An Open-Loop Flow Control Scheme Based on the Accurate Global Information of On-Chip Communication

Woo-Cheol Kwon, Sung-Min Hong, Sungjoo Yoo, Byeong Min, Kyu-Myung Choi, Soo-Kwan Eo
CAE Team, System LSI Division, Semiconductor Business, Samsung Electronics

ABSTRACT

3D stacked memory is being adopted as a promising solution to offer high bandwidth and low latency in memory access. Compared with the on-chip network design with conventional off-chip memory, it gives a new problem of minimizing communication conflicts since multiple concurrent high bandwidth data transfers will flow through the on-chip network. In order to tackle this problem, we propose applying an open-loop flow control scheme based on the accurate global information (destination and status) of on-chip communication. The proposed open-loop flow control scheme exploits the information and selectively buffers and arbitrates data transfers to remove conflicts at destinations in a preventive manner. As an implementation of the presented scheme, we present on-chip buffers called Buf3D's that share the global information with each other to perform the selective buffering and arbitration of data transfers. Experiments with synthetic test cases and an industrial strength DTV design show that the proposed method improves aggregate memory bandwidth significantly (average 19.0%~25.8% in the synthetic cases and up to 18.4% in the DTV case) with a small area overhead (15.2% in the DTV case) of on-chip network.

1. Introduction

3D stacked memory is gaining more and more attention as one of promising system-in-package architectures [1][2][3]. The main benefits of 3D stacked memory are twofold: increased memory bandwidth and lower latency. It gives higher memory bandwidth with more memory ports (tens of 64b ports per memory die can be achieved via vertical die to die interconnect, e.g. face-to-face or through silicon via [2]) than conventional DDR memory usage (one or two ports per memory die). It gives also lower memory access latency since three dimensional interconnect can give shorter wire length than two dimensional one [2]. Thus, 3D stacked memory architectures are expected to be applied to many-core designs [4][5][6] where the requirements of memory bandwidth and latency will be much higher than current SoC designs.

The problem of simultaneously accessing multiple slaves (e.g. small SRAM's or multiple peripheral devices) may not be new in on-chip communication. In reality, it was not a performance critical issue. However, the performance to access multiple off-chip DDR memories is an important performance problem since it will determine the entire system performance.

DDR memory is usually used for heavy (high bandwidth and large volume) data flows. With 3D stacked memory, masters can exploit the increased bandwidth and reduced latency by accessing multiple memory modules simultaneously. Thus, there can be significant amount of concurrent data transfers between a few heavily communicating masters and memory modules. In such a case, there will be more possibility of conflicts at those masters than when conventional (one or two port) DDR memory is used.

The conflicts are mostly caused by concurrent data transfers going to the same destination. For instance, a master may generate concurrent data transfers (for read data) from multiple memory modules to the master by issuing consecutively multiple memory read requests to different memory modules. In such a case, the concurrent data transfers (carrying the requested data) from different memory modules may conflict with each other at the master interface port.

Conflict (in other words, congestion) resolution requires flow control. In general communication networks, closed-loop flow control schemes are usually applied. Congestion (conflict) info is propagated from the congested destination (or congested link) to the sources via the feedback loop such as back-pressure through buffer overflow, delayed permit, increased retransmission, and/or choke packet [12][13]. However, these methods suffer from unnecessary occupation of network resource (e.g. buffer) and memory modules (whose new data transfer is blocked by the previously blocked data transfer) or large and unpredictable delay in the feedback loop. Section 3 gives the example of this problem in the context of 3D stacked memory usage.

Compared with conventional communication networks, one salient characteristic of on-chip network is that the accurate global information, i.e. which data transfer flows currently from which source to which destination, can be gathered and exploited to better control conflicts in communication. In this paper, we propose applying an open-loop flow control scheme based on the accurate global information of on-chip communication at the communication sources, not based on the feedback information. Based on this information, the proposed scheme selectively buffers and arbitrates data transfers, in front of sources, to avoid conflicts at destinations in a preventive manner. For the implementation of the scheme, we present an on-chip network architecture with a global information bus and buffers dedicated to the open-loop flow control scheme.

This paper is organized as follows. Section 2 reviews related work. Section 3 gives an example of communication conflict with multiple memory modules. Section 4 presents the on-chip network architecture with the proposed flow control scheme. Section 5 reports experimental results. Section 6 concludes the paper.

2. Related Work

In order to fully utilize both the increased bandwidth and reduced latency of 3D stacked memory, on-chip cache and on-chip network architectures need to be optimized for multiple memory modules. There have been several studies on cache and network-on-chip (NoC) architecture for this purpose [2][7]. They tackle the design of cache and NoC by exploring cache design space (e.g., shared or private L2, inclusion or not, etc.) or by devising an NoC router architecture for communication among vertically stacked dies.

Closed-loop flow control is applied to NoC in several studies based on buffer usage (e.g. credit-based [14], router-to-router handshake-based [9], etc.) or in a bufferless manner [15]. In [11], a method of predictive flow control is presented. It is based on the router and traffic source models that allow for the prediction of congestion in the network.

In terms of exploiting the global information of communication, a concept of control NoC is presented in [16]. It transfers the information of communication statistics (e.g., # blockings) from destinations to sources. There are two main differences between [16] and ours: the source of global information and flow control scheme (open or closed-loop). [16] gathers an indirect information (e.g. congestion status) at destinations and exploits it in controlling packet injection rate at sources while ours gathers a more direct and detailed information (e.g. which transfers (can) go to which destinations) and exploits it to arbitrate (and buffer) data transfers at the source side.

Compared with the existing closed-loop flow control schemes for on-chip networks, the benefit of the proposed open-loop scheme is that it allows for a more precise control of conflicts at the destinations based on the arbitration and buffering, at the source side, using the detailed global information of data transfers.

3. Motivational Example

Figure 1 illustrates an example of on-chip network with two memory modules. In the figure, the on-chip network is exemplified with a 2x2 mesh NoC (rectangles annotated with 'R' represent routers). We assume that masters and memory controllers have AXI interface [10]. Thus, we use network interfaces (NI's) to connect them to the NoC.

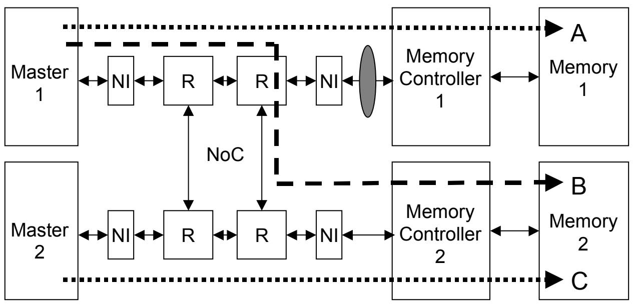

Figure 1 An on-chip network with multiple memory modules

Assume that master 1 tries to exploit the increased bandwidth and (possibly) reduced latency of multiple memories by making two independent read requests A and B consecutively to two memory modules as denoted with dotted and dashed arrows in Figure 1[1]. Master 2 makes a read request C to memory 2 just after request B arrives at the memory controller 2. We assume zero-latency memory for the simplicity of explanation while this example and our approach hold for real delay cases as in our experiments.

Figure 2 (a) shows an operation scenario of the memory accesses. At time t_1, we assume that two requests A and B arrive at memory 1 and 2, respectively. Since both data transfers of requests A and B are ready, they start at time t_1. However, they have a conflict with each other at their destination, master 1. Assume that request A's data are transferred to master 1 before request B's data[2]. Thus, request B's data are blocked[3]. The backpressure of blocking is propagated along the path in the NoC to memory 2. The blocked data transfer may continue to occupy the NoC resource (e.g. virtual channel buffers), and finally to block memory 2 during the blocked period. The figure illustrates the (partially) blocked status of NoC and memory 2 with shaded rectangles.

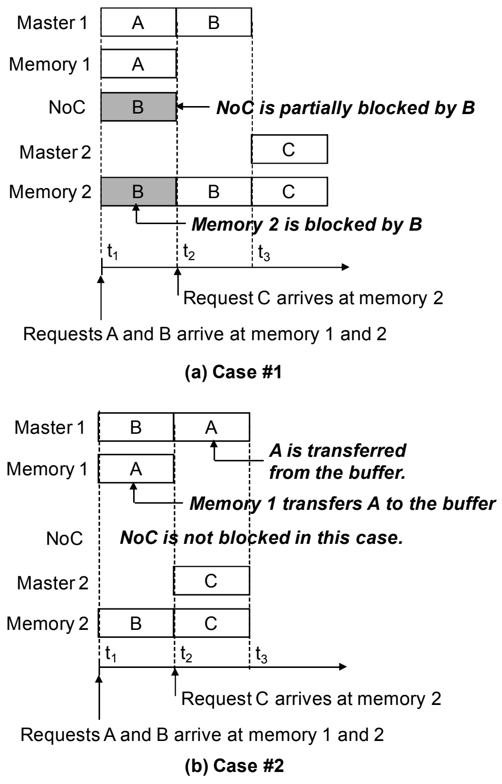
Master 1
A
B
Memory 1
A
NoC
B
NoC is partially blocked by B
Master 2
C
Memory 2
B
B
C
Memory 2 is blocked by B
t1
t2
t3
Request C arrives at memory 2
Requests A and B arrive at memory 1 and 2
(a) Case #1
Master 1
B
A
A is transferred from the buffer.
Memory 1
A
Memory 1 transfers A to the buffer
NoC
NoC is not blocked in this case.
Master 2
C
Memory 2
B
C
t1
t2
t3
Request C arrives at memory 2
Requests A and B arrive at memory 1 and 2
(b) Case #2

Figure 2 Two cases of data transfers

The blocking is resolved at time t_2 when the data transfer of request A finishes, and that of request B starts. Request C's data transfer starts only after request B's finishes at time t_3. Thus, in this case, the three data transfers run sequentially though there is enough resource (two memory modules and multiple paths on NoC) for concurrent data transfers.

The sequential execution results from the conflict between A's and B's data transfers at master 1. Flow control based on buffering may resolve the conflict. Assume that a buffer is inserted between

[1] We assume that the master can make multiple outstanding requests that most of advanced on-chip network protocols, e.g. AXI and OCP support. Note also that there is a critical design issue on how many bus ports to use for accessing multiple memories. It is beyond the scope of this paper. We assume that the number of bus ports is fixed and that a single port can access multiple memories.

[2] The optimal decision of which data transfer to allow first is a fundamental problem in this case. Our solution presented in this paper tackles this problem in the viewpoint of congestion control. A complete solution will require further investigations.

[3] Even in the case that virtual channels [8] are used in the link between master 1's NI and the router and that both data transfers of request A and B share the link via virtual channels, we will still have the problem of occupying NoC resource (by each of the two data transfers at a half level) and, finally, possible blocking at memory 1 and 2.

memory controller 1 and the NI as illustrated with a shaded oval in Figure 1. When using the buffer, one critical runtime decision is whether to store data in the buffer or to bypass the buffer without storing data. A simple closed-loop flow control scheme will be that, after the data transfer is blocked at the destination or somewhere on the NoC and the feedback of blocking information is received, the remaining data are stored in the buffer. If this scheme is applied, i.e. if request A's data are not buffered but bypass the buffer, we will obtain the same scenario as the one in Figure 2 (a). Thus, we need a sophisticated scheme to make a better buffering decision to resolve the conflict.

Figure 2 (b) exemplifies the flow control scheme proposed in this paper. At time t_1, when both data transfers of requests A and B are ready, request A's data do not enter the NoC, but are stored in the buffer. The decision of buffering request A's data can be made based on the global information that there are two concurrent data transfers (of requests A and B) bound for the same destination (master 1). The rationale of buffering request A's data is that *since both data transfers are bound for the same destination and finally there will be an inevitable conflict, it will be better to buffer one of the two transfers and to allow the buffered one to be transferred later when there is no possibility of conflict.* This scheme is 'open-loop' one since it does not depend on the feedback information on conflicts on destination or NoC. Instead, based on the global information available at sources, it predicts precisely whether conflicts will occur at destinations or not and makes the decision of buffering and arbitration to avoid the conflicts in a preventive manner[4].

As shown in Figure 2 (b), at time t_2, the buffered data of request A start to be transferred to master 1. At the same time, request C's data are transferred from memory 2 to master 2. Note that at time t_2, memory 1 becomes free after finishing the transfer of request A's data to the buffer.

As shown in Figure 2 (b), the proposed scheme gives two benefits. First, the total runtime decreases since both the data transfers of requests A and C run concurrently in this scheme. Second, there is no blocking on NoC and memory modules. Comparing the two cases in Figure 2, the proposed flow control scheme can give a better result than a simple closed-loop flow control scheme gives.

4. Proposed Open-Loop Flow Control Scheme

In this section, we first introduce an on-chip network architecture with the open-loop flow control capability. Then, we explain a two-step arbitration to avoid conflicts based on the global information and the key component of the proposed scheme, the buffer called Buf3D (buffer for 3D stacked memory).

4.1 On-chip Network Architecture with Open-Loop Flow Control Capability

Figure 3 shows the on-chip network architecture. The figure shows only the part of read data network in the on-chip network since there is no change in the other parts of on-chip network for request and write data processing compared with the conventional on-chip network. The read data network consists of the read data network of conventional on-chip network[5] (shown as On-chip Network in Figure 3) and two new parts: global information (GI) bus and buffers for 3D stacked memory (Buf3D's).

The buffers (Buf3D's) perform the open-loop flow control. Thus, they need to share the global information with each other and make the buffering decisions based on the information. In Figure 3, the GI bus carries the global information. Each Buf3D provides the GI bus with the information of its currently ready data transfer. Each Buf3D also receives via the GI bus the information of currently ready data transfers of the other Buf3D's.

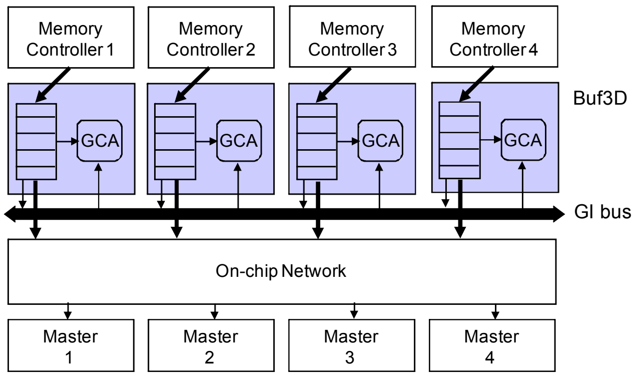

Figure 3 On-chip network architecture with the open-loop flow control

Functionally, the Buf3D can handle multiple independent data flows. We define a *data flow* to be an ordered set of data transfers (identified by transaction id in the case of AXI protocol). The Buf3D supports a specified number (called reorder depth N_{RD}) of those data flows. The Buf3D has one fifo per one data flow. Thus, dynamic assignment of fifos to data flows is performed in the Buf3D.

When a data transfer arrives at the Buf3D from the memory controller, if there is any fifo assigned to the data flow of the data transfer and if the fifo is not full, the data is forwarded to the fifo. If there is no fifo assigned to the data flow and if there is an empty fifo, the fifo is assigned to the data flow of the data transfer.

4.2 Two-Step Arbitration to Avoid Conflicts

The arbitration of data transfers among Buf3D's is the critical function in the presented open-loop scheme. It consists of two steps: local and global arbitration. Figure 4 illustrates the two-step arbitration.

In the local arbitration, each Buf3D selects a local winner among multiple ready data transfers from its fifos. The global arbitration is performed per destination. It selects a global winner among the local winners of Buf3D's that are bound for the same destination. For both local and global arbitration, we can apply existing arbitration schemes, e.g., (weighted) round robin, fixed priority, mix of round robin and fixed priority, min/max latency-based, etc. The

[4] To be exact, the proposed method performs an 'open loop' control from the viewpoint of entire on-chip network. However, there are closed loops among sources via the GI bus.

[5] For instance, in the case of AXI-based on-chip network, the read data network corresponds to the network connecting only the read data channels among the five AXI channels (read/write address, read/write data, and response channels).

arbitration scheme(s) applies to the fifos in the local arbitration and to the Buf3D's in the global arbitration.

In Figure 4, we assume that the N_{RD} of Buf3D is four. We denote ready data transfers with shaded rectangles. Each ready data transfer also shows its destination id. For instance, Buf3D #1 has three ready data transfers: one bound for master #4, another master #2, and the other master #3[6].

As shown in Figure 4, the two-step arbitration is applied in a number of rounds (at maximum, min(# masters, # Buf3D's) rounds). Suppose that the round robin scheme is applied to the local arbitration of all Buf3D's[7]. Figure 4 exemplifies that Buf3D #1 selects data transfer #2 as the local winner, Buf3D #2 data transfer #2, and Buf3D #3 data transfer #3 in the local arbitration of the first round.

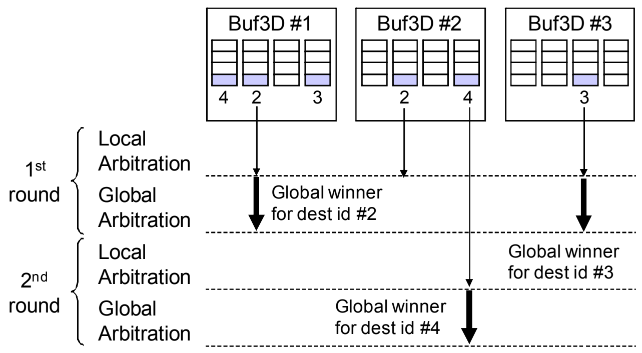

Figure 4 An example of two-step arbitration

In our example of Figure 4, in the global arbitration of the first round, two data transfers with destination id #2 (one from Buf3D #1 and the other from Buf3D #2) have a conflict. To be exact, if they are allowed to start, they will finally have a conflict at their destination, master #2. Assume that the priority-based scheme is applied to the global arbitration and that the priorities are assigned as Buf3D #1 > Buf3D #2 > Buf3D #3 (Buf3D #1 has the highest priority). Since Buf3D #1's priority is higher than Buf3D #2's, Buf3D #1's transfer becomes the global winner for destination #2. In the case of Buf3D #3, since there is no conflict with its data transfer bound for destination id #3, its data transfer becomes the global winner for destination #3.

In the second round, the Buf3D's with global winners do not participate. Instead, only the Buf3D's that lost the first round participate with a new local winner. In the example, as shown in the figure, Buf3D #2 selects data transfer bound for destination id #4 as the new local winner in the second round. Since there is no conflict with other transfers, the data transfer becomes the global winner for destination #4.

Figure 5 shows the pseudo code of two-step arbitration in Buf3D. Note that the basic assumption of two-step arbitration is that each Buf3D has the same global arbitration scheme per destination. Note also that the arbitration makes the decision of 'buffer the data' (when the data is not the winner) or 'bypass the buffer' (when the data just arrives at the Buf3D, if it becomes the winner in the two-step arbitration, it is sent to the destination without being stored at the buffer) mentioned in Section 3.

Note that the GI bus interconnects Buf3D's that can be distributed over the entire SoC. Thus, the communication over the GI bus can have multi-cycle delay. In such a case, each Buf3D will perform the two-step arbitration based on the cycle(s) old information of the ready data transfers of the other Buf3D's, which may degrade the performance of the proposed method. In our experiments, we investigate the effect of GI bus delay.

```
1 TwoStepArbitration() {
2   while ( lw = LocalARB() ) {
3     gw = GlobalARB(lw);
4     if ( gw == lw ) { mark lw as the winner transfer; break; }
5     else goto line #2;
6   }
7   if ( there is a winner ) StartWinnerTransfer();
8 }
```

Figure 5 Pseudo code of two-step arbitration

4.3 Buffer for 3D Stacked Memory

Figure 6 shows the internal structure of Buf3D (with AXI interface protocol[8]). It consists of a shared data buffer (and its related logic) and a block for global conflict avoidance (GCA). Buf3D receives data from the memory controller while sending data to the on-chip network in a pipelined manner. Buf3D also provides the GI bus with its information of ready data transfer, i.e. destination id and the status of its local winner. In the case of AXI-based Buf3D, destination id corresponds to *master id*[9] and the status of data transfer to signal *rvalid* in the read data channel of AXI interface. They represent whether the data transfer bound for the master with *master id* is currently ready or not (if *rvalid* is high, the data are ready to be transferred).

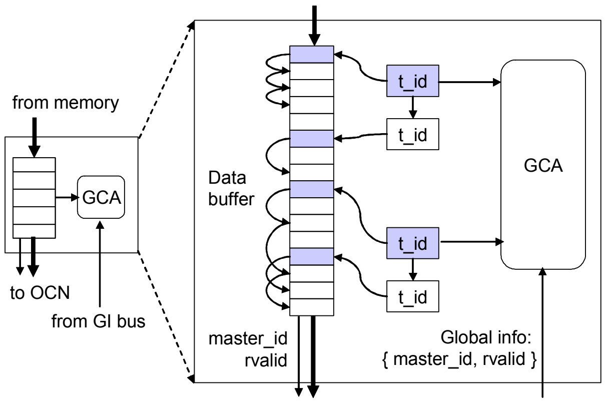

Figure 6 Internal structure of Buf3D with AXI protocol

In order to implement the fifos assigned to data flows, the shared data buffer has its own control logic that consists of transfer id

6 In this section, we use two terms, destination and master, interchangeably.

7 Note that each Buf3D can have different local arbitration schemes. For simplicity of explanation, we assume the same policy for all the Buf3D's in this example.

8 Other interface protocols such as OCP can also be applied to the Buf3D design. In this section, we present the AXI-based Buf3D to give a more concrete example.

9 Master id is a part of transaction id of AXI interface, called port id, in the case of ARM crossbar component, PL301 [10].

nodes and linked lists. In the figure, transfer id nodes are denoted with rectangles. Each independent data flow is managed with a separate linked list of transfer id nodes. The figure illustrates two linked lists of transfer id nodes. The head transfer id nodes are denoted with shaded rectangles. Only the data transfers at the heads of the lists are ready ones. Each transfer id node has an associated linked list of data elements in the data buffer as illustrated in the figure.

The GCA block performs the two-step arbitration. To do that, it uses both the local and global information. The local information is the list of currently ready data transfers stored in the data buffer. As illustrated with two arrows going to the GCA block from transfer id nodes in Figure 6, the GCA block reads the local information of head transfer id nodes. It also reads the global information that is a set of signals from the GI bus (in this example, the set consists of *master id*'s and *rvalid*'s in the data channels of AXI interface of the other Buf3D's).

5. Experimental Result

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

5.1 Synthetic Test Cases

The synthetic test cases are similar to the one in Figure 3. They consist of M 64b AXI masters and N 32b DDR memory modules (tCL-tRP-tRCD=3-3-3, 4 banks/memory). We change the number of masters, M (2 ~ 8) and that of memory modules, N (2 ~ 8) in the experiments. We use commercial AXI crossbar (PL301) and memory controller (PL340) [10]. PL340 has the data fifo with size 16 (64b words). The GI bus consists of N x ($\log_2 M$ + 1) signals[10]. Thus, for instance, in the case of 8 memory modules and 8 masters, we have a 32b GI bus ($32 = 8*(\log_2 8+1)$).

Given N memory modules, each master accesses all the memory modules simultaneously by continuously generating memory read requests (with random memory addresses and burst size 8). We run RTL simulation. In the simulation, each master reads the same total amount of data from multiple memory modules. We set the local and global arbitration schemes to round robin and priority-based one, respectively. We set N_{RD} to four and the size of shared buffer to 128 bytes for each of Buf3D's.

Figure 7 shows the comparison of normalized performance (total runtime[11] to complete the entire simulation) among two designs with conventional closed-loop scheme (CL and CL+Buffer) and four designs with Buf3D. The X-axis represents the configurations of masters and memory modules (e.g. 4x8 represents the case of 4 masters and 8 memory modules). We implement the case with the closed-loop scheme and buffer (CL+Buffer) by using Buf3D's only as buffers. The four designs with Buf3D's are experimented to show the effect of GI bus delay. We experiment the GI bus delay from zero (the case, Buf3D in the figure, with only combinational delay among Buf3D's) to three cycles (Buf3D 3L).

As shown in Figure 7, the open-loop flow control scheme gives significant performance improvement. It gives 9.2%~36.6% (average 25.8%) performance gain w.r.t. the closed-loop scheme in the case (Buf3D) that only the combinational delay exists in the GI bus. As the GI bus delay increases, the gain decreases slightly. Even in the case of significant GI bus delay of three cycles (Buf3D 3L), the gain is still 7.5%~25.3% (average 19.0%). These results show that the proposed scheme can be effective in removing the communication conflicts thereby improving the memory performance.

Figure 7 also shows the case of applying only the buffering to the closed-loop scheme (CL+Buffer). As the figure shows, the buffering does not improve the average performance. This is a result already expected when flow control with buffer was mentioned in Section 3. Without a sophisticated scheme like the one presented here, a simple buffering in the closed-loop scheme is not effective. A more extensive comparison between the proposed method and general buffering methods (e.g. buffering in routers, memory controllers, and network interfaces) will be beyond the scope of this paper. We will investigate this issue in our future work.

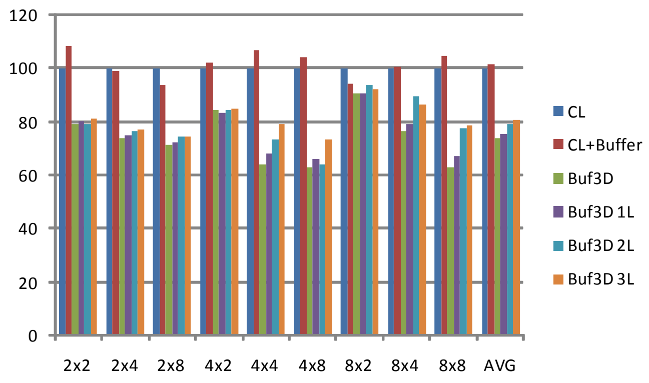

Figure 7 Comparisons of normalized runtime

5.2 Digital TV SoC Case

The digital TV SoC consists of +30M gates and more than 50 masters/slaves for QFHD size (3840x2160) video processing [17]. Many of masters/slaves are connected to low bandwidth local AXI/AHB/APB buses. We evaluate the effect of the proposed scheme by applying it to the backbone on-chip network of this SoC which is similar to the one in Figure 3. The backbone network has four memory modules, four memory controllers (PL340's), and four Buf3D's. We use the same configurations of Buf3D and memory controller (the fifo size is 16) as in the synthetic test cases. It has eight masters: four video codec IPs and four different pixel processing IPs (e.g. noise reduction, mixer, 3D graphics, etc.).

In terms of memory access behavior of masters, video codec IPs and pixel processing IPs have different access patterns. Each of video codec IPs accesses only one dedicated memory module since each IP covers 1/4 of total video frame. Thus, the memory accesses from different video codec IPs do not overlap with each other at memory modules. However, each of pixel processing IPs accesses all the four memory modules. This design decision results mainly from the fact that duplicating pixel processing IPs incurs too high area overhead. Thus, there are conflicts at the input of pixel processing IPs when the requested data transfers arrive simultaneously at the same master inputs.

Figure 8 shows the performance comparison for digital TV SoC case. The proposed scheme gives up to 18.4% and 12.6% improvement (in the case of Buf3D in the figure) in aggregate memory utilization and average memory access latency (from

[10] The size of destination id, i.e. port id size is $\log_2 M$ where M is the number of masters covered by all the Buf3D's.

[11] In this case, the aggregate memory bandwidth is reversely proportional to the total runtime since the aggregate bandwidth is equal to (total transferred data volume that is fixed) / (total runtime).

master's request to the end of data transfer at the master). Aggregate memory utilization is defined to be the average of each memory module's utilization. The memory utilization of each memory module is defined to be (# clock cycles used for data transfer) / (# total clock cycles).

Figure 8 also shows the effect of GI bus delay on the aggregate memory utilization and latency. We obtain the best case when the GI bus has a combinational delay (Buf3D). As the GI network latency increases up to three clock cycles, the aggregate memory utilization and average memory access latency degrade slowly. Thus, when the delay is three clock cycles, the improvements are still significant, 8.5% and 10.3% in utilization and latency, respectively. The results show that the proposed method is still effective when the GI bus has a relatively long latency cycle. However, in the case that the latency becomes more severe, application-specific optimizations (topology and floorplan) of GI bus design and latency-aware methods for the open-loop flow control will be required, which is our future work.

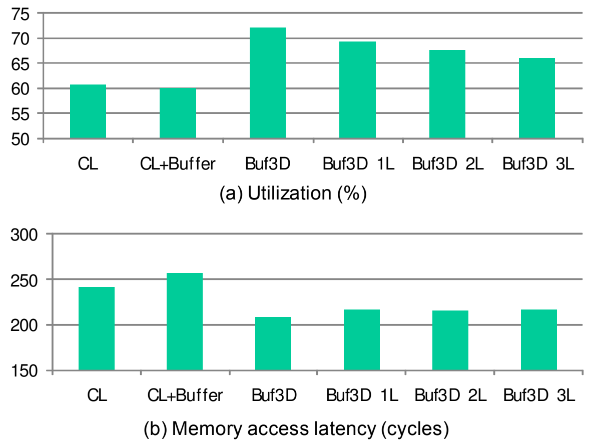

Figure 8 Performance comparison in the DTV SoC case

Table 1 Area comparison for digital TV SoC case (in gates)

	Conventional	Proposed
Total area of OCN	646K	762K
Area of four Buf3D's	0	116K (15.2%)

Table 1 shows the area comparison obtained from synthesis (Synopsys DC) with 200MHz as the target frequency at a proprietary 65nm low power technology. The total area of on-chip network (OCN) includes that of PL301 (including internal pipeline stages to meet 200MHz), PL340's, and Buf3D's. As shown in the table, the proposed scheme has a small area overhead (15.2%) compared with the total on-chip network area including that of Buf3D's. The area overhead of the GI bus (32b) is very small w.r.t. that of total on-chip network area. As a specific example, assuming a fully connected implementation of GI bus among the four Buf3D's with three cycle latency, the gate count of the GI bus will be about 5.8Kgates (32b/link * 6 links * 3 * 10gates/b ~ 5.8Kgates) where a 1 bit F/F is assumed to have 10 gates. Considering the total chip size (+30Mgates), the overhead (116Kgates, less than 0.39% of total chip area) can be justified by the significant improvement (18.4%) in the aggregate memory bandwidth of on-chip communication over the backbone network.

6. Conclusion

In this paper, we propose an open-loop flow control scheme and its implementation, a buffer called Buf3D. The proposed scheme reduces the conflicts of data transfers from multiple memory modules to the same masters thereby improving the memory utilization by up to 18.4% with a small area overhead (15.2%) of on-chip network for an industrial strength DTV SoC design.

The design space related with Buf3D can be large since there are many parameters impacting the performance/area/power of Buf3D's: reordering depth and fifo size per Buf3D, local and global arbitration schemes (including QoS) per Buf3D and per destination id, respectively, the assignment of Buf3D's to memory modules, etc. In addition, since the proposed open-loop scheme does not take into account congestion in the NoC, there can be further optimizations by integrating both open and closed-loop flow control schemes to tackle both conflicts at destination and network links.

7. References

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

[2] F. Li, *et. al.*, "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory", Proc. ISCA, 2006.

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

[4] Intel TeraScale Computing, available at http://www.intel.com

[5] S. Vangal, *et. al.*, "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS", Proc. ISSCC, 2007.

[6] K. Asanovic, *et. al.*, "The Landscape of Parallel Computing Research: A View From Berkeley", available at http://view.eecs.berkeley.edu/wiki/Main_Page

[7] L. Hsu, *et. al.*, "Exploring the Cache Design Space for Large Scale CMPs", SIGARCH Computer Architecture News, 33(4), 2005.

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

[9] A. Pullini, *et. al.*, "Fault Tolerance Overhead in Network-on-Chip Flow Control Schemes", Proc. SBCCI, 2005.

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

[11] U. Y. Ogras and R. Marculescu, "Prediction-based Flow Control for Network-on-Chip Traffic", Proc. DAC, 2006.

[12] D. Qiu and N. B. Shroff, "A Predictive Flow Control Scheme for Efficient Network Utilization and QoS", IEEE Trans. on Networking, 12(1), Feb. 2004.

[13] D. Bertsekas and R. Gallager, "Data Networks", Prentice-Hall, 1992.

[14] A. Radulescu, *et. al.*, "An Efficient On-chip NI Offering Guaranteed Services, Shared Memory Abstraction, and Flexible Network Configuration", IEEE Trans. on CAD, 24(1), 2005.

[15] E. Nilsson, *et. al.*, "Load Distribution with the Proximity Congestion Awareness in a Network on Chip", Proc. DATE, 2003.

[16] P. Avasare, *et. al.*, "Centralized End-to-End Flow Control in a Best-Effort Network-on-Chip", Proc. EMSOFT, 2005.

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