Router Architecture

Lecture #10: Monday, May 7, 2001
Lecturer: Prof. Bill Dally
Scribe: Anamaya Sullery, Allen Ong, Gaurav Chandra
Reviewer: Kelly Shaw

1 Router architecture

A block diagram of a wormhole router is given in figure 1. In general, the router can be divided into its datapath and control section. A brief description of each block is provided below.

Input Units : Input ports buffer incoming flits and store the state information for each input channel. Refer to Section 3 for more details on the Input unit.

Output Units : Outgoing flits are received from the switch and then forwarded to the downstream router. Contains state information indicating status of the output channels. Refer to Section 2 for more detail.

Virtual Channel Allocator : The Virtual Channel Allocator arbitrates between bids for output virtual channels by the incoming head flits. Many implementations of routers locate the input port state registers within this block.

Switch Allocator : The Switch Allocator arbitrates switching requests to connect the input ports to output ports and schedules the Switch.

Switch : An interconnection network, which connects the input ports to output ports based on the configuration determined by the Switch Allocator. This would typically be an N-input x N-output crossbar.

2 Router functionality example

An example was provided in class to illustrate the basic functionality of each router sub-block. The example considers the processing of a 3-flit packet arriving at the router input. Figure 2 illustrates the pipelined state diagram for this packet.

At **cycle 0**, the head flit arrives at the input unit carrying the packet header. This information is passed to the routing engine to obtain the set of virtual channels that can be used for this packet. The Global State at the input unit is set to Routing (RT) and the Next Buffer Pointer is updated to point to this flit.

At **cycle 1**, the head flit completes the routing stage and the resulting set of virtual channels is stored in the Routes (R)field. Subsequently, the VC Allocator arbitrates for

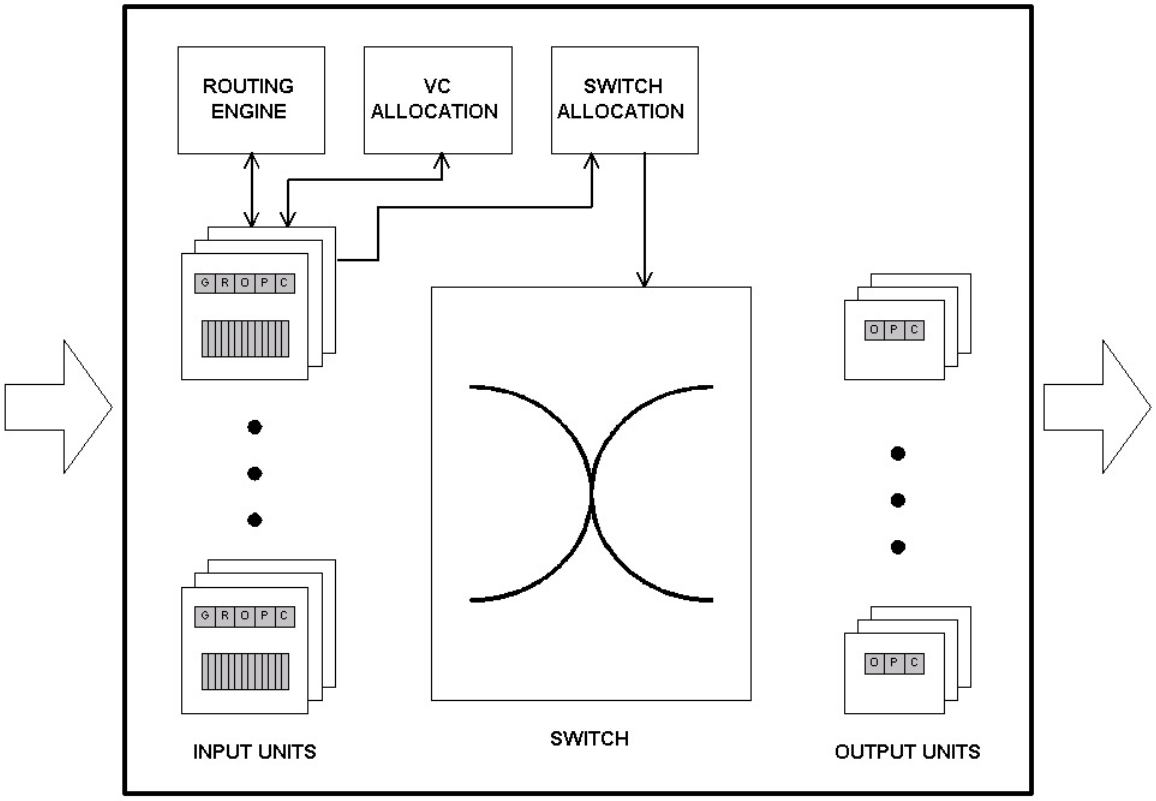

Figure 1: Block Diagram of Wormhole Router

an available VC that is also present in the routing list. The body flit arrives at this cycle and is stalled because VC allocation has not been completed yet. The VC needs to be allocated before the head flit can bid for Switch Allocation. The Global State at the input unit is set to VC Allocation (VC)

At **cycle 2**, an output VC has been allocated to the packet. The result is stored in the actual output VC (O) field. The head flit then starts bidding for switch allocation. If the Credit field is non-zero, the Switch Allocator performs the scheduling for switch bandwidth to transfer the flits. The body flit continues to stall, waiting for the head flit to complete switch allocation. The tail flit arrives at this cycle and is also stalled. Stalls are needed here to avoid misordering of the flits. The flits must traverse the switch in order as they do not have sequence numbers. The Global State at the input unit is set to Active (A) and the state of the output unit is set to Busy (B).

At **cycle 3**, a path has been allocated through the switch and the head flit is transferred to the output port. The Next Buffer Pointer is updated to point to the body flit and the credit count is decremented. The body flit bids for switch bandwidth while the tail flit continues to wait.

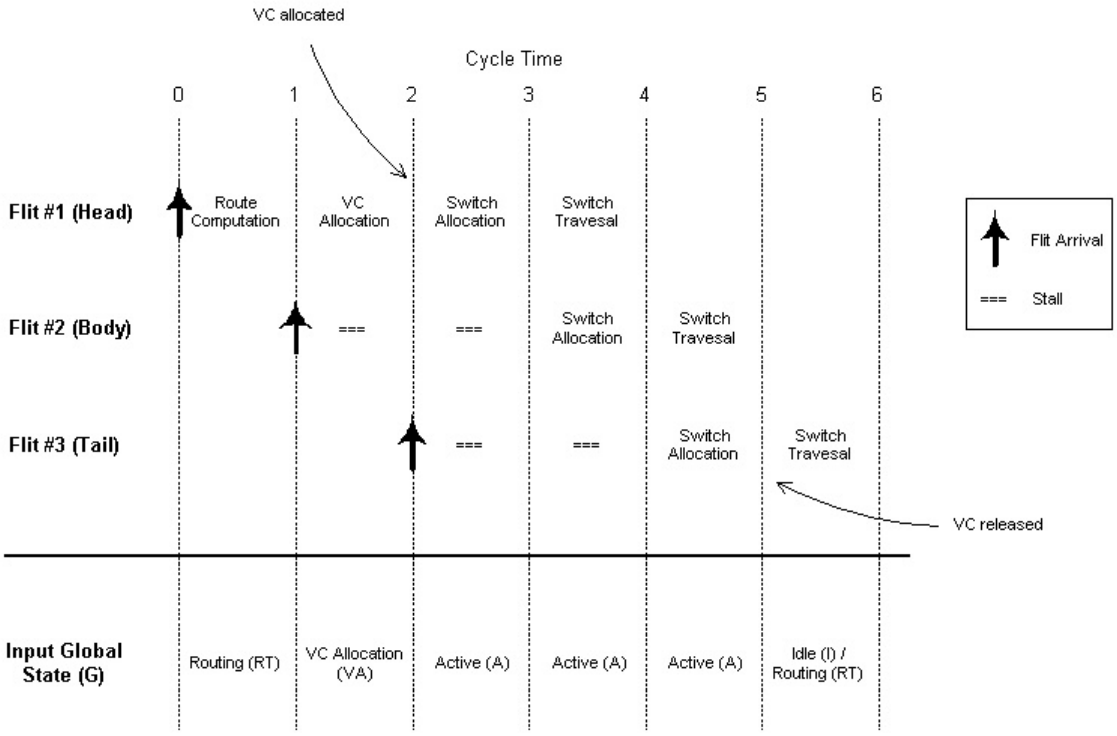

Figure 2: Router Pipeline Diagram

At **cycle 4**, a path has been allocated through the switch and the body flit is transferred to the output port. The Next Buffer Pointer is updated to point to the tail flit and the credit count is decremented. The tail flit bids for switch bandwidth.

At **cycle 5**, a path has been allocated through the switch and the tail flit is transferred to the output port. Theoretically, we can deallocate the virtual channel at this stage and transfer other packets via this VC. A more conservative approach will be to wait until all the flits are acknowledged by the downstream router. This is determined by waiting for the Credit count to return to its maximum.

3 Input unit

The router contains a series of input ports. Multiple virtual channels can be associated with each input port. For each associated input VC, there is buffer unit to hold the flits of the arriving packets and a state table to indicate the status of that input VC. The state table consists of the following 5 fields :

G - Stores the Global state of the input VC. The 4 states are described below and figure 3 illustrates the state diagram.

- Idle (I) : Idle or inactive, when there are no new head flits arriving.
- Routing (RT) : Route computation. The destination is determined from the head flit by the router to produce a route (set of routes in case of adaptive routing). There can be stalls at this stage if the routing table is shared and more than one head flit arrives simultaneously.which will lead to a stall.
- VC Allocation (VA) : Virtual channel is being allocated. Stalling occurs at this state when VC is not available.
- Active (A): In this state the VC bids for the switch. It remain in this stage until tail flit is done with switch allocation

R - Stores the route(s) returned by the Router. It could be an output VC or a set of output VCs (in case of adaptive routing)

O - Actual output VC that has been granted.

P - Contains the next buffer pointer and the last buffer pointer. These pointers are primarily used to determine buffer occupancy (i.e. number of flits is same as difference between next buffer pointer and the last buffer pointer). The next buffer pointer is also used to retrieve the next flit from the buffer.

C - Contains the count of empty buffers on the downstream router (credit). A bid for the switch will only occur when the number of credits > 0 and, therefore the number of empty buffers on the next router > 0.

4 Output unit

At the output virtual channel of a router, we have three pieces of information, G, I and C, analogous to the input unit. We now present a description of these.

G : This is the global state, just like at the input side. The difference here is that we just need two entries : idle and busy. The purpose of this state is for the Virtual Channel Allocater to know which channels are free while it is allocating the channels.

I : This is the input virtual channel that actually has the output virtual channel. We need a connection in both directions so that we can direct the credits that are coming back from next router back to the correct input state. Strictly speaking, we could do without the input virtual channel information. This is because input controller knows the output virtual channel it is sending to. It can appropriately configure the reverse crossbar switch used to send the credits back from the output. But in practice, output controller controls the credit switch to keep things simple.

It must be noted that to direct the credits, we actually have another, smaller, crossbar in the reverse direction below the main crossbar shown in figure 1. This is to direct the information of credit arrival back to the inputs. The credits actually get buffered at the output, though, and just get reflected at the inputs by this crossbar switch.

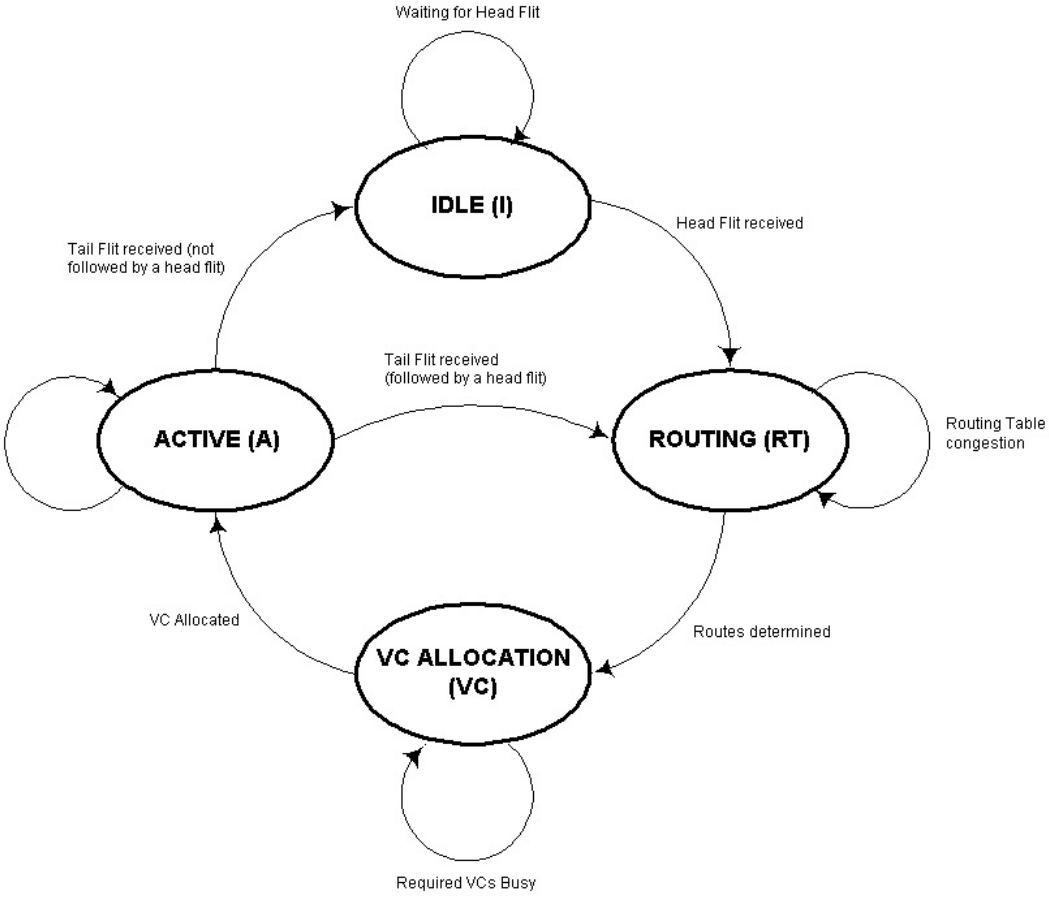

Figure 3: State Diagram for G

Sometimes, one can design the virtual channel allocater to do this task of directing the credit information.

It was speculated that one could store the credit information at the inputs rather than the outputs. Then we would not need to transfer it through a combinational path. However, it is good to avoid redundancy and store the credits only at the output and use a combinational path to transfer the information to the inputs. Suppose we have the number of virtual channels running into thousands. Then the combinational communication would be expensive because of excessive wiring demands. In such a case, we could copy the credit information at the inputs. We can not just keep it at the inputs, though, because when an input gives up a virtual channel credit count may not be maximal. Though there is no more data to send, the credit count would take some time (a round trip time) to get updated. But once we have given up the input virtual channel and a

new flow acquires it, we have no way of knowing about the old flow. Thus we need to keep a duplicate copy at the output. If we are maintaining a duplicate copy, we still need to keep the two in synchonization. Thus, we need a connection between the input and the output whenever output gets some new credit information.

Another question was whether we can put the credit count in the allocator. This would solve the problem of transferring the credit information since allocator is a global entity. That can certainly be done. Actually in practice, a whole lot of the state on the input side does get wrapped up in the allocator.

It was also pointed out that one would need to take care of the contention for the reverse channel. One idea is that bandwidth could be reserved in the future. But advanced reservation is not possible. This is because the return time of a credit is not deterministic and can vary with congestion down stream. In many cases, to avoid the credit delay due to contention, designers overprovision the return crossbar. The credit is really just one bit and this is not too expensive. The delay of credits can undermine the performance severly as we shall see later.

5 Stalls

The pipeline presented so far is the most general case. There are instances where we may not have all the stages. For example, a source routing scheme does not need the RC stage. If virtual channels are not used, then the VA stage is not present, and so on.

For the most general case of a pipelined router, we now ask the question : what are the possible stalls in this pipeline? A description of the possible stalls now follows.

Routing stall : As mentioned before, sometimes we do not dedicate routing hardware to every input but instead want to share it. We could have more than one packet wanting to use the routing hardware simultaneously, in which case, we would have a structural conflict. So the header flit coming in might have to wait an extra cycle before getting routed.

VA stall : Suppose a particular header flit wants to go out on a virtual channel that is being occupied by a particularly lengthy packet that runs into thousands of flits. In this case, our unlucky packet ends up waiting for a long time before its bid for the virtual channel is successful. This is an example of a stall in the VA stage.

The above two stalls are called header stalls, since these are specific to a header flit. These stalls keep the flit from entering the active state. However, once we reach the active state, we could still have stalls in further stages. These stalls could happen for any flit.

SA stall : If a particular flit wants to go through the switch and fails to acquire the resources to do so, it has to just sit in the buffer for that clock cycle. This is a stall in the switch allocation stall.

Figure 4 illustrates the above mentioned stalls in the pipeline.

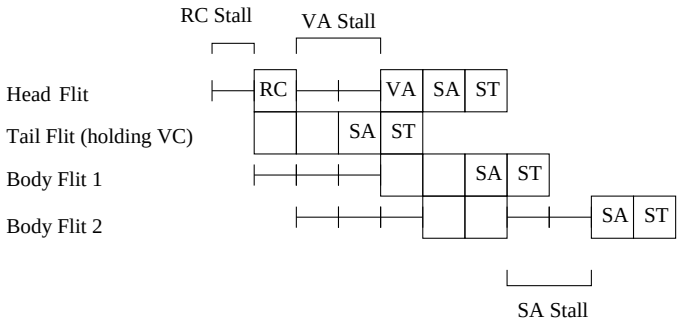

Figure 4: RC, VA and SA Stalls in a pipeline

What other things could cause stalls ? If our credit count runs dry we end up with a credit stall.

Credit stall : This would happen if we underprovision the buffer space at the input. To avoid this, we need to make sure we account for not only the round trip delay from the next stage, but also the pipeline delays of the flit as well as the credit that comes back.

In the pipeline, at which stage do we get to send the credit back ? We can conclude that the buffer occupied by a certain flit will be freed in next clock cycle only after the switch allocation has been done for that flit. Before that we are not sure whether the buffer will be freed up because that flit might not win the switch allocation. This leads to a delay in sending the credit back. This delay must be accounted for while estimating the buffer space at the input.

As an example of penalties if we do not consider pipeline delays, consider two switchs shown in Figure 5, and the pipeline flow. The wire delay between the routers is two cycles, so the buffer has been naively set to a size of 4, which is the round trip delay.

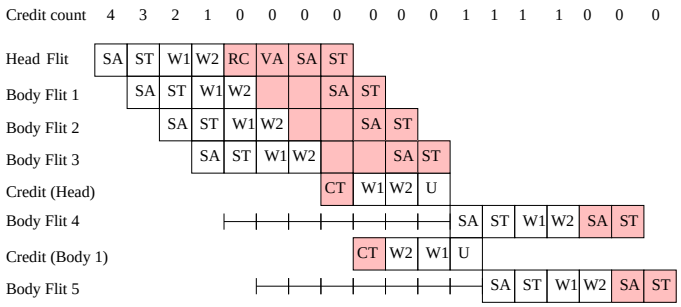

Figure 5: Credit Stalls in a pipeline

We get to send 4 flits before our count gets down to 0, preventing further flits. Meanwhile, the header flit from the white router flows through the pipeline and lands at the next (gray) router after 4 cycles. A credit can not be sent, however, until the SA stage is done. At this point, the gray router sends back a credit for the flit. We are optimistically assuming here that it can be done in one cycle. After traversing the wire, this credit comes back and it take one cycle to update the credit information. Again, this is an optimistic estimate. The credit comes in at the output side and needs to be transferred to the input. One also needs to update pointers and the credit count. As mentioned earlier, the credit information can be put in the allocator that is global, to avoid these difficulties. But it'd probably still need one clock cycle to know that the credit has arrived. It might be collapsed to get rid of the extra cycle only by some really clever design.

So number of cycles for the credit to come back is :

$$T_{cycles} = 4(pipeline) + 4(roundtripdelay) + 2(CT, U) + 1(startNextSA) = 11$$

As a result, we see 7 idle states in our router. After 7 cycles, we'll have another 4 flits going by, and so on. The duty factor is a measly 4/11.

People do build routers this way, taking only round trip delay into consideration. What they are counting on is that fact that we have a number of virtual channels at the input and that would make sure we keep things busy. However, a better way would be to keep a bigger buffer size, 11 in our example here.

Can we have further stalls ? We could, if we simply run out of flits to send at the input. As an example, suppose during the traversal of a flit, the header flit goes past a router. At some point later, the subsequent flits might collide with another packet wanting to go the same way. Flits would then be multiplexed and this would lower the duty cycle for body flits. This increases the delays for subsequent flits. When a head flit goes down a chain of routers, it reserves the virtual channels on the way. If the body flits are delayed as mentioned above, these channels sit idle and this is harmful to performance. We end up burning up a lot of virtual channels.

6 Flit format

The discussion of flit format is really motivated by the question of how can we piggypack credits on a flit. To answer this question adequately, we discuss a flit format in more detail.

In general, we could make a flit span multiple clock cycles. Suppose we have a 16 bit path (and thus 16 bit wide flit), and 16 virtual channels. Every flit needs a VID (virtual channel identifier). To encode these 16 channels we use 4 bits leaving only 12 bits for the data. This is an overhead of 33% and would be unacceptable.

As an alternative, we could have one flit spanning 4 clock cycles. An overhead of 4 bits out of 64 bits is certainly better. We could now put credit information into this

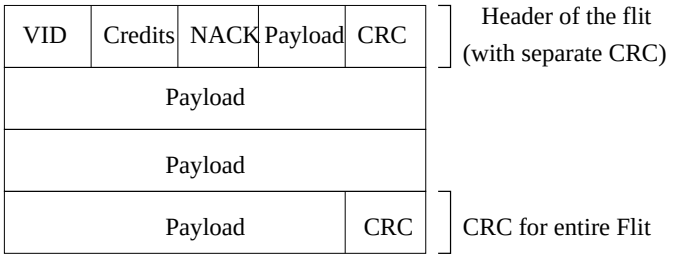

Figure 6: A sample flit format for a 4 cycle long flit

flit without much overhead. We need another 4 bits for this. We now have 15 possible virtual channels (one possibility is idle flit, too). So we need 16 values for credit field. We could have no flit but still have a credit.

We would also like to put in a error correction code for the flit. Sometimes people put a separate CRC just for the part containing credit information. To understand why, we need to see what happens for a single bit error in credit information. In general, we have two routers with two pieces of state C (amount of credit) and P (number of flits in buffer), respectively. If B is the buffer space we have, we need :

$$C \leq B - P$$

It doesn't have to be equal since this is conservative. The data transmission end can send more data only after the data receiver end sends it a credit. The data receive end frees the flit buffer (and hence a credit) before the tranmission end gets to know of it.

Now suppose because of an error we increment C. This is like a dormant virus, since right now the credit count may be big and we may be getting lots of credits back. But imagine that after, say, 10,000 cycles the sender runs out of credits and sends one packet too many. This would be dropped by the receiver and things would go haywire. It would be very difficult to find out what went wrong.

For this reason, people want to make the flit header extra robust. In case of a fault, one could do an error correction or ask for retransmission. It turns out that latter is easier to do. The sender end would just buffer the flits and roll back and retransmit in case of a Nack.

The complete schematic of a flit format is shown in Figure 6.

Obviously, we have a trade off in deciding the flit size. Too big a flit would lead to fragmentation overhead. Too small a flit leads to an increase in percentage of header overhead.

Another possibility for sending credit information is to dedicate a wire for sending back the credit, different from the set of wires we have for transferring the flit. This is a horribly inefficient way to do it. We are using precious resources and credit would only

go one bit at a time leading to high serialization latency. By our earlier apporach, we get the credit all at once.

At this point it was speculated whether we could do some kind of optimistic flow control, speculating we might get the credit. This would certainly help cut down the overhead of credit return. To do this you need lots of buffer storage on the output. At the output stage of a router, we keep buffer space only enough to take care or CRC violations and retransmissions. In an optimistic flow control, we'll have to store, for all virtual channels, the past few packets we have sent. If the number of virtual channels is large, this would result in large buffer overhead.

Another thing that was pointed out is that if we are piggypacking credits on data flits, at times we'll have to send idle flits just to carry the credit information. For big flits this would be inefficient. The neat answer is that we can cut down the fragmentation by noticing that the header part of the flit is self sufficient, complete with a CRC. If we send an idle flit, we can start an idle flit right in the next cycle.

Sometimes the designers would want the flit duration fixed so that the clockwork is precise. They would not cut down a flit to have short idle flits. That, of course, is not a very good stand to take.

7 Allocation and deallocation of resources

There are two virtual channel being talked about in this context. One is the input virtual channel which is deallocated when the tail flit passes switch allocate stage. Switch traversal is a brainless operation and does not need any state. So once the tail flit passes the switch allocate stage the input virtual channel can be deallocated.

Suppose you deallocate output virtual channel only once all credits have been received. Consider an example where there is a two flit packet, a head flit and a tail flit (figure 7). In the example, head flit is allocated a virtual channel at time 2. The tail flit is transmitted at time 4. The credit for the tail flit arrives at time 14. At this time, the virtual channel gets deallocated. So the next packet, which went through route computation at time 2, has to wait for 11 cycles from time 2 to time 14 to get a virtual channel. To keep the physical channel busy we need 11 virtual channels. This is done in order to ensure only one flit present in buffer at any time.

To solve this problem, suppose we deallocate a virtual channel after the tail flit goes through switch allocate stage. In this case, the flits of the next packet have to wait in the next buffer untill the tail flit goes through the switch allocate stage. This may take multiple cycles. We are creating a dependency between packets which may cause deadlock. The only way we make sure that we satisfy Duato's Algorithm is by waiting for the credit flit to arrive before allocating virtual channel for the next packet.

The deadlock may be prevented by other methods which gives us more latitude in handling these idle cycles.

Firstly, a virtual channel can be deallocated as soon as the tail flit passes the switch allocate stage (figure 8). So the virtual channel for the second head flit can be allocated

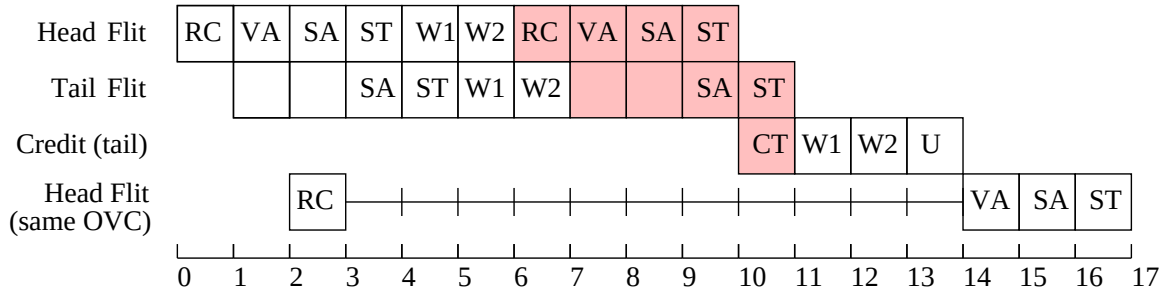

Figure 7: Virtual channel allocation after recieving credit for tail flit

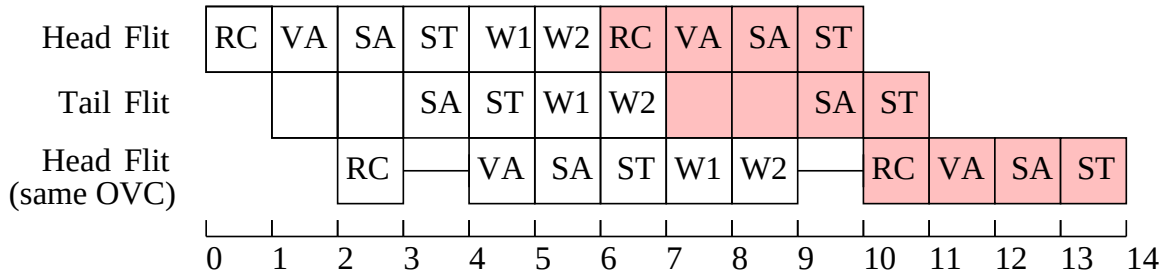

Figure 8: Virtual channel allocation after tail flit completes SA

at time 4 instead of time 14. We do not wait for the credit for the tail flit. This saves us four cycles.

Secondly, there still remains an idle cycle at time 9 for the second packet. Here, the head flit is waiting for the tail flit of the previous packet to go past the switch allocate stage. Head flit needs the global state to store the output of the route computation, which is only released once the other tail flit gets through switch allocate stage. But if we look closely, the head flit only needs to update the R field, while the tail flit only needs to read the O field. So we can allow the head flit to overwrite the R field and save another cycle. There can be a period for which the route in the global state does not correspond to other entries, but that is acceptable. This way the head flit starts route computation at time 9.

There is still a cycle of stall after time 3. That is because the head flit can not reserve the virtual channel until the tail flit passes the SA stage it may remain in SA stage for a while. The only way to get rid of this idle cycle is through speculation.

The sequence of operation is route computation, virtual channel allocation and switch allocation. Unless the route is computed, we do not know which virtual channel is to be used. Hence, we can not proceed to virtual channel allocation stage until the routing is

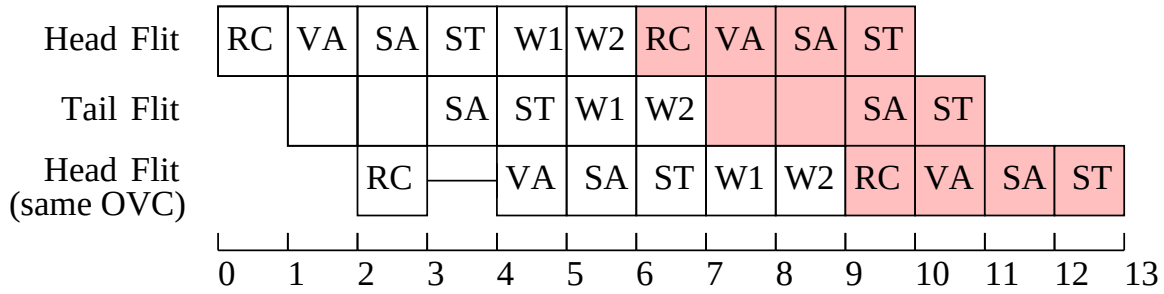

Figure 9: Route computation in parallel with switch allocation

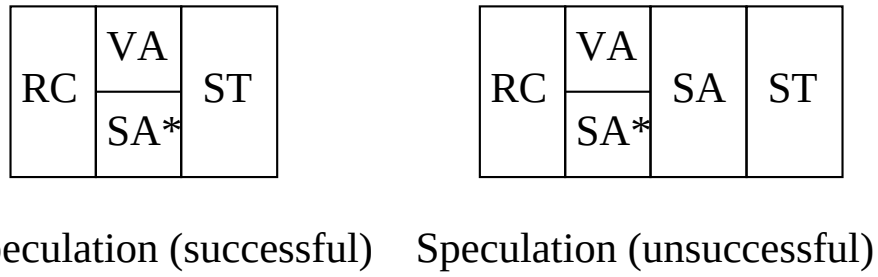

Figure 10: Speculative switch allocation

complete. We could ask for all virtual channels, and then discard the once we do not need, but since there is a real dependency, and there are multiple routes, this is a futile exercise.

Once the route is determined, virtual channels are allocated. It is not necessary to wait for vitual channel before proceeding to switch allocation. If the routing function is such that there is a single output port, we know the output port in advance. If there are multiple ports, all can be speculatively allocated. The complexity however increases. With speculation, the pipeline looks like in figure 10.

The requests for switch allocation are categorized as speculative and non-speculative. Non-speculative requests have priority and thus the speculation does not interfere with normal operation.

This method cuts down latency by one cycle. The stall at time 3 in (figure 9) is removed.

Virtual channel allocation can be split into multiple cycles. In this case, the speculative switch allocation overlaps with last stage of virtual channel allocation. Splitting into multiple stage is not a very practical idea. Suppose virtual channel allocation is split into three stages. At stage 1 and stage two, it does not know whether a previous packet

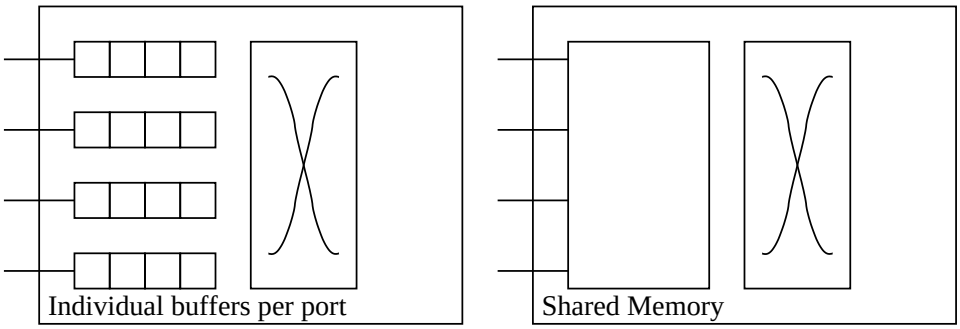

Figure 11: Buffer Architectures

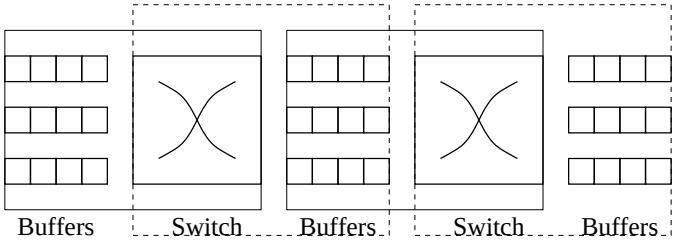

Figure 12: Possible chip boundaries

will be allocated a virtual channel, or any packet will free a virtual channel. So only precomputation can be done at these stages. Therefore it is important to have virtual channel allocation in single cycle.

8 Buffer organization

The buffer architecture can be one of the two types. Either we can have FIFOs for each port or shared memory for all the input controller (figure 11).

Shared memory requires high bandwidth. But we can dynamically allocate buffer space across the ports, providing more buffer space to more active ports. In the case of per port buffers, dynamic allocation can be done among the virtual channels only and not between the ports.

In fact, a whole switch can be built out of shared memory. It can be either input queued or output queued. Fixed bandwidth is provided to all the input ports or output ports in such a case.

Technically the buffers can be put at the output. But this complicates the flow

control. The state of all the outputs of the router at the next hop need to be maintained for virtual channel allocation. Packets basically traverse a chain of buffers and switches. We can have a chip boundary which has an input buffer and a switch, or a switch and an output buffer (figure 12). But from the perspective of flow control, only the former method makes sense.