memory Systems. This is lecture number 5, where we are going to discuss Directory Protocol Optimizations. Optimization about what? About performance. So, let us discuss performance to begin with. Right.

So, we are discussing directory coherence or overall a set of network transactions is the topic and if I want to improve performance of this, I need to consider various aspects of a network transaction where we can address issues to improve the performance. Alright. So, network transactions in a normal message passing based interface would go through, one computer generates a message which goes through the network to the other computer. So, that is done at the software level.

But when I am talking of a shared memory system where implicit messages get generated on behalf of the program, they are automatically generated by the protocol and not only that, but not by the processor, but by the communication assist or the cache controllers. So, they are going to generate these network transactions. These are very tiny transactions, that is, you send a request for example, a bus read request goes and response of data comes back. So, very tiny transactions of small amount of data and at times you get a data response back. Right.

So, the size of, the number of transactions may be more, but the size of every transaction is small. And when these transactions travel through the network, they go through the processor , that is the requesting processor has a read or write miss, it goes to its cache, then moves out from the cache controller to the communication assist, then outside to the network. Right. So, that is the path. Once it goes on the network, it reaches the destination node through that, either it reaches to the required processor, if required, but most of the time the destination processor may not be involved because if you want the data block, the cache controller or the communication assist at the destination can itself do the job for us.

Right. So, and we also need to take care that the amount of time it spends on the network, that is the network latency, the transfer latency should be optimized. So, these are the parameters we should be considering when we consider improving the performance of the system. All right. So, we are going to address this using three types of or handle this using three issues which are we will try to optimize the protocol.

We will see if can we arrange the machine organization which will be more efficient. Every organization is good, but which is a better one and can I have specialized designs for the communication assist or the parameters. Okay. So, we are going to see more of details of protocol optimization and quickly a brief review of the other two topics.

Okay. So, performance we are saying that cache coherence protocols are going to generate network messages atomically and sorry automatically and they are having a small size. Where are the overheads into the system? The overheads are with the requesting processor, the network delay and the communication assist at the endpoints. So, these endpoints, the processor at these endpoints may not be involved because the CA that is the communication assist at the endpoint would do the job for us. So, we need to concentrate on how do I improve the network delay and how can I reduce the involvement of the CA or how to make the CA more faster. Right. So, for this we are going to look at optimizing the protocol, some machine organizations and specializing the communication parameters. Okay.

Okay. So, protocol optimizations I am going to do in detail in the next few slides. We look at the point number 2 because we are going to do a high level discussion about the other two points. The second performance optimization was high level machine organization. So, machine, you have the processor, the cache, the directory, the memory and the network. So, these are the components and when I want to implement a directory protocol or any coherence protocol, we need to be aware that the request transfers from one node to the other through the network and having a very large system of multiprocessor nodes.

We could have a hierarchy of nodes connected, if I have a hierarchical protocol or I will have a network which has multiple processors, another one, another one which consumes this whole thing and so on. So, there could be hierarchical arrangement of the processors and if I have more and more depth in my hierarchy, it is going to take more time. That is my protocol is going to go through several levels in this hierarchy and incur more latency. So, to avoid this, we normally would say that if I have a two level hierarchy that would perform much better. So, two level hierarchy is that you have a global network and multiprocessor nodes are connected to this network. So, we have a global network and I have nodes connected to this and each of these nodes is a multiprocessor node.

So, there are n processors in every node. Within the node you can have a snooping or a directory protocol. You may say that well snooping is going to take time within this node also, but still it is popular and if you have a commodity server available which supports snooping, you might want to just plug it into your system and use it. Hence snooping is still being used, right. So, for a machine organization we would prefer a two level hierarchy, although other options are also viable.

The third one hardware specializations to improve the communication parameters. So, here the communication assist which we discuss about, we want to make sure that the assist is involved for a shorter amount of time, because the assist should do the work very quickly. So, how do I improve the communication assist or what is its role that I can address. So, this CA, when it gets a request from the network about a read write or any transaction, it has to look up its directory and its associated cache. So, this has to be done in a quicker way and it should be able to do it on its own.

So, essentially the memory, the cache controller or any memory related modules should be tightly integrated with the communication assist. They are definitely integrated with the processor hierarchy, but we also want the CA to be closely associated with these modules, that is it should have the power and control of accessing them more freely. Okay. So, there has to be a tighter integration. Then we should also see, that if the CA is busy in serving a transaction and suppose newer transactions come, the CA module should be capable of serving the new transactions in parallel to the pending transactions. Right. So, it again has to be itself a parallel process that it is serving block number A, in the meanwhile request for block B comes it should again be able to serve that request. Okay.

Then the directory to which it is going to look up, it should be fast and to make it fast, we can say that store the directory in a SRAM device rather than a DRAM memory. So, we can use SRAM for the directory. That is one option. Another thing is we had long ago discussed that, if the memory knew that the particular cache block is being cached by others or not, the memory can maintain a bit about it. Right. So, we can use this idea here and not disturb the memory module as such, but maintain another lookup table in SRAM with every address and say that these particular addresses, whether they are cached or not, whether they are dirty or not. So, that the communication assist can quickly see through this and decide the further actions.

So, overall we want to see that the communication assist is able to do things quickly and parallelly for multiple transactions. If your assist is going to take more time or the design is more complex maybe go for a pipelined design, so that you can overlap actions. Apart from using an SRAM directory and lookup table for SRAM. Okay. So, moving on to protocol optimizations. So, protocol optimization's goal is that I should reduce the number of network transactions and the number of transactions on the critical path, on number 1 and number 2. By reducing the number of network transactions what are we achieving? The overall bandwidth demand reduces because I am sending lesser transactions, so the bus is available for other transactions to take place and so overall bandwidth demand is reduced and you get more utilization of the same resources.

Similarly, the communication assist will also, involvement will also reduce because if the number of transactions is less, we will have more time that the CA spends on more number of transactions than on a single transaction. The second objective is reducing the number of transactions in the critical path of the processor, that is once you start one transaction several messages will pass from the sending processor through the network to the receiving processor and then back. So, we want to reduce this number, so that the processor starts processing as early as possible and this is possible if I can kind of divide the transaction into small pieces and make sure that I can overlap the complete activity instead of doing it more sequentially. So, these two we can achieve independently and it also depends or our approaches will depend on the way the directory is organized. That is how, how am I storing the directory information. If I am storing the directory information using memory, that is a memory based bit vector I can send many things in parallel.

For example, if you are invalidating the cache blocks from a bit vector you can simply send parallel messages to all these sharers. These can happen in parallel whereas this is not possible in a cache based system. So, but in a cache based systems, you can do something else to improve the performance. So, in both cases directly or indirectly we have options to improve the protocol. Okay. So, first we will look at optimizations in a memory based system and later optimization in a cache based system.

So, this is an overview slide one point reference for you, but we will do these three diagrams one by one. Protocol enhancements for latency. Here, I am going to discuss communication methods of three types, strict request response, number two is intervention forwarding and third is reply forwarding. While doing this, I am going to use the terminology, L for a local node, H for a home node and R for a remote node or a dirty node. I hope you can recollect, we had discussed these terminologies in a previous lecture.

We are going to do these one by one. So, strict request response. Here, as the word says strict, it is going to be when you send a request you are guaranteed to get a response and you can do nothing else in the middle. So, when you send a request you expect a response. So, I am going to take a scenario, here, that I have a local node which is also the requester node, a home node and a remote node. We have three nodes to consider. Whenever there is a cache miss, the local node will then or the directory controller will send request to the home node for the block.

So, the first action is a request. As it is a strict request response you are expecting a response back. So, what should the home do now. Because home knows that, I am just taking a scenario that the block is probably dirty in the remote node and home has to somehow make sure that L gets the data from R. So, either home gets the data or home tells L to get the data. Because I am following a strict request response, in response to 1, H has to send a response. So, 1 was a request, so 2 is a response and what do you think will go in this response. Because this one transaction has finished and now L should be able to go to R to fetch the data. So, essentially in this response, I am going to forward the identity of R.

So, identity of R will be sent by the home to the requester because it will use this information and then go to R as transaction number 3. So, third is another request. So, this request says that L says to R that, give me the data block and R gives the data block to L. So, as part of this, 4 is a response. So, this response carries the data because the data block is transferred from R to L. Right.

So, now you would wonder that the new data, the fresh value is moving from R to L, but what about the home node. Suppose L had asked for writing then it does not matter that home still can hold the stale copy, but if L had asked for reading home needs to be updated and who would update home. Very likely here, it is the responsibility of R. So, R has to update home as part of the fourth transaction. So, I can divide this as 4a and 4b and say that 4b, it revises the content of the local node. Okay. So, this is how a strict request response is implemented. You send a request, you get a response.

So, 1 and 2 and then 3 derives into 4. So, if you look at this diagram you will see or rather I would say you pause the video and count the number of network transactions, count the number of transactions in the critical path and then decide whether this is an optimal method of doing it. Here I have 4 network transactions in the critical path because I need to do 1, which results into 2, that results into 3, and then a 4a. Because once this complete thing is over, the processor at L can start processing. The 4b transaction can happen in parallel to 4a, because the processor at L need not wait for 4b to finish because it is going to happen in background. Overall we have 5 network transactions, out of which 4 are in the critical path.

Can we improve this? Okay. So, this can be improved by using the next method of intervention forwarding. Okay. Same scenario we have L, we have H and we have R, L sends a request to H. Now it is not strict request response and we are doing this to reduce the number of transactions. Here H says that let me go to R and fetch the data. So H goes to R and this message is called an intervention.

Why intervention? It is also kind of a request but H is generating this request because of another request from L. So, because that first transaction happened, this is a reaction to the first transaction , hence it is called an intervention. So H sends an intervention to R. What should R do? Actually here in 2 it asks R for the data. So you get a response from R along with the data and then H forwards that data to L. Okay.

So here I have 1 transaction 2, 3 and 4. So only after all these are over, can the processor move forward. Hence in the critical path, I still have 4 transactions. And how many total transactions in the system? Even then we have 4 total transactions. If we compare it with the previous one, here, we had 5 total transactions out of which 4 were in the critical path. We have improved a little by making the 5 to 4.

So we have 4 transactions as the total and out of that again still these 4 remain in the critical path. Okay. So this concept of intervention is, it a request sent as a reaction to another request and this request is directed to a cache and not to the memory. Because L sends a request to H which is a memory whereas H sends a request to R which is to be addressed by the memory. By the cache at R. Alright, so we have improved from 5 down to 4. Critical path is still 4. Can we improve further?

So let us see the third method called reply forwarding. So reply forwarding, again I will take those 3 nodes, a request. Now, H is going to send an intervention because it has to go to R and this method had worked for us, so I will still use this. I will just use a shortcut word inter here.

So 2 sends an intervention to R. Okay. So how do I improve from the previous one? I will go back to the previous slide. Here you can see, that R sent an answer or response to H and then H forwarded that response to L. How could have we done better than this? We could have done better, if R was able to give the data to H and L both in parallel . So that will reduce the number of transactions in the critical path because then we can overlap some transactions. Okay. So here, let us try that. I will now make R send a response in parallel to both of them. So I will make this a 3A and a 3B, so this is called a response because it goes to L in response to that request number 1 and 3A, I will say it is a revised message because it is only going to go and update the data.

So how many transactions in the critical path? Pause the video and calculate. So we have 1, then you have 2 and 3B. Right. So you have 3 transactions in the critical path, total network transactions is still 4 because 3A and 3B count as 2 transmissions. So total transactions is still 4 but in the critical path we have reduced it from 4 to 3. So this is popularly called a 3 message miss type of a communication, the reply forwarding.

Okay. So the question is 1, 2, 3 which one to use? The one strict request response was the slowest one, so we do not want to use it. So how about 2 and 3? 2 was intervention forwarding, 3 was reply forwarding, they are not strict request response. Now when we are doing away with the strictness, we have problems related to deadlock. Because if I am sending a request and getting back a response, there is less chance of waiting . But if I am sending a request, that is sending another request, probably another request and eventually we will get. So the delay increases, the number of modules involved in this transaction increases, which leads to or which complicates the deadlock avoidance. So these intervention and reply forwarding are good, but they may complicate deadlock avoidance.

Intervention I would say is an intermediate design because it is good at latency and traffic but still the critical path has more transactions than the reply forwarding. Another disadvantage is that you have several such transactions waiting at the home. Okay, I will just draw it here for a quick reference. So this was the intervention and so when H sends an intervention to R, it could be sending such interventions to multiple R's for multiple other requests in parallel. So this H ends up remembering that I have sent these many interventions and it has to wait for all of them to respond to reply back to the requesters.

So there could be several outstanding interventions at the home and at max it has to keep track of K * P where there are P nodes in the system and every node could send K requests. So you need that much buffer space and management of these K outstanding requests. Whereas this problem is not there in reply forwarding. So this is about, so the marked zone is about intervention forwarding and in reply forwarding what happens? The H need not remember but R is going to send the answers directly to L. So H need not remember that it has to forward, because the role of H finishes when the revised message arrives.

Hence it is much efficient and most of the systems use reply forwarding. Of course we need to take care of the deadlock issues. Right. So that was memory based optimization. Now cache based. Again in this, what was cache based? You had a linked list or a distributed linked list across every node for all the sharers.

Now when I send invalidation message for a particular block, we first go to the directory that is the home node. From the home node you will know the identity of the head node that is the first sharer. You have to go invalidate that. From that first sharer you will get identity of the next sharer and so on. So you have to traverse to this linked list invalidating every block in the system of sharers and then return back and that is bring the acknowledgments back home.

So this is the overall protocol. How will I optimize it? Even here, we are going to use those three methods, strict request response, intervention forwarding and reply forwarding. So let us do three of them one by one. Okay. So here there is a write request because we can only optimize the worst case that is the invalidations were passing through the system. Okay. So let us do this. Here I am going to say, I will abstract the other requests out because I am assuming that I start from the H node just to simplify the diagram.

So scenario is processor has sent a write request, that goes to the local node that is L. L sends a request to H and H has a pointer to the first sharer. So H is pointing to sharer S1, S1 is pointing to S2, S2 is pointing to S3. I have taken this much example.

So there is a linked list from H to S1 to S2 to S3. It is a write request, so I have to send invalidations. So H is going to send first message invalidate S1. Because there are multiple sharers, the block is clean, hence we do not want a data response back, we simply need an acknowledgement. So it sends invalidation to S1.

We are dealing in the scenario of strict request response. Okay. So this is my protocol. Under this protocol, for every request a response has to be generated. So 2 is going to be a acknowledgement from S1 to H. If we do this the link between S1 and S2 would be not known to H.

So H will not know how to invalidate S2. So to overcome this, when S1 sends an acknowledgement along with that, it has to send the identity of S2 to H, because the next transaction H is going to send an invalidation to S2. S2 is going to send a response which is ACK in. Okay. So the number this should be 3 and this should be 4. 4 is the ACK, along with that you will get identity of S3, hence H is going to send the next message to S3 in a message number 5 of invalidation.

S3 sends a response to H of acknowledgement. 6 is ACK. With this no identity comes, that is you send a null, so that H knows that all sharers are invalidated. Okay. So in this example calculate the total number of network transactions and those in the critical path. So you can pause the video and solve this. So total transactions in the system is 6 in my case.

Here it is 6 which is equal to twice the number of sharers. And how many are there in the critical path? I would say the same number. So which is also to the total number of transactions in the critical path. Okay. So a neater diagram is given here. Next is intervention forwarding then we will also see reply forwarding. Okay. Same number of sharers I will do them, one by one below the other, these two examples we have H S1, S2, S3.

H sends an invalidation to S1. Okay. We want to do intervention and not reply immediately. So intervention is S1 should do something on behalf of H. So S1 sends something to S2 as an intervention. So it sends one message here.

In addition to this, S1 has to tell H that it has invalidated itself. So I am going to have two things in parallel. Here I will send the acknowledgement. S1 sends acknowledgement to H that it has invalidated the block.

S1 sends a request to S2 of invalidation. This 2a is the intervention. In response to 2a, S2 is going to do two actions, one to S3, another back to H. This is going to be 3a, 3b. 3b is an acknowledgement and 3a is an invalidation request from S2 to S3. At the end, S3 has no more sharers, that is the linked list stops here.

Hence it simply sends a acknowledgement 4. Alright, so total number of network transactions and those in the critical path. How many are there? Total transactions is, for me 1, 2, 3, 4, 5, 6 and those in the critical path, you have 1, then 2a, then 3a and then 4. The 2b and the 3b, they can finish in the background by the time the 3a and 4 happen. Okay. So in the critical path I have 4 and in total I have 6. So what is the formula? Here it is 2 times the number of sharers and critical path is the total number of sharers plus 1, if S is the number of sharers. Okay.

So that was intervention forwarding. Now we will do reply forwarding. H, S1, S2, S3. H sends an invalidation here. So overall, look at the red diagram and let us build a better blue diagram at the bottom. What can we optimize? When a request goes from H to S1, we were saying, S1 does the job on behalf of H by doing the 2a transaction and also replies to H. So we could say in a way that this 2b and this 3b they happened in the background, they were not in the critical path, but they were still consuming the bandwidth, because it was contributing to the total number of transactions. Can I avoid the 2b and the 3b? I would say yes, because if S1 has forwarded it to S2, it is, the protocol guarantees that it will delete the block from S1.

So the 2b acknowledgement, I would say it would be a redundant action, if we can somehow guarantee that, at the end H knows that everybody has deleted. So let me not do 2b and see what I can do in future. So S1 simply says okay, I will invalidate. So it guarantees that it invalidates its block and then just forwards that request to S2 , because S1 has information of S2.

It does the same thing here. Then S3 has got no more sharers after that. So S3 has to send an acknowledgement. Now here is the trick that S3 would now directly send an acknowledgement to the home node as action number 4 and when this ACK comes, the protocol design says that H understands everybody in the system or all sharers are invalidated. Okay. So with this I have avoided the 2b and the 3b. So in the critical path how many do I have? I still have 4. Right.

But the total number of transactions is also 4. So the total is 4 and in the critical path is also 4. And what's the formula? It is S plus 1 because you are going to send 1, 2, 3 messages, that is 3 sharers, and final ACK adds to the last one. So S plus 1 is the critical path. All right. So we have seen optimizations on memory based and cache based systems on three different varieties of protocols and we conclude that reply forwarding is the best performing one.

Of course it has problems related to deadlock avoidance or complications which we need to be careful and handle it ,and we can discuss this when we handle the correctness aspects. Okay. So with this we stop this lecture. Thank you so much.