In-Network Reorder Buffer To Improve Overall NoC Performance While Resolving the In-Order Requirement Problem

Woo-Cheol Kwon, Sungjoo Yoo*, Junhyung Um, Seh-Woong Jeong
Computing Platform, System LSI Division, Semiconductor Business, Samsung Electronics
*Department of Electronic and Electrical Engineering, Pohang University of Science and Technology

ABSTRACT

Data-intensive functions on chip, e.g., codec, 3D graphics, pixel processing, etc. need to make best use of the increased bandwidth of multiple memories enabled by 3D die stacking via accessing multiple memories in parallel. Parallel memory accesses with originally in-order requirements necessitate reorder buffers to avoid deadlock. Reorder buffers are expensive in terms of area and power consumption. In addition, conventional reorder buffers suffer from a problem of low resource utilization. In our work, we present a novel idea, called in-network reorder buffer, to increase the utilization of reorder buffer resource. In our method, we move the reorder buffer resource and related functions from network entry/exit points to network routers. Thus, the in-network reorder buffers can be better utilized in two ways. First, they can be utilized by other packets without in-order requirements while there are no in-order packets. Second, even in-order packets can benefit from in-network reorder buffers by enjoying more shares of reorder buffers than before. Such an increase in reorder buffer utilization enables NoC performance improvement while supporting the original in-order requirements. Experimental results with an industrial strength DTV SoC example show that the presented idea improves the total execution cycle by 16.9%.

1. Introduction

SoCs are requiring more and more on-chip data bandwidth. DTV chips are pursuing UD (ultra definition) video decoding [1], sophisticated pixel processing for video quality enhancement, etc. Mobile SoCs are now supporting HD video coding and 3D graphics [2]. This trend of requiring higher data bandwidth will accelerate as innovative applications are expected to be realized on many-core SoCs [3]. In order to support the ever increasing data bandwidth on chip, the notion of 3D stacked (DDR or SRAM) memory has been introduced [4][5]. 3D stacked memory enabled by TSV (through silicon via) offers multiple independent memory channels thereby higher memory bandwidth.

Data-intensive functions on chip, e.g., codec, 3D graphics, pixel processing, etc. need to make best use of the increased memory bandwidth, possibly, concurrently accessing multiple memory channels (in short, multiple memories throughout this paper). Compared with conventional implementations of those data-intensive functions for one or two memories, we may require a new architectural feature, memory access parallelization, in order to exploit multiple memories. However, there are two practical limitations in obtaining memory access parallelization. First, when existing IPs are reused, they need to be redesigned to use multiple memories concurrently. However, such a redesign may require high design and verification costs due to the restructuring of internal architectures (functions as well as interfaces) to exploit parallel memory accesses. Second, when designing a new IP, the level of memory access parallelism may need to be chosen considering the trade-off between the possible performance improvement and the overhead of silicon resource and design efforts. Memory parallelism may vary from one design (e.g., 2 wide memory ports) to another (e.g., +16 narrow memory ports). However, it will be practically difficult to design a new IP to support all the possibilities of memory parallelism (in a configurable way) or to support only the highest parallelism since it may incur significant design efforts (especially for configurable designs) or an overdesign (when the highest parallelism is targeted).

Thus, there can be a mismatch between the memory access parallelism of IPs and available parallelism in memory. Suppose the case that the memory access parallelism of IP (e.g., two memories are assumed) is smaller than the actual memory parallelism (e.g., eight memories). From the viewpoint of IP, its memory accesses have in-order requirements since it is designed to access only a limited number of memories in parallel. Thus, it has a limited number of independent streams of memory access. In other words, it has in-order requirements on each of independent streams (identified by the transaction ID[1] in the case of AXI bus protocol [6]). When such an IP is used in SoCs with a larger memory parallelism (e.g., eight memories), the increased memory bandwidth may not be fully exploited due to the limited memory access parallelism, i.e., the in-order requirements (See Section 3 for the example of in-order requirement).

Since such in-order requirements prevent designers from fully utilizing the increased memory bandwidth, it is imperative to devise solutions to resolve the problem. Recently, a notion called transaction ID renaming is presented in [7] to address this problem of increasing effective memory utilization in presence of in-order requirements. The key idea is to manage two independent sets of transaction ID on master and network sides, respectively. In order to realize this notion, the authors present an NoC component called request parallelizer which consists of reorder buffer (ROB) and transaction ID management block. The request parallelizer is located at NoC entry and takes in-order requests[2] from master IP(s) and renames transaction IDs between master and network ID sets.

Our observation shows that, although the request parallelizer improves performance by resolving the in-order problem, its

[1] We assign a distinct transaction ID to an independent stream of accesses. Thus, the accesses of the same transaction ID need to be processed in order. The order of memory accesses with the same transaction ID needs to be maintained at the I/O port of the master. However, memory accesses with different transaction IDs can be processed out of order. AXI protocol has 'transaction ID' and OCP has 'tag ID' to represent to which transaction, i.e., to which independent stream of accesses the request/data/response belongs.
[2] We define packet types in Section 4.2.1.

978-3-9810801-5-5/DATE09 © 2009 EDAA

implementation is not area-efficient. Its area (dominated by the ROB) occupies a significant portion, up to 30.1%, of total NoC area [7]. In addition, according to our observation, the utilization of ROB is low. It is used only when in-order packets are in transit.

In our work, in order to improve the utilization of ROB resource previously used only for in-order packets, we propose to move the ROB from the NoC entry (i.e., the request parallelizer) to NoC (i.e., the network router). To do that, we present a novel idea called *in-network reorder buffer* (in short, INROB). We also present a microarchitecture of router to realize the idea. In the presented idea, the resource of the original ROB in the request parallelizer becomes additional virtual channels (VCs) in the network router. The additional VCs are used for normal VC operations (when there is no in-order packet) as well as reorder buffer operations (when there is any in-order packet traversing the router). Assuming that the same amount of resource (that was used by the ROB in the request parallelizer) is now used for the additional VC's, our idea can offer double benefits: (1) improving overall NoC performance (by allowing other packets to use the additional resource and by allowing in-order packets to enjoy more share of ROB) and (2) still supporting the in-order requirements.

This paper is organized as follows. Section 2 reviews related work. Section 3 exemplifies the in-order requirement problem, and presents existing solutions and their limitations. Section 4 presents the idea of in-network reorder buffer, a method of virtual channel management and a router microarchitecture. Section 5 reports experimental results. Section 6 concludes the paper.

2. Related Work

The deadlock problem due to the in-order requirement (to be explained in Section 3) can be resolved by limiting parallelism. In [8], multiple outstanding requests are allowed towards different destinations as far as there is only one outstanding request per destination. In [7], the notion of transaction ID renaming is presented and request parallelizers are placed at network entry or fork points to transform in-order requests into out-of-order ones using the transaction ID renaming and the reorder buffer. However, as Section 3.2 will show, such an implementation of ID renaming and reorder buffer can suffer from a low resource utilization.

Reorder buffers are used in multi-path routing methods to maintain the in-order requirement at the reconvergent points [9][10][11]. Compared with the reorder buffer in existing multi-path routing methods, e.g., [11], the usage of reorder buffer in this paper (and in [7]) is different in that the reorder buffer and ID renaming is applied to the bi-directional transfer of multi-cast memory access requests (from a master to multiple slave DDR memories) and corresponding data (from the slave memories to the master) while the conventional reorder buffer in adaptive routing handles only the uni-directional data transfer (e.g., from a slave DDR memory to a master). However, reorder buffer usage in both [7] and conventional adaptive routing can suffer from a problem of low utilization of reorder buffer to be explained in Section 3.2.

There have been presented several router architectures for NoC purposes [12][13][14]. Among them, the notion of flit reservation flow control in [14] is similar to ours in that flit buffers are reserved. However, the purposes of flit buffer reservation are different since [14] aims at the reduction in the turnaround time in flit control while ours reserves flit buffers to realize the reorder buffer.

3. Background

3.1 In-Order Requirement Problem

Figure 1 illustrates the in-order requirement problem and a solution that resolves it by limiting parallelism. Figure 1 (a) shows a 2x2 NoC connected with two masters, and two DDR memories (via their memory controllers). Figure 1 (b) shows a scenario of memory read accesses from the masters. In the scenario, master 2 sends a request A to memory 1 at time 1. Master 1 sends a request C to memory 2 at time 2 and a request D to memory 1 at time 3, respectively. Master 2 sends a request B to memory 1 at time 4. Assume that each master uses a distinct transaction ID for its memory accesses. Thus, memory accesses are in-order ones for each of masters. For instance, request A needs to be served earlier than request B from the viewpoint of master 2. However, requests from different masters are independent from each other.[3] Thus, memory controller 1 can reorder independent requests A and D (the same for memory controller 2) in order to increase memory utilization. Suppose that memory controller 1 reorders the two requests A and D since request D accesses an already open row in the DDR memory. Thus, the data of request D (in short, data D) is read out earlier than data A as Figure 1 (b) shows. Suppose also that memory 2 serves request B earlier than request C due to the same reason of favoring the open row access.

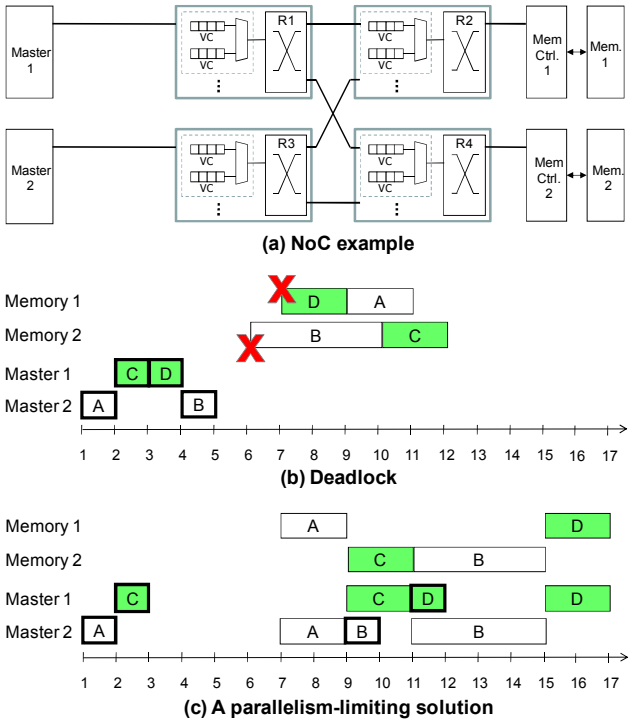

Figure 1 Multiple memory accesses and deadlock problem

We assume zero NoC delay in our examples in Figures 1, 2, and 3. However, our arguments hold in presence of real NoC delay. As shown in Figure 1 (b), at time 6, master 2 cannot receive data B due to the in-order requirement that master 2 needs to receive data A before data B. Thus, master 2 stops data B at its input port while waiting for data A. The same situation occurs at master 1. At time 7, master 1 stops data D at its input port while waiting for data C. However, data A and C cannot advance due to the blocked data D and B, respectively. Thus, a deadlock happens.

Figure 1 (c) shows a solution to resolving the deadlock by limiting parallelism. As shown in the figure, it does not issue requests D and

[3] The requests and data with the same shading in the figure have the same transaction ID.

B until data C and A are received by the masters. By doing that, the deadlock does not occur since each IP can have outstanding requests for only one destination memory. However, performance suffers from limited parallelism.

3.2 Existing Transaction ID Renaming Solution

Figure 2 (a) illustrates a recent solution of transaction ID renaming [7]. It breaks the in-order requirements by managing two sets of transaction ID: master ID and network ID. The transaction ID is renamed by a block called request parallelizer (RP). The RP has an internal reorder buffer (ROB) as Figure 2 (a) shows. When a new inorder read request is issued by a master to the RP, first, the availability of ROB is checked. If there is an available buffer space for the return data of the request, the RP reserves it for the return data of the request[4], issues a new network ID to the request and sends the renamed request to the network. When the data arrive at the RP, they are stored in the previously reserved buffer space. Then, the data in the ROB are transferred to the master according to the original issue order of requests by the master.

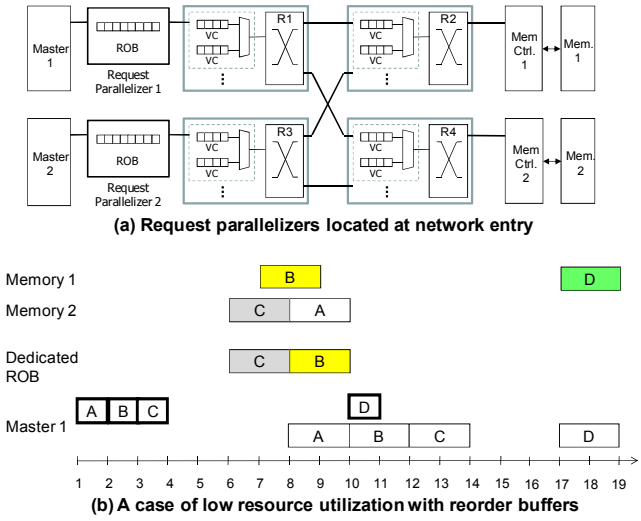

Figure 2 Reorder buffer usage at the entrance of NoC [7]

The transaction ID renaming outperforms the solution of Figure 1 (c) since it allows masters to issue multiple outstanding requests to multiple memories (i.e., multiple outstanding requests per memory), which enables higher memory utilization by memory access reordering. However, it has two limitations. One is a high area overhead of reorder buffer. According to [7], the ROBs in the RPs occupy up to 30.1% of total NoC area. The other limitation is its low utilization of such an expensive ROB resource. Figure 2 (b) illustrates the limitation. In this case, there are two RPs, each for master 1 and 2, respectively. Assume that master 1 tries to make four consecutive read requests A, B, C and D (B and D to memory 1, and A and C to memory 2). Assume also that the ROB in the RP supports up to two outstanding requests for reorder operation. As shown in Figure 2 (b), master 1 can have three outstanding requests.[5] Request D can be issued only after master 1 finishes receiving data A. From the viewpoint of resource utilization, the ROB for master 2 is not utilized in this scenario. In our work, we try to utilize the unused ROB resource of master 2 by moving it to the network router in order to improve the overall performance of NoC as well as that of master 1's requests.

4. In-Network Reorder Buffer Scheme

In this section, we first introduce the basic idea of in-network reorder buffer (INROB). Then, we explain the management of INROB and present the micro-architecture of network router to realize the INROB. In our work, we assume that the control and data network share the same routers in the NoC.

4.1 Basic Idea

Figure 3 illustrates the case that the ROB resources (and transaction ID renaming functions) are moved from the request parallelizers to the network routers as compared with Figure 2 (a). Figure 3 (a) shows that the routers have additional virtual channels (VCs) called *reorder VCs* (RVCs). We show RVCs in the figure only for the illustrative purpose in order to explain that some additional VCs are added to the router (to be exact, as VCs for the data network). In reality, there is no difference between normal VC and RVC since any VC can be used for a reorder or normal purpose by the virtual channel management to be explained in Section 4.2. In the NoC with the INROB, the router needs to maintain the in-order information while performing transaction ID renaming.

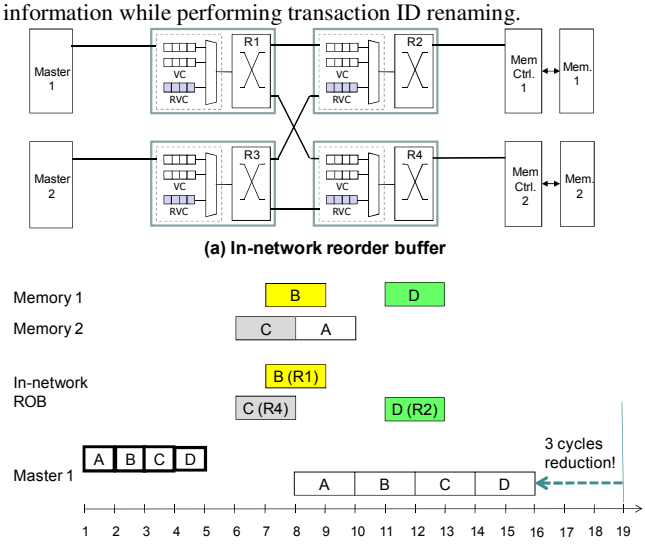

Figure 3 Proposed idea: in-network reorder buffer

Assume the same scenario shown in Figure 2 (b) with the NoC of Figure 3 (a). Each router is assumed to have the additional RVC to support the reservation of (the return data of) one request.[6] Thus, the overall ROB resource is the same as in Figure 2 (a). Master 1 issues four read requests from time 1 to 4. Request A traverses routers R1 and R4 without transaction ID renaming and RVC reservation since it does not have any in-order requirement with previously issued requests. Router R1 is a fork router towards the two memories. Request B is renamed at router R1 and an RVC is reserved for the return data of request B. Request C is renamed at R1. However, request C cannot reserve any RVC at R1 since request B already used up the RVC quota allocated to router R1. Thus, the renamed request C is routed to router R4 and reserves an RVC at the router. Due to the same reason (R1's RVC is occupied by request B), request D is renamed at R1 and reserves an RVC at R2.

[4] Buffer reservation is necessary to avoid the deadlock [7].

[5] Note that the first request A does not reserve the ROB since it does not have any in-order requirement with previously issued requests.

[6] To be exact, the RVC quota is allocated on an input port basis as explained in Section 4.2.

At time 6, memory 2 sends data C to router R4. They are stored in the previously reserved RVC at R4. At time 7, memory 1 sends data B to router R1. They are stored in the RVC at R1. At time 8, memory 2 provides data A. They pass the two routers (R4 and R1) and arrive at master 1. At time 10, R1 detects the end of data A's transfer. R1 starts to transfer data B since data B is now independent. At time 11, memory 1 sends data D to the RVC of R2. At time 12, when the transfer of data B finishes, R1 informs R4 of the completion of data B's transfer by sending a control packet called a *RVC release packet* (in short, release packet). On receiving the release packet, router R4 starts to transfer data C to master 1. At time 14, when the transfer of data C finishes, R1 sends a new RVC release packet to R2 to inform that data D are now independent and ready to be transferred.

Compared with the scenario in Figure 2 (b), the concept of INROB improves by three cycles the total execution cycle of completing the four requests. The improvement in Figure 3 (b) mainly results from the fact that the requests of master 1 can utilize more ROB resources (supporting three requests) in Figure 3 (b) than in Figure 2 (b) (supporting only two requests).

4.2 Reorder Virtual Channel Management

Basically, the router supports normal VC management [15]. In addition to the basic function, we add new functions: transaction ID renaming, RVC reservation and release.

4.2.1 Packet Types

The NoC has two types of packets: control and data packets. The control packet has three sub-types: request, response and release packets. The request packet has two modes, complete and incomplete modes, for the purpose of RVC management. Completeness represents the requirement or status of RVC reservation. All the requests start from the master with the complete mode. Then, when it is assigned a new network ID (at the fork router to be explained in Section 4.2.3), its mode changes to the incomplete mode since RVC reservation is required. After the reservation finishes, the mode changes back to the complete mode. We denote a request packet with the complete (incomplete) mode with a CR (IR) packet.

The request packet has a field called ROB_size which represents the required size of ROB. Initially, it is set to the size of corresponding data packet. When the request finishes RVC reservation, this field becomes zero indicating that the mode becomes complete.

4.2.2 Reorder VC Budgeting

Each input port of router has a quota, i.e., a threshold in RVC usage. Setting the threshold, i.e., *RVC budgeting* plays an important role in the efficiency of INROB. There can be a design space for both design-time and runtime RVC budgeting. We will investigate this in our future work. In the experiments, we used a uniform RVC allocation (one RVC per input port) whose details will be given in Section 5.

RVC budgeting needs a special attention not to cause deadlock in network routing. To be specific, in order to avoid deadlock, RVC budgeting takes the same approach of building a deadlock free NoC with adaptive routing [16]. We assume that a deadlock free network is already designed (e.g., by using distance classes, deadlock free routing such as dimension order routing, etc.). Then, additional resources (e.g., RVCs) are added (with RVC budgeting) on top of the deadlock free network.

4.2.3 Transaction ID Renaming

The transaction ID renaming (between master and network IDs) of the original request parallelizer is performed by the router (called fork router) where request packets with the same master ID bifurcate towards different destinations. Note that the router renames request packets with the same transaction ID only when their destinations are different. In order to detect the difference in the destination, the router keeps the information of destination (i.e., outport) per active transaction ID.[7] Thus, the router monitors the transaction ID of incoming packet and checks to see if it matches any of active transaction IDs. If so, the destination check is performed. If the bifurcation in destinations is detected, then the router performs transaction ID renaming and changes the mode of incoming request into the incomplete mode since it now requires RVC reservation.

Renaming is implemented as assigning an increasing counter value to the NoC sequence field of request packet. The transaction ID of data/response/release packet follows that of the corresponding request packet. The fork router also issues release packets in a look-ahead manner as explained in Section 4.2.5.

4.2.4 Reorder VC Reservation

Figure 4 shows the pseudo code of RVC reservation. For each incoming incomplete request (IR) packet, the RVC availability (within the RVC budget) is checked for the port of return data path (line 2). If there is availability, the router tries to reserve the VC up to the ROB_size of the IR packet within the RVC budget (line 3). If the reservation is incomplete, i.e., only a part of required buffer space (no buffer space) is reserved, the ROB_size field of the packet is updated (unchanged) (lines 4-5). Then, the request packet is routed towards its destination (line 6). The IR packet can arrive at the last router connected to the destination without completing the reservation. In such a case, the request is kept at the last router as a pending request occupying a VC in the control network. We set a threshold of maximum number of pending IR packets that a router can support. Thus, if the threshold is reached, the router blocks incoming IR packets until the pending ones are resolved. The pending request will be resolved later when a release packet arrives at the router as explained in Section 4.2.5.

```
1  For each IR packet
2    If VC is available at the port of return data path
3      Reserve the VC as RVC at the port
4      If the reservation is incomplete
5        Update the ROB_size field in the packet
6  Forward the request packet towards the destination
```

Figure 4 Reorder VC reservation

4.2.5 Reorder VC release

Reorder VC release is managed by a release packet sent by the fork router that previously renamed the corresponding request packet. The router sends the release packet for the next request in the original request packet order. For instance, in the case of Figure 3 (b), at time 12, router R1 sends a release packet to R4 to inform that data C becomes ready to be transferred since the previous data B completes the data transfer. The release packet contains the transaction ID of the corresponding packet (for which the ROB setup was tried). The release packet has a higher priority than the other packet types, request, response and data packets.

[7] When there is any outstanding request, the transaction ID of the outstanding request is called *active transaction ID.*

```
1  If there is any RVC reserved for the corresponding packet
2    Change the status of RVC to normal VC
3  If the corresponding packet is pending at the router
4    Change the pending IR packet to a CR packet
5    Forward the CR packet towards the destination
```

Figure 5 Router behavior on receiving a release packet

Figure 5 shows the pseudo code of handling the release packet. Upon receiving a release packet (for instance, at R4, time 12 in Figure 3 (b)), the router checks to see if there is any RVC that the release packet targets (line 1 in Figure 5). If so, it changes the status of RVC into 'Normal VC' (line 2) since the corresponding request (request C in the example) is now independent and does not require RVC reservation. Then, the contents of RVC (now, normal VC) are sent to the corresponding master. If the corresponding IR packet resides at the router as a pending request (line 3) due to a partial RVC reservation or a flow control, the router changes the status of IR request from 'incomplete' to 'complete' since the request is now independent (line 4). The CR (complete request) packet is forwarded towards its destination (line 5).

The delay from the issue of release packet to the arrival of next data at the fork node can affect the performance of INROB scheme. In order to reduce the effect, the fork router performs a look-ahead issue of release packet. When detecting the first flit of currently ready data packet, the fork router issues a release packet to inform the next data packet. As shown in Section 5, the look-ahead release packet improves the performance of INROB scheme significantly.

4.3 Router Architecture

Figure 6 shows the structure of router supporting the concept of INROB. It consists of three pipe stages: Route Compute/VC allocation/RVC management – VC selection (for the input of crossbar) – Outport arbitration (for the output channel of router). The RVC management includes RVC allocation to an IR packet (if there is available buffer space for RVC), status update from a pending IR packet to a CR packet, VC allocation to an IR packet (if it is blocked by the next router), and sending a release packet (if the router is the fork one).

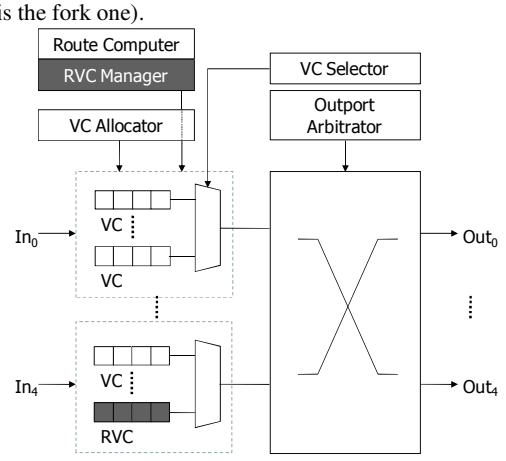

Figure 6 Router architecture

The look-ahead release packet makes multiple data streams (corresponding to consecutive request packets) advance towards the master. When the router performs port arbitration (pipe stages 2 and 3 for the inport and outport arbitration of crossbar), the router utilizes the original in-order information (i.e., network ID which is the sequence number in the packet header) to select the arbitration winner as in [7].

5. Experimental Results

In our experiments, we use two test cases: synthetic ones and industrial strength DTV SoC example.

5.1 Network-on-Chip

Figure 7 illustrates the entire architecture consisting of a 16 node mesh NoC, eight processing elements (PEs, denoted with 'P' in the figure) and eight memory sub-systems (denoted with 'M') each consisting of a memory controller and a 32b DDR memory (CL-tRP-tRCD=3-3-3, 4 banks/memory). We use a commercial memory controller (PL340) [6]. The memory controller has the data FIFO of size 16 (64b words).

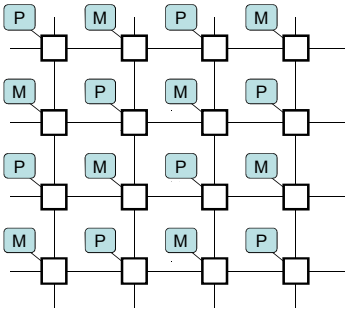

Figure 7 A 16 node mesh NoC

The NoC uses 64b flits. The control packet has 1 flit, and the data packet 8 flits. The router performs X-Y routing and wormhole switching. For the data network, the baseline router has a 4-flit deep buffer per VC and 2 VCs per input port. For the control network, the router has four VCs (each 1-flit deep buffer) per input port.

5.2 Synthetic Test Cases

In the synthetic test cases, four PEs (among total eight) accesses eight memories in three scenarios: random, local, and mixed traffics. In the random traffic, the four PEs send in-order read requests to eight memories with randomly generated addresses. Thus, the memories are selected randomly for each read request. In the local traffic, the four PEs send random in-order read requests to the near memories more than other remote memories. We used the weighting factor proportional to $1/2^d$ where d is the distance between the PE and the memory according to [17]. In the mixed mode, one PE sends in-order read requests to eight memories randomly while each of the other three PEs sends, to its own designated memory, read requests that do not require RVC reservation. In the three scenarios, each PE sends burst 8 read requests to the memories depending on the given request rates (e.g., given 20%, a burst 8 request is sent every 40 cycles).

When the method in [7] is applied, the ROB size of request parallelizer per PE is 4*8*64b (supporting five outstanding in-order requests of 64b burst 8), which corresponds to eight (4-flit deep) VCs in the data network. In the NoC with INROB, we distribute the ROB resource of request parallelizers (equivalent to total 64 VCs) to 16 routers. Thus, each router has four additional VCs (i.e., one additional VC per input port) as the RVCs. The RVC budget is set to one VC per input port. The threshold of the number of maximum pending IR packets is set to three.[8] For each input port, the router keeps up to eight pairs of information of active transaction ID and destination for the destination check.

We run RTL simulation with the NoC and PE models (written in Specman), the RTL code of PL340, and the Denali DDR memory

[8] We leave, for the release packet, one VC (among four) at the input port of router in the control network.

model. The simulation runs until all the PEs complete the same amount of memory accesses.

Figure 8 shows the simulation results. We measure latency from the time when a packet of memory access request is issued from the PE to the time when the data are completely received by the PE from the memory. As Figure 8 shows, compared with the existing method (RP) in [7], the presented methods (INROB_L and INROB) decrease average read latency as the request rate increases. For instance, in the case (shown with arrows in the figure) that the request rate is 90% for the random traffic, the presented method with the look-ahead release packet (INROB_L) gives 27.3% reduction in average read latency.

The benefit of the presented method becomes much clearer in the mixed traffic scenario as Figure 8 (c) shows. The presented method (INROB_L) gives 32.8% reduction in the latency in the case of request rate of 90%. The main reason of such a performance improvement is that the packets that do not require RVC reservation benefit from the increased resource, i.e., four VCs per router.

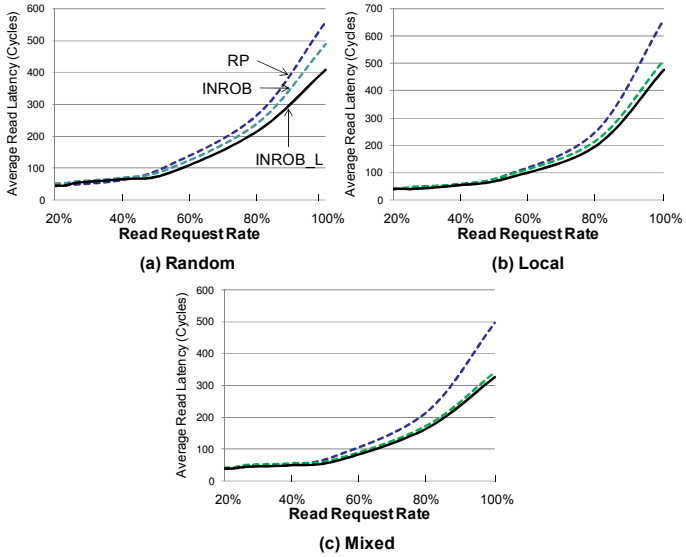

Figure 8 Performance evaluation under synthetic test cases

5.3 DTV SoC Case

We use an industrial strength DTV SoC case used in [7]. It supports MPEG2 decoding for QFHD size (3840x2160) video processing [1]. We apply the presented approach to the backbone NoC, 4x4 mesh as shown in Figure 7. The architecture has, as masters, four video codec IPs and four different pixel processing IPs, e.g., noise reduction, mixer, etc. The DTV example uses four DDR memories (located at the central two columns in Figure 7). Each of the four codec IPs accesses only its own dedicated DDR memory. Thus, the memory accesses of codec IPs do not require RVC reservation. Each of the other four IPs has two master transaction IDs and accesses all the four memories. We use the same configuration of NoC and memory controller as in the synthetic test cases.

Table 1 Performance comparison with the DTV case

	Avg. Latency	Normalized Exec. Cycle
RP [7]	138	1
INROB	131	0.907
INROB_L	119	0.831

Table 1 shows the performance comparison. The presented method gives 16.9% reduction in total execution cycle compared with the method in [7]. The main reason is analyzed as follows. The communication periods of the eight IPs do not always overlap with each other. Thus, in our method, there are possibilities that the codec IPs as well as the pixel IPs benefit from the increased VC resource in the router. In this case, the look-ahead release packet gives further improvement (7.6%) in the total execution cycle.

6. Conclusion

This paper presented a novel idea of in-network reorder buffer that improves the utilization of reorder buffers by embedding them into the network routers. Compared with the existing method, the presented idea allows both in-order and other packets to utilize the increased resource (reorder virtual channel, RVC) in the routers while supporting the in-order requirements. We presented the method of RVC management that performs the reservation and release of RVCs. We also presented the micro-architecture of router that supports RVC management. Experimental results show that the presented idea improves overall system performance by 16.9% in the case of DTV SoC example. In our future work, we will work on the architectural exploration of RVC budgeting, IR packet threshold assignment, QoS guarantee for in-order requests, etc.

7. References

[1] Y. Lin, "Design Challenge of a QuadHDTV Video Decoder", MPSoC School, 2007, available at http://tima.imag.fr/mpsoc.

[2] nVidia Tegra, http://www.nvidia.com/page/handheld.html.

[3] Intel Tera-scale Computing Research Program, http://techresearch.intel.com/articles/Tera-Scale/1421.htm.

[4] S. Borkar, "Thousand-Core Chips - A Technology Perspective", Proc. DAC, 2007.

[5] T. Ezaki, *et al.*, "A 160Gb/s Interface Design Configuration for Multichip LSI", Proc. ISSCC, 2004.

[6] AMBA3 protocol and components, http://www.arm.com/products/solutions/AMBAHomePage.html.

[7] W. Kwon, *et al.*, "A Practical Approach of Memory Access Parallelization to Exploit Multiple Off-chip DDR Memories", Proc. DAC, 2008.

[8] A. Harris, *et al.*, "Bus deadlock avoidance", US Patent 7219178, 2007.

[9] J. Hu and R. Marculescu, "DyAD – Smart Routing for Networks-on-Chip", Proc. DAC, 2004.

[10] J. Kim, *et al.*, "A Low Latency router Supporting Adaptivity for On-Chip Interconnects", Proc. DAC, 2005.

[11] S. Murali, *et al.*, "A Multi-Path Routing Strategy with Guaranteed In-Order Packet Delivery and Fault-Tolerance for Networks on Chip", Proc. DAC, 2006.

[12] E. Rijpkema, *et al.*, "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip", Proc. DATE, 2003.

[13] T. Marescaux and H. Corporaal, "Introducing the SuperGT Network-on-Chip", Proc. DAC, 2007.

[14] L. Peh and W. J. Dally, "Flit-Reservation Flow Control", Proc. HPCA, 2000.

[15] W. J. Dally, "Virtual Channel Data Flow Control", IEEE Trans. Parallel Distributed Systems, 3(2), March 1992.

[16] W. J. Dally and B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann Publishers, 2004.

[17] NoC-Benchmark Working Group, "Network-on-Chip Micro-Benchmark Specification", OCP-IP, May 2008.