CARS: Congestion-Aware Request Scheduler for Network Interfaces in NoC-based Manycore Systems

Masoud Daneshtalab, Masoumeh Ebrahimi, Juha Plosila, Hannu Tenhunen
Department of Information Technology, University of Turku
{masdan, masebr, juplos, pakrli, hanten }@utu.fi

Abstract- Network congestion is a critical issue of memory parallelism in network-based manycore systems where multiple memories can be accessed simultaneously. Therefore, a congestion-aware method is necessitated to deal with the network congestion. In this paper, we present a streamlined method in order to reduce the network congestion. The idea is to use the global congestion information as a metric in network interfaces to reduce the congestion level of highly congested areas. Network interfaces connected to memory modules are equipped with an adaptive scheduler using the global congestion information to reduce additional traffic to congested areas. Experimental results with synthetic test cases demonstrate that the on-chip network utilizing the proposed adaptive scheduler presents up to 23% improvement in average latency.

Keywords: Network-on-Chip, Congestion-Aware Scheduler, Network Interfaces

I. INTRODUCTION

Network-on-Chip (NoC) has been emerged as a solution for the communication requirement of many-core processors (MCP) [1]. Network Interface (NI) provides communication between IP cores and the network infrastructure using a standard communication protocol like AXI [2]. In-order delivery should be handled in MCPs when using memory access parallelization by sending requests from a master IP core to multiple slave memories [3][4]. In NoC-based MCP architectures, in addition to the in-order delivery, network congestion affects the system performance considerably [5][6][7][8][9]. To deal with the network congestion in MCPs, several congestion-aware methods have been presented such as adaptive routing algorithms (output selection) [10][11][12][13][14] and router's arbitration policies (input selection) [15][16]. Input and output selection can be based on local or global congestion information. However, none of the proposed approaches have used the congestion information in network interfaces for serving (memory) requests. This approach has two main advantages. First, it reduces the waiting time of requests arriving from less-congested regions to receive service in the slave node (memory). Second, it temporarily decreases the injection of traffic to congested area(s). Therefore, in this paper, we present an efficient congestion-aware method for NoC-based MCP, named Congestion-Aware Request Scheduler (CARS), in order to reduce the network congestion. The key idea of CARS is to prioritize requests in slave-side (memory) network interfaces according to the global congestion information.

The paper is organized as follows. Section II reviews the background and related works. In Section III, the proposed CARS method is presented while the experimental results are discussed in Section IV. Finally, the summary and conclusion are given in the last section.

II. BACKGROUND AND RELATED WORK

In distributed shared memory systems, when a master sends requests to different memories, the responses are required to be returned in the same order in which the master issued the addresses, and therefore a reordering mechanism in NoC-based MCPs should be integrated in network interfaces [17][18]. To be compatible with existing transaction-based IP-cores, we use the AMBA AXI protocol [2] where master cores initiate transactions by issuing read and write requests and one or more slaves (memories) receive and execute each request. Transactions from the same master core, but with different IDs have no ordering restriction while transactions with the same ID must be completed in order.

Yang et al proposed NISAR [19], a network interface architecture using the AXI protocol capable of packet reordering based on a look up table; NISAR uses a statically partitioned reorder buffer and thereby it has a simple control logic, but suffers from low buffer utilization in different traffic patterns. In addition, NISAR does not support burst transactions which can be considered a shortcoming. The drawbacks of NISAR have been addressed in [17] where separate master and slave network interfaces equipped with an efficient buffer management structure. However, none of the aforementioned interfaces have considered the network congestion as a metric in order to balance the network traffic.

Based on the AXI model, network interfaces can be classified into the master network interfaces (Fig. 1(a)) and slave network interfaces (Fig. 1(b)). In the master network interface, the forward path is composed of an AXI-Queue, and a Packetizer unit, while the reverse path, receiving the responses from the network, is composed by a Packet-Queue, and a Depacketizer unit; the Reorder unit is shared between the forward and reverse paths. Reorder unit is the most influential part of the network interface. In the forward path, preparing the sequence number for corresponding transaction ID and avoiding the overflow of the reorder buffer by the admittance mechanism are provided by this unit. On the other side, in the reverse path, this unit determines where the outstanding packets from the Packet-Queue should be transmitted (Reorder unit or Depacketizer), and when the packets in the reorder buffer could be released to the Depacketizer unit. As illustrated in Fig. 1(b), to avoid losing the order of header information (transaction ID, sequence number, and etc) carried by arriving requests, a FIFO has been considered in the slave network interface. After processing a request in the slave core, the response packet should be created by the Packetizer unit. As can be seen from Fig. 1(b), to generate the response packet, after the header content of the corresponding request is invoked from the FIFO, and some parameters of the header (destination address, and packet size, and etc) are modified by the adapter, the response packet will be formed. Indeed, the components of slave-side interface in both forward and reverse paths are almost similar to the master-side interface components, except the Reorder unit.

III. CONGESTION-AWARE REQUEST SCHEDULER (CARS)

A. The General Idea of CARS

The main idea of CARS is to utilize global congestion information to reduce sending packets to the congested area by prioritizing packets in slave network interfaces. Packets collect the congestion information along paths and carry it to slave network interfaces. However, as indicated in the CATRA approach [20], the information of nearby routers are sufficient to estimate the global congestion of different regions. The reason is that, the information of faraway routers is outdated when it arrives to the source node. Based on this perspective, in CARS, packets start collecting information when their

978-3-9815370-0-0/DATE13/©2013 EDAA

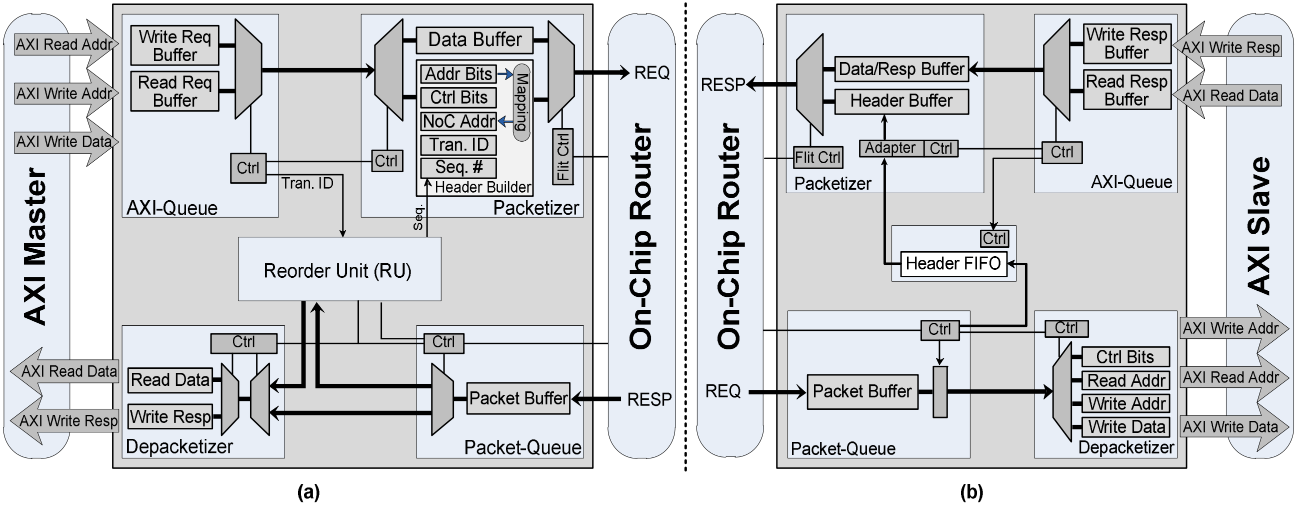
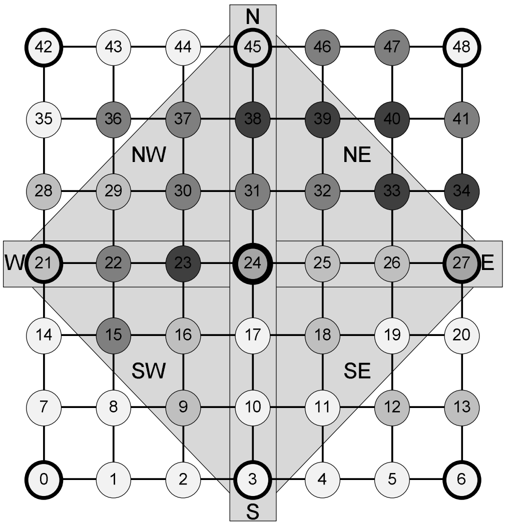


Fig. 1. (a) Master-side and (b) slave-side network interface architectures [17]

distances are equal or less than three hops from the destination node. Each slave network interfaces maintains a table to store the estimated congestion value carried by packets.

Packets are belonging to eight regions in the network. As shown in Fig. 2, there are four triangular regions as NE, NW, SE, and SW and four columnar regions as N, S, E, and W. Triangular regions also includes the congestion at columnar regions as packets coming from NE, NW, SE, and SW directions may pass through columnar regions to reach the slave network interface. On the other hand, if the slave and master nodes are located in the same row or column, only the congestion information of columnar regions is useful.

We explain the CARS approach using an example illustrated in Fig. 2 where master nodes 0, 3, 6, 21, 24, 27, 42, 45, and 48 send requests to the slave node (memory) 24. The congestion level of routers is represented by colored nodes (i.e. the darker color of the node is, the higher congestion level is). By using the congestion information carried by requests from master network interfaces to the slave network interface, the congestion status of each region can be estimated at the slave network interface. In this example, the congestion levels of different regions are assumed to be the ones shown in the congestion table (CT) of Fig. 2. Since the regions NE, NW, N, and W are highly congested, delivering the response messages to the master nodes 48, 42, 45, and 21 not only exacerbates the congestion in the congested areas but also increases the average latency of packets. This problem can be mitigated by prioritizing requests based on the congestion level of regions such that requests from less-congested regions prioritize over the other requests. This approach has two main advantages. First, it reduces the waiting time of requests arriving from less-congested regions to receive service in the slave node (memory). Second, it temporarily decreases the injection of traffic to congested area(s). Based on this scheduler mechanism, in the example of Fig. 2, the request coming from the south direction is served earlier than those from SE, E, and SW directions.



	CT
NE	5
NW	4
SE	1
SW	3
N	4
S	0
E	2
W	5

Fig. 2. Different congestion levels in routers and regions.

B. Implementation of CARS

We reserve a 3-bit field, named Congestion Status (CS), in the header of each packet to store the congestion information of the path being traversed by the packet. As it is already mentioned, packets start collecting the information when the distance toward the destination node is less than or equal to three hops. The congestion value of a router indicates the number of congested input ports corresponded to response messages. In this paper, we use two virtual channels per input port, such that one virtual channel is allocated to request messages and another one to response messages. In this way, the maximum congestion value of a router is five that can be coded in 3 bits. The congested input port implies that the number of occupied cells of the buffer is larger than a threshold value (i.e. 70% of the total buffer slots). The congestion value of the router and the CS value in the packet's header flit are combined together and the resulted value is stored in the header flit as the new CS. Here, the combination of the values means the average of two values.

In slave network interfaces, the estimate congestion values of regions are kept in the congestion table (CT). Once a packet enters a slave network interface, the congestion information, i.e. CS, is extracted from the packet's header and combined with the existing value in the congestion table. The scheduler selects a request coming from the less congested region. This may result in starving the requests coming from the congested regions. To prevent starvation, the priority values of defeated packets are incremented after each scheduling process. Fig. 3 shows the pseudo code of request scheduling mechanism in slave network interfaces.

As the routing function returns a set of admissible channels to send a packet (either request or response message), the selection function chooses one of them based on the local or non-local congestion information. In this paper, we employ the odd-even [11] routing algorithm which is an adaptive routing model without requiring virtual channels. In this method, the selection is made locally according to the congestion condition of the neighboring nodes. The number of occupied buffer cells at the corresponding input buffers of the neighboring nodes is considered as the congestion metric. Therefore, if the occupied space of the input buffer is larger than a threshold value, then the congestion flag of the input port becomes ‘1’, otherwise ‘0’. Note that for simplicity, we do not consider the non-local congestion information in routing decisions.

```
i: i(th) request
Region(req(i)): congestion value of the region
                related to i(th) request
W: waiting periods of packet
----------------
process Find_MinValue is
begin
  for ‘i=0 to all requests’ loop
     if “req(i) is newly arrived” then
        W(i) <= 0;
     else
        W(i) <= W(i) + 1;
     end if;
     if (Region(req(i))-W(i)) < MinValue then
       MinValue <= Region(req(i))-W(i);
       select   <= i;
     end if;
  end loop;
end process;
```

Fig. 3. Pseudo code of the request scheduling mechanism in slave network interfaces.

IV. Experimental Results

To evaluate the CARS method a NoC simulator is implemented with VHDL. The simulator models all major components of the NoC such as network interfaces, routers, and wires. The on-chip network composed of the presented CARS method is compared with the baseline architecture in terms of average network latency under different traffic patterns. The baseline architecture comprises typical master slave network interfaces without using the CARS method.

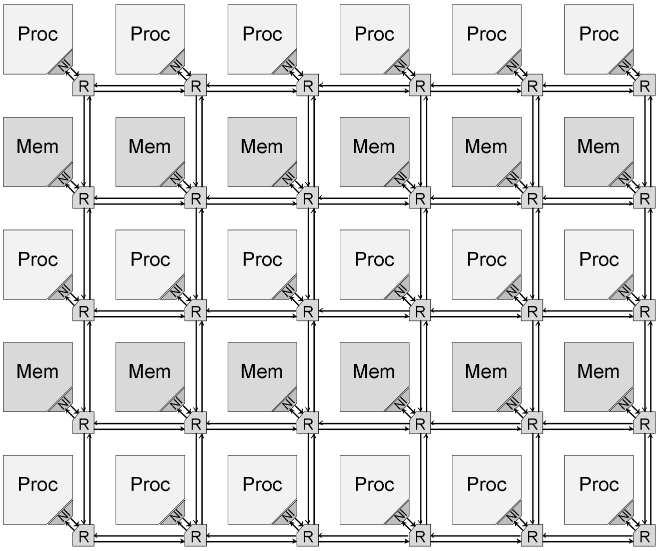


Fig. 4. 6×5 NoC layout.

A. System Configuration

In this paper, we use a 30-node (6×5) 2D mesh on-chip network for the entire architecture. As illustrated in Fig. 4, out of 30 nodes, 18 nodes are assumed to be processor (master cores, connected by master network interfaces) and other 12 nodes are memories (slave cores, connected by slave network interfaces). The processors are 32b-AXI and the memories are DRAM (t_{RP}-t_{RCD}-t_{CL}=2-2-2, 32b). We adopt a commercial memory controller with memory interface, DDR2SPA module from Gaisler ip-cores [22] where it is placed between the memory and the slave network interface. In addition, each router structure includes input buffers, a VC (Virtual Channel) allocator, a routing unit, a switch allocator and a crossbar. Each router has five input/output ports, and each input port of the router has two VCs. Packets of different message types (request and response) are assigned to corresponding VCs to avoid message dependency deadlock [23]. The routing algorithm, link width, number of VCs, buffer depth of each VC, and traffic type are the other parameters which are specified for the simulator. The routers adopt the Odd-even [11] routing and utilize wormhole switching. For all routers, the data width (flit size) was set to 32 bits, and the buffer depth of each VC to 5 flits. For the request, the command and all its control bits (flags) are included in the first flit of the packet, the memory address is set in the second flit, and the write data (in the case of a write command) are appended at the end. For the response message, the control bits are included in the first flit while the read data are appended at the end if the response relates to a read request. Hence, the packet length for write responses and read requests is 1 flit and 2 flits, respectively, while the packet length for data messages, representative of read responses and write requests, is variable and depends on the write request/read response length (burst size) produced by a master/slave core. As a performance metric, we use latency defined as the number of cycles between the initiation of a request operation issued by a master (processor) and the time when the response is completely delivered to the master from the slave (memory). The request rate is defined as the ratio of the successful read/write request injections into the network interface over the total number of injection attempts. All the cores and routers are assumed to operate at 1GHz; and for fair comparison, we keep the bisection bandwidth constant in all configurations. All memories (slave cores) can be accessed simultaneously by each master core continuously generating memory requests. Furthermore, the size of each queue (and FIFO) in the network is set to 8×32 bits and the size of the reorder buffer is set to 48 words.

B. Performance Evaluation

To evaluate the performance of the proposed schemes, uniform and non-uniform/localized synthetic traffic patterns are considered. These workloads provide insight into the strengths and weaknesses of the CARS method in the congestion-aware on-chip networks, and we expect applications stand between these two synthetic traffic patterns [24][25]. The random traffic represents the most generic case, where each processor sends in-order read/write requests to memories with a uniform probability. Hence, the target memory and request type (read or write) are selected randomly. Eight burst sizes, from 1 to 8, are stochastically chosen according to the data length of the request. In the non-uniform mode, 70% of the traffic is local requests, where the destination memory is one hop away from the master core, and the rest 30% of the traffic is uniformly distributed to the non-local memory modules.

Fig. 5 (a) and (b) show the simulation results under the uniform and non-uniform traffic models, respectively. In the presented configuration, the on-chip network utilizing the CARS method is compared with the network without utilizing it. As demonstrated in both figures, compared with the baseline architecture, the NoC using CARS reduces the average latency when the request rate increases under the uniform and non-uniform traffic models. The performance gain near the saturation point (0.6) under the uniform and non-uniform traffic models is about 27% and 23%, respectively. The reason for such an improvement is due to using the presented adaptive scheduler mechanism, which can diminish the congested areas so that the average network latency decreases.

C. Hardware Cost Analysis

In this section, the hardware cost of the scheme is evaluated. Since all queues (and FIFOs) are equal in the size, it would not affect the comparison. The network interfaces are synthesized with Synopsys Design Compiler using the UMC 90nm technology with a timing constraint of 1 GHz for the system clock and supply voltage 1 V.

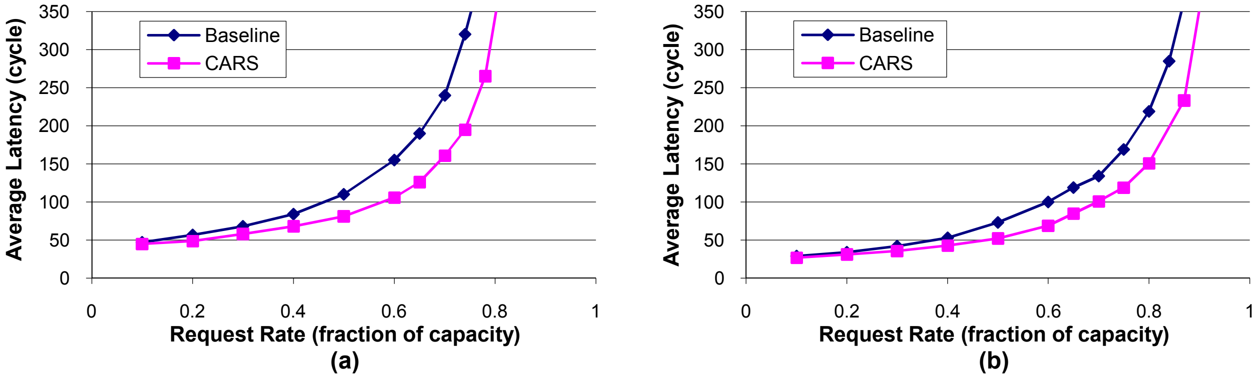


Fig. 5. Performance evaluation under (a) the uniform and (b) non-uniform traffic models.

Table 1. Hardware implementation details.

Components	Area (mm^2)	Power (mW)
Slave network interface	0.0428	17
Master network interface	0.0755	28
CARS	0.0491	24
Router	0.1853	65

The synthesized netlist is then again verified through post synthesis simulations. Finally, we perform place-and-route, using Cadence SoC Encounter, to have precise power and area estimation in wire-dominated structures. The layout areas and power consumptions of different network components are listed in Table 1. As can be seen from the table, the adaptive scheduler scheme in the slave network interface imposes 9% hardware overhead while offering 23% performance gain.

V. Summary and Conclusion

Network congestion is a critical issue in such architectures where processors communicate with memory modules through the on-chip network. To manage the network congestion, an efficient method based on the global congestion information is presented at slave network interfaces. The micro-architecture along with the implementation of the proposed method is presented. A cycle-accurate simulator is used to evaluate the efficiency of the proposed congestion management method along with employing the network interface.

References

[1] Y. Hoskote et al., "A 5-GHz mesh interconnect for a teraflops processor," in Proc. IEEE Micro, 27:51–61, 2007.

[2] ARM, AMBA AXI Protocol Specification, 2004.

[3] W. Kwon et al., "A Practical Approach of Memory Access Parallelization to Exploit Multiple Off-chip DDR Memories," in Proc. DAC, pp. 447-452, 2008.

[4] A. Radulescu et al., "An Efficient On-Chip NI Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration," in Proc IEEE TCAD, 24(1), 2005.

[5] J. Duato et al., "Interconnection Networks: An Engineering Approach." Morgan Kaufmann, 2002.

[6] P. Gratz et al., "Regional Congestion Awareness for Load Balance in Networks-on-Chip," in Proc. HPCA, pp. 203–214, 2008.

[7] A. Singh et al., "GOAL: A Load-Balanced Adaptive Routing Algorithm for Torus Networks," In International Symposium on Computer Architecture, pp. 194–205, 2003.

[8] M. Ebrahimi et al., "HARAQ: Congestion-Aware Learning Model for Highly Adaptive Routing Algorithm in On-Chip Networks," in Proc. of NOCS, 2012.

[9] A. Singh et al., "Globally Adaptive Load-Balanced Routing on Tori" IEEE Computer Architecture Letters, v.3, I.1, pp.2-6, 2004.

[10] M. Li et al., "DyXY - a proximity congestion-aware deadlock-free dynamic routing method for network on chip," in Proc. DAC, pp. 849-852, 2006.

[11] G. Chiu, "The Odd-Even Turn Model for Adaptive Routing," IEEE Tran. On Parallel and Distributed System, pp. 729-738, 2000.

[12] M. Dehyadegari et al., "An Adaptive Fuzzy Logic-based Routing Algorithm for Networks-on-Chip," in Proceedings of 13th IEEE/NASA-ESA International Conference on Adaptive Hardware and Systems (AHS), pp. 208-214, June 2011, USA.

[13] M. Daneshtalab et al., "Adaptive Input-output Selection Based On-Chip Router Architecture," Journal of Low Power Electronics, v. 8, no. 1, pp. 11-29, 2012.

[14] M. Daneshtalab et al., "A Generic Adaptive path-based routing method for MPSoCs," Journal of Systems Architecture (JSA-elsevier), v. 57, no. 1, pp. 109-120, 2011.

[15] M. Daneshtalab et al., "Adaptive Input-output Selection Based On-Chip Router Architecture," Journal of Low Power Electronics *(JOLPE)*, Vol. 8, No. 1, pp. 11-29, 2012.

[16] D. Wu et al., "Improving Routing Efficiency for Network-on-Chip through Contention-Aware Input Selection," In Proc. of ASP-DAC, pp. 36 – 41, 2006.

[17] M. Daneshtalab et al., "Memory-Efficient On-Chip Network with Adaptive Interfaces," IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, v. 31, no. 1, pp. 146-159, 2012.

[18] M. Daneshtalab et al., "A systematic reordering mechanism for on-chip networks using efficient congestion-aware method," Elsevier Journal of Systems Architecture (*JSA-elsevier*), 2012.

[19] X. Yang et al., "NISAR: An AXI compliant on-chip NI architecture offering transaction reordering processing", in Proc. of ASICON, pp. 890-893, 2007.

[20] M. Ebrahimi et al., "CATRA-Congestion Aware Trapezoid-based Routing Algorithm for On-Chip Networks," in Proc. of DATE, pp. 320-325, 2012.

[21] S. E. Lee et al., "A Generic Network Interface Architecture for a Networked Processor Array (NePA)", In Proc. of ARCS'08, pp. 247-260, 2008.

[22] Gaisler IP Cores, http://www.gaisler.com/products/grlib/, 2009.

[23] S. Murali et al., "Designing message-dependent deadlock free networks on chips for application-specific systems on chips," In Proc. of VLSI-SoC, pp. 158-163, 2006.

[24] R. Das et al., "Design and Evaluation of a Hierarchical On-Chip Interconnect for Next-Generation CMPs," in Proc. of HPCA, pp. 175-186, 2009.

[25] P. P. Pande et al., "Performance Evaluation and Design Trade-offs for Network on Chip Interconnect Architectures," IEEE Transactions on Computers, v. 54, no. 8, pp. 1025-1040, 2005.