Advance Virtual Channel Reservation

Boqian Wang
KTH Royal Institute of Technology
National University of Defense Techonology
boqian@kth.se

Zhonghai Lu
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
zhonghai@kth.se

***Abstract*—We present a smart communication service called Advance Virtual Channel Reservation (AVCR) to provide a highway to packets, which can greatly reduce their contention delay in NoC. AVCR takes advantage of the fact that we can know or predict the destination of some packets ahead at the network interface (NI). Exploiting the time slack before a packet is ready, AVCR establishes an end-to-end highway from the source NI to the destination NI. This highway is built up by reserving virtual channel (VC) resources ahead and at the same time, offering priority service to those VCs in the router, which can therefore avoid highway packets' VC allocation and switch arbitration delay in NoC. Additionally, optimization schemes are developed to reduce VC overhead and increase highway utilization. We evaluate AVCR with cycle-accurate full-system simulations in GEM5 by using all benchmarks in PARSEC. Compared to the state-of-art mechanisms and the priority based mechanism, experimental results show that our mechanism can significantly reduce the target packets' transfer latency and effectively decrease the average region-of-interest (ROI) time by 22.4% (maximally by 29.4%) across PARSEC benchmarks.**

I. Introduction

In recent years, the core count is growing quickly in Chip Multi/Many-core Processors (CMPs) to support more threads running at the same time. In addressing the communication challenge, Network-on-Chip (NoC) has become the de facto solution for CMPs. Due to pipelined transmission and better resource utilization, virtual-channel (VC) flow control is commonly used in NoC in which the physical channel is shared across several VCs. Nevertheless, packets still suffer a lot of contention delay caused by VC allocation and switch arbitration during their transfer in NoC. Previously, a number of latency-reduction techniques [1] [2] [3] [4] have been proposed for NoCs over years, which intend to provide a general communication service or speed up specific packets. However, their packet acceleration results are limited and they often do not take the cache coherence protocol and application behaviors into consideration. As a result, it is difficult to effectively translate the reduction of packets latency directly into the improvement of the program performance.

In this paper, we propose a latency reduction scheme from a fresh perspective. We observe that for some packets, before they are ready to be transferred via the network interface (NI), their destinations can be known or predicted in advance due to the packet transfer pattern, the cache coherence principle or application behaviors. Taking advantage of this opportunity, we propose *Advance Virtual Channel Reservation (AVCR)*, an end-to-end highway service, to accelerate the packet transfer process in NoC. This highway is established by sending out a single-flit reservation packet, which will reserve all the VCs end-to-end from source to destination, before the target packet is ready to traverse in the NI. Additionally, the reserved buffer will be recorded in the router so that the switch can first offer service to flits in the reserved VC buffer. To support AVCR, we design a dedicated controller in the NI and router. Besides, we propose a highways combination mechanism and an algorithm for predicting the best timing when the reservation packet should be sent out, both of which can optimize buffer utilization. Finally, we implement an application-level example, where we apply AVCR to critical packets generated by slow progress threads or involved in the lock competition phase, directly translating the packet delay reduction into application improvement.

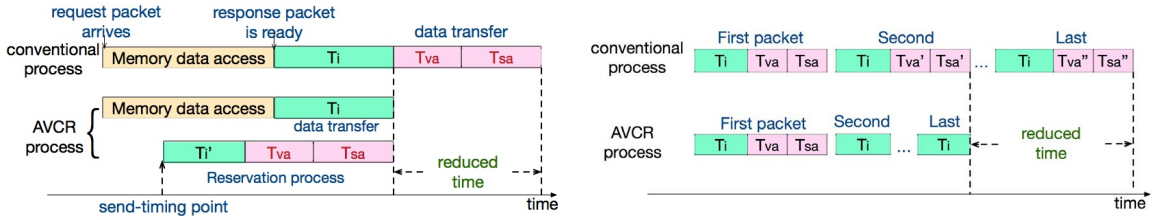
The rest of the paper is structured as follows. Related work is discussed in Section II. In Section III, we present the AVCR concept and mechanism, followed by the AVCR implementation and application in Section IV. Experiments are reported in Section V. Finally, we conclude in Section VI.

II. Related work

Latency reduction techniques. Techniques which reduce memory access latency can be found in [1] [5]. In [1], Sharifi *et al.* offers a higher priority to packets that will access idle memory banks and at the same time gives higher priority to response packets with high latency. [5] focuses on the DRAM access packet. It models the round-trip latency prediction, based on which, different priorities are offered to packets in the DRAM access process. In [3], Liu *et al.* propose a highway based Time Division Multiplexing (TDM) NoC. It builds a highway with special buffer queues to enhance the throughput and reduce data transfer ideally. [6] [7] use network coding to improve the NoC performance. A mechanism is offered to support different packets that use the same VC resource in [4]. It guarantees high VC utilization and reduces the packet latency. There are also many works about the architecture design. [8] proposes a single-cycle NoC router which is realized by VC pre-built between adjacent routers. It conducts switch allocation ahead so that it takes one cycle for a flit to go through the router.

Resource reservation techniques. There are a number of previous works investigating resources reservation. For example, in [9], Peh and Dally illustrate a flit-reservation flow control mechanism. It splits the network into two parts, which separately support the transfer of control and data flits. For each packet, it needs an additional head control flit and

978-3-9819263-2-3/DATE19/©2019 EDAA



(a) Case 1: AVCR process for response packets

(b) Case 2: AVCR process for request packets

Fig. 1. The AVCR process in contrast to the conventional VC scheme

an additional body/tail control flit to reserve the router in the next hop. This flow control method aims to reduce the latency caused by credit turnaround time in VC mechanism and it needs another network and additional flit overhead. In [2], Lu and Jantsch propose virtual circuits in TDM NoC to satisfy communication QoS requirements. In [10], the authors use VC reservation to provide QoS guarantees by designing a new router. [11] proposes a pre-reservation mechanism. It is realized by path coding in the NI and applied to mutual exclusive communication processes. In [12] [13], the authors set different kinds of packet types and use different categories for them. Abousamra *et al.* [14] propose a circuit switching scheme to accelerate the critical message.

III. Concept and Mechanism

A. Concept

AVCR is a packet-acceleration scheme designed to accelerate target packets in NoC. In AVCR, before the target packet arrives at the NI, a *single-flit reservation packet* will be sent ahead to the destination. During its transfer in NoC, it will reserve all the flit buffers it gets through so that only flits belonging to the target packet can occupy them. At the same time, the reserved buffer will be recorded in the router so that the switch will first serve it. The reserved buffers, together with the priority service, makes a *highway*, where the target packet incurs no delay caused by VC allocation (T_{va}) and switch arbitration (T_{sa}), enjoying an ideal transfer time (T_i).

AVCR is built upon the fact that the packet's destination can be often known or predicted before the packet is ready for delivery, i.e., *knowing destination ahead*.

The first case happens to *response packets*. In the optimized directory-based cache coherence protocol [15] (two-level cache architecture in our example) which is commonly used today, when the L1 cache or memory controller receives a request packet (from the L2 cache), the response packet's destination must be the one tagged as requestor in the request packet. However, if the L2 cache receives a request packet, we can not conclude the destination of the triggered packet, since it depends on the cache hit or miss. In this case, we proactively predict a data hit in the L2 cache, which is always of high possibility, making it the same scenario as the one described above. If the prediction is wrong, the AVCR mechanism will delete this highway after the timeout. Apparently, the slack time from knowing destination and the actual packet's arrival indicates the potential for delay reduction. If it is too small, AVCR has little to save. Otherwise, AVCR can be significant. To genuinely examine the opportunity, we conducte measurements in GEM5 with a 64-core two-level cache CMP running the PARSEC benchmarks. The detailed system configuration can be found in Section V. Table I shows the slack time interval between destination known and packet's arrival for response packets in the memory hierarchy. As can be seen, the slack time increases from the L1 cache to the L2 cache and then to the off-chip memory. Even for the on-chip part, 10-20 packet cycles can be saved by AVCR.

TABLE I
Time interval of knowing destination ahead

L1 cache	L2 cache	Off-chip memory
Around 10 cycles	Around 18 cycles	Over 45 cycles

The second case happens to *request packets*. Generally speaking, the destination prediction of a request packet will be a very sophisticated problem, taking the unpredicted cache data state into consideration. However, for the repeated request packets, which transfer to the same destination in a time period, the destination of subsequent packets can be directly obtained from the first request packet in the series. In fact, this is the result of a common synchronization process related to program behaviors in a multithreading parallel program. More details will be discussed in Section IV.

Making full use of the knowing-destination-ahead possibility can greatly reduce packets' delay in NoC. The detailed process is shown in Figure 1. In Figure 1(a), when the cache or memory controller receives a request packet, it will trigger a data access process if the data is valid in the cache (cache hit) or memory. Conventionally, the response packet will be sent to the destination by NoC after data access process and incur VC allocation delay T_{va} and switch arbitration delay T_{sa} in addition to the ideal transfer time T_i during the transfer process. But in our AVCR process, once the request packet arrives at the NI, a single-flit reservation packet will be generated in the NI and sent to the known or predicted destination of the corresponding response packet to build a highway in advance. The reservation packet will still suffer T_i', T_{va} and T_{sa}, while ideally the response packet can reach the destination in a certain time T_i without any conflict. In this way, the T_{va} and T_{sa} of the response packet are successfully reduced by our mechanism. Apparently, the *send-timing point* is critical, as it determines when a reservation packet should be sent out, so that the reduced time and buffer utilization are both maximum. (See details in Section III-B2)

In the conventional process in Figure 1(b), the request packets will be sent to the same destination repeatedly and each will incur T_{va} and T_{sa}. In the AVCR process, the first

packet is responsible for building a highway (like the process illustrated above), then subsequent packets can avoid their conflict delays. Thus, the reduction time is the sum of all subsequent packets' T_{va} and T_{sa}.

B. Mechanism

1) Basic AVCR process: We first explain the basic AVCR process for the response packet, as shown in Figure 2. When the NI receives a request packet, it will transfer this packet to the connected memory and at the same time the AVCR controller generates a single-flit reservation packet, the destination of which is known or predicted by the request packet. This reservation packet will be flitisized to a 'HEAD' flit and then be inserted into the VC buffers connected to NoC in an appropriate time. We take advantage of the conventional VC flow control mechanism, where the 'HEAD' flit is responsible for reserving buffers in NIs and routers which will only be released by a 'TAIL' or 'HEAD_TAIL' flit. Additionally, in the router, flits in a reserved buffer will always be served first by the switch. In this way, when the corresponding response packet is ready, it will be flitisized as 'BODY' and 'TAIL' flits and then be inserted into the reserved buffer in the NI. So it will be transferred to the destination by the highway, decreasing packet latency which we define as the time a packet takes to transfer from the VC buffer of the NI to the destination NI. The 'HEAD' flit will be blocked if there are no buffers available for reservation (by default, only one VC per inport is used for highway) and if the reservation 'HEAD' flit is caught up by the following 'BODY' flit, the rest process will be more like a conventional packet transfer process. Finally, the destination NI will delete the reservation packet ('HEAD' flit) and deflitisize the rest flits as a complete response packet. To make the mechanism complete, we add a timeout parameter in each NI to record the lifetime of a reserved highway. If the predictive result is wrong or the off-chip memory incurs a page fault, the timeout parameter will trigger the NI to generate a clean packet which will be flitisized as a 'TAIL' flit to release this highway, forbidding a long time occupation of buffer resources. Besides, we apply AVCR process to

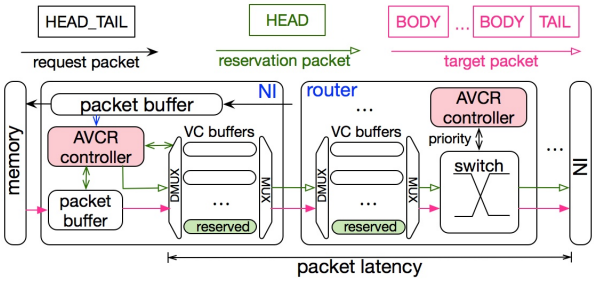


Fig. 2. An example of basic AVCR mechanism

single-flit request packets. The first request packet will be flitisized as a 'HEAD' flit (instead of the 'HEAD_TAIL' flit type in the conventional mechanism) to reserve VCs from source to destination. The consecutive request packets, which have the same destination as the first one, will be marked as 'BODY' flits and sent to the reserved VCs in the source NI. Those body flits will enjoy the benefit of reserved VCs. When the destination NI receives those 'HEAD' (the first one) and 'BODY' flits, it will handle each of them separately as a complete request packet. When this request process ends, the timeout parameter is responsible for deleting this highway and releasing all the resources available.

2) Send-timing point prediction and highway combination: We develop a method to predict the most suitable timing point when the reservation head flit should be sent out. The best case is that when the head flit arrives at the destination, the following body flit comes in the next cycle. This *sending point* T can be predicted by Equation 1 where T_h is the pipeline stage number of a router and H is the number of routers a flit will get through. T_p is the time before the response packet is ready which can be predicted by the historical information. If $T < 0$, the reservation head flit should be sent out immediately once receiving the request packet. Otherwise, the reservation packet should be sent out T cycle(s) later.

$$T = T_p - T_h \times H \tag{1}$$

Besides send-timing point prediction, we also develop a *highway combination* mechanism to improve resources' utilization in AVCR process for response packets. As shown in Figure 3, if the destination of two request packets' reservation paths is the same, we combine the two highways in the next-hop router. By combination, two separate single-flit request packets will be injected into the same buffer in the next hop, sharing the buffer resources so that there is still at maximum one reserved buffer in each port. This mechanism is assured correctness by the fact that the destination NI will treat each flit in the highway as a complete request packet.

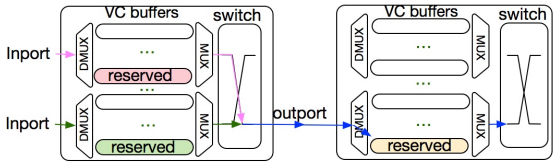


Fig. 3. An example of highway combination

IV. Implementation and Application

A. Implementation

1) Packet format: As shown in Table II, we add two tag bits in the flit to distinguish different scenarios. The tags '01' and '10' are used for AVCR response and request packets (including their corresponding reservation packets). Packets with tag '11' will release the highway, including reserved resources and records in corresponding routers and NIs.

TABLE II
Packet type and its corresponding tag

packet type	tag value	packet type	tag value
normal packets	00	AVCR response packet	01
AVCR request packet	10	AVCR release packet	11

2) Support AVCR in NI: As shown in Figure 4, we use an AVCR controller to record and control the AVCR process in the NI. The AVCR controller stores 6 parameters and has 4 functional units which are described below:

- Match information extractor (MIE): It extracts the *address* and *destination* information from the request packet

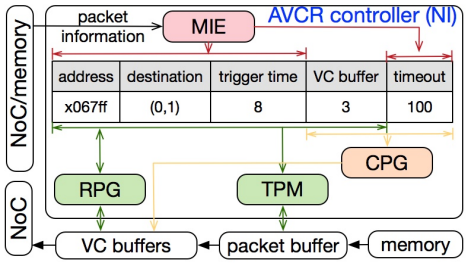


Fig. 4. AVCR controller in the NI

and records them as well as *trigger time* (calculated by Equation 1) and *timeout* (20 cycles for the cache and 100 cycles for the off-chip memory) parameters, which will reduce 1 per cycle.

- Reservation packet generator (RPG): When the *trigger time* reaches 0, it generates a reservation packet with the *destination* in record and sends it into the VC buffer. The *VC buffer* ID will then be recorded.
- Target packet monitor (TPM): It monitors *address* and *destination* parameters in received packets from memory and compares them with the corresponding recorded information to find the target packet which will be flitisized into the recorded VC. Then, the record will be erased.
- Clean packet generator (CPG): When the *timeout* reaches 0, it generates a clean packet ('TAIL' type) and sends it to the recorded corresponding destination to release resources in the highway. Then, the record will be erased.

Our mechanism is totally based on the original VC flow control mechanism except for one assumption that one reservation will block another reservation in the same virtual network, which may cause deadlock. The CPG assures that reserved resources will not be occupied permanently. They will be released either by the normal 'TAIL' flit or by the clean packet. So our mechanism is deadlock free.

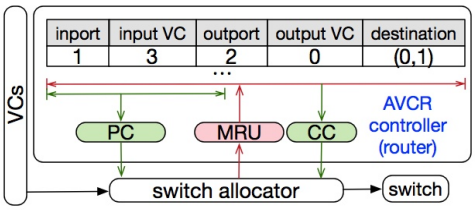


Fig. 5. AVCR controller in the router

3) Support AVCR in router: In Figure 5, we show the implementation of the AVCR controller in the router. It stores 5 kinds of parameters and has 3 functional units which are described as follows:

- Monitor and record unit (MRU): It monitors the switch allocator and records *inport, input VC, outport, output VC* and *destination* information for the highway. During the highway teardown process, it erases the record.
- Priority controller (PC): It controls the switch allocator to offer the highest priority for flits in the reserved VC.
- Combination controller (CC): For the request reservation packet, if it has the same *destination* as those in record, it will control the switch allocator to allocate the recorded *outport* and *output VC* to this reservation packet.

B. Application

AVCR can be applied to many scenarios. In this paper, we use it in two cases to speed up program runtime. Firstly, we apply it to response packets belonging to several slowest threads, which significantly dominate program's process, to reduce the packets response time of those threads and accelerate those threads' execution. We use the program-level information (loop count) similar to [16] [17] to predict threads progress. In the AVCR mechanism, some non-target packets will be delayed due to the resources reservation, which may delay the threads with fast progress. But under the fact that program execution is defined by the slowest thread, no side effect will be made on the program finish time . Secondly, we use AVCR to accelerate the lock request packets. Nowadays, spinning lock is a common solution for most system kernels to realize mutex lock and barrier in parallel applications. The algorithm for spinning lock is shown in Algorithm 1. In the first line it triggers a read request to the lock and in the forth line it triggers a write request to the lock. This process will repeat if the judgment process (the second and fifth lines) fails. In conjunction with the cache coherence protocol, it results in a repeatedly request process for the same lock parameter in the remote shared cache until the lock is available, perfectly fitting the request case in our AVCR mechanism.

Algorithm 1 Assembly code for lock spinning

```
1: Lock: LD R2, 0(R1)      ; Load of lock to R2
2: BENZ R2, lock           ; Lock unavailable, spin
3: ADD R2, R0, #1          ; Change lock value to 1
4: EXCH R2, 0(R1)          ; Swap
5: BENZ R2, Lock           ; Loop to line 1 if lock is 1
```

V. Experiments

A. Methodology

We evaluate AVCR with timing-detailed full-system simulation using PARSEC benchmarks [20]. For data validity, we only report results obtained from region-of-interest (ROI) execution in the experiments. We implement and integrate our design in GEM5 [21], in which the embedded network GARNET [22] is enhanced according to our technique. The key configurations of the simulation platform are shown in Table III. Each PARSEC program runs on 64 cores with one core running one thread. To show the benefits of AVCR in expediting multi-threaded programs, we implement a priority method in switch arbitration, which simply offers priority to those accelerated packets in the AVCR mechanism but without advance resources reservation. Besides, we also compare AVCR with two state-of-art related works, namely OCOR [18] and iNPG [19], which accelerate critical section access. [18] proposes a software-hardware cooperative mechanism that can opportunistically maximize the chance that a thread wins the critical section access in the low-overhead spinning phase, thereby reducing the competition overhead. In [19], the authors propose "big" routers which can generate in-network packets to reduce high cache coherence overhead.

TABLE III
SIMULATION PLATFORM CONFIGURATIONS

Item	Number	Description
Processor	64 cores	Alpha based 2GHz out-of-order cores. 32-entry instruction queue, 64-entry load and store queues, 128-entry reorder buffer.
L1-Cache	64 banks	Private, 32KB per-core, 4-way set associative, 128B block, 2-cycle latency, split I/D caches, 32 MSHRs.
L2-Cache	64 banks	Chip-wide shared, 1 MB per-bank, 16-way set associative, 128B block size, 6-cycle latency, 32 MSHRs.
Memory	64 ranks	4GB DRAM, 512 MB per-rank, up to 16 outstanding requests for each processor, 8 memory controllers.
NoC	64 nodes	8x8 mesh network. Each node consists of 1 router, 1 network interface, 1 core, 1 private L1 cache, and 1 shared L2 cache. X-Y dimensional order routing. **Router is 2-stage pipelined, 4 VCs per port, 4 flits per VC. 128-bit data path. Directory based MOESI cache coherence protocol. One cache block consists of 1 packet, which consists of 8 flits. One coherence control message consists of 1 single-flit packet.**
AVCR	-	By default, only 1 of 4 VCs per port is used for AVCR. The highest priority is given to packets in reserved VCs during switch arbitration in each NI and router. 16 entries recording table in each NI and router.
OCOR [18]	-	128 retry times in the spinning phase. 9 priority levels, 8 higher levels for requests in the spinning phase, each priority level is mapped to 16 retry time; 1 lowest priority level for wakeup requests.
iNPG [19]	-	32 normal routers and 32 big routers, where one big router is deployed between every two normal routers. 16 entries message recording table in each packet generator.

B. Experimental results

1) Packet delay reduction: As shown in Figure 6, we trace the first 200 packets which are accelerated by AVCR mechanism in each benchmark to illustrate the reduction of their contention delay with the application of AVCR mechanism. For comparison convenience, we normalize the latency of those packets obtained under the original mechanism to 100%. Each latency is divided into two parts, the ideal latency without contention T_i and the delay caused by contention T_c. Comparing the results with and without AVCR, the ideal latency part in each benchmark stays the same. The average value for the contention delay in the original mechanism is 37.0% and up to 52%. But with our mechanism, the contention delay is significantly reduced to 4.6% on average. In this context, the existing contention delay is caused in the case that the reservation 'HEAD' flit is caught up by the following 'BODY' flit.

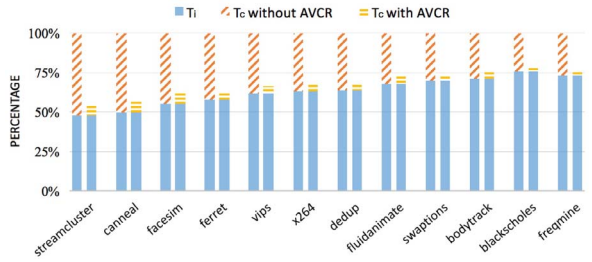


Fig. 6. Comparison of packet delay with and without AVCR

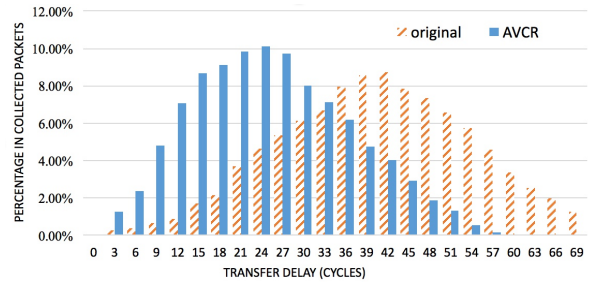


Fig. 7. Comparison of delay distribution for packets with and without AVCR

Taking *streamcluster* as example, Figure 7 depicts the distribution of occurred latency for those packets in detail. In the original mechanism, the average and maximum values are respectively 41 and 112 cycles. After the acceleration by the highway, those values are greatly decreased to 25 and 64 cycles. As all those packets are either from the slowest threads or from the lock related mutex phase, the decrease of delay will significantly improve the application performance.

2) Application performance: In Figure 8, the lock competition time averagely takes around 33.3% ROI time among 10 benchmarks, except for *swaptions* and *blockscholes* which do not have lock competition process. After applying the AVCR mechanism to lock request packets, the ROI finish time averagely reduces 7.3%, which is shown by 'ROI reduced (1)' in Figure 8. The figure also illustrates that the threads unbalance part (total maximum thread waiting time at barrier) takes about 32.5% ROI time among 12 benchmarks. By applying AVCR response-packet acceleration mechanism to slow threads (16 threads in our experiment), the ROI finish time reduces 18.2%, which is shown by 'ROI reduced (2)' in Figure 8.

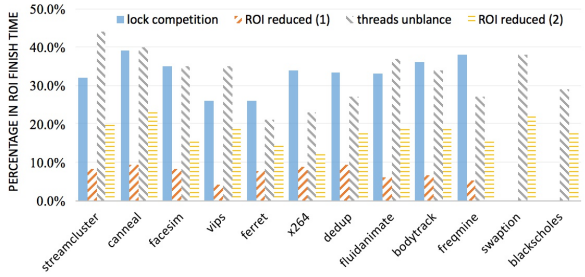


Fig. 8. Improving program runtime by AVCR

3) Performance comparison: We show the overall improvement and compare the results with previous work as well as the priority method in Figure 9. For comparison convenience, we normalized the original ROI finish time to 100%. On average, the AVCR mechanism reduces the ROI finish time by 22.4% and up to 29.4%. The results are better than iNGP (16% on average) and OCOR (10% on average) among those 12 benchmarks. For the priority method, the results show that it can only offer an average 5.6% gain in performance.

4) Scalability: Finally, we discuss the scalability of the AVCR mechanism. As the network size grows, more parallel threads can run on chip. This may greatly increase the traffic congestion and the packet transfer delay, offering more

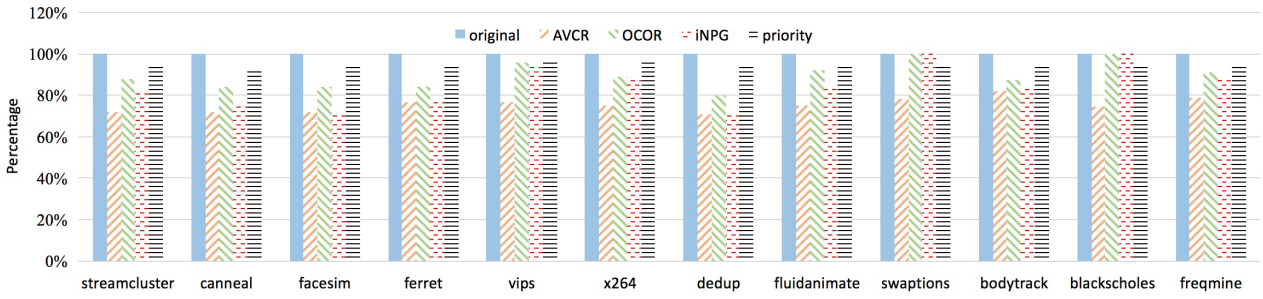


Fig. 9. Comparison of ROI finish time with different methods

opportunity for performance improvement. Figure 10 shows the reduction of ROI finish time among 12 benchmarks under three different network sizes with 2D mesh topology: 2×2, 4×4 and 8×8. The results show that the ROI finish time is only averagely reduced by 6.0% (maximally 7.1%) in the 2×2 mesh topology. But for 4×4 and 8×8 mesh topologies, the corresponding results increase to 13.3% and 22.4% on average. Our AVCR is beneficial for large network size and heavy network congestion.

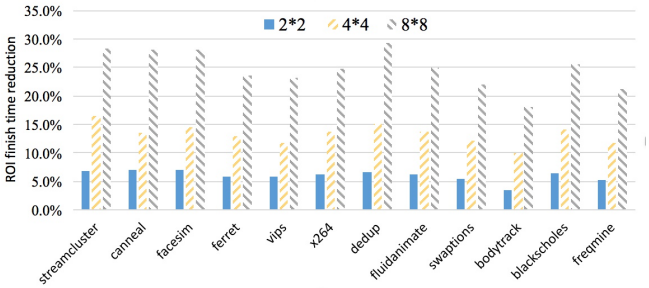


Fig. 10. Comparison of ROI finish time under different network sizes

VI. Conclusion

We have proposed a service mechanism named AVCR to reduce the latency of packets, which is based on the fact that the packet destination can often be known or predicted before the packet is ready to deliver in the NI. This mechanism establishes an end-to-end highway from source NI to destination NI to reduce the packet delay. We apply this mechanism to real applications, where the delay reduction of critical packets leads to the improvement of applications' performance. Full-system simulation results in GEM5 with the PARSEC benchmark show that target packets achieve averagely 32.4% latency reduction (up to 49.9%) and the average performance improvement across the benchmarks reaches 22.4% (up to 29.4%).

References

[1] A. Sharifi, E. Kultursay, M. Kandemir, and C. R. Das, "Addressing end-to-end memory access latency in NoC-based multicores," in *Proceedings of the 45th International Symposium on Microarchitecture*, 2012.

[2] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 8, pp. 1021–1034, 2008.

[3] S. Liu, Z. Lu, and A. Jantsch, "Highway in TDM NoCs," in *Proceedings of the 9th International Symposium on Networks-on-Chip (NOCS)*, 2015.

[4] S. Ma, N. E. Jerger, and Z. Wang, "Whole packet forwarding: Efficient design of fully adaptive routing algorithms for networks-on-chip," in *Proceedings of the 18th International Symposium on High Performance Computer Architecture (HPCA)*, 2012.

[5] X. Chen, Z. Lu, S. Liu, and S. Chen, "Round-trip DRAM access fairness in 3D NoC-based many-core systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 162, 2017.

[6] S.-Y. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on information theory*, vol. 49, no. 2, pp. 371–381, 2003.

[7] Y. Xue and P. Bogdan, "User cooperation network coding approach for NoC performance improvement," in *Proceedings of the 9th International Symposium on Networks-on-Chip (NOCS)*, 2015.

[8] A. Kumary, P. Kunduz, A. Singhx, L.-S. Pehy, and N. Jhay, "A 4.6 Tbits/s 3.6 GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS," in *Proceedings of the 25th International Conference on Computer Design (ICCD)*, 2007.

[9] L.-S. Peh and W. J. Dally, "Flit-reservation flow control," in *Proceedings of the 6th IEEE Symposium on High-Performance Computer Architecture (HPCA)*, 2000.

[10] M. Horchani, M. Atri, and R. Tourki, "Design of a NoC-router guaranteeing QoS based on virtual channels reservation," in *Proceedings of the 2nd International Conference on Signals, Circuits and Systems*, 2008.

[11] S. Evain and J.-P. Diguet, "Efficient space-time NoC path allocation based on mutual exclusion and pre-reservation," in *Proceedings of the 17th Great Lakes Symposium on VLSI (GLSVLSI)*, 2007.

[12] S. Volos, C. Seiculescu, B. Grot, N. K. Pour, B. Falsafi, and G. De Micheli, "CCNoC: Specializing on-chip interconnects for energy efficiency in cache-coherent servers," in *Processdings of the 6th International Symposium on Networks-on-Chip (NOCS)*, 2012.

[13] J. S. Miguel and N. E. Jerger, "Data criticality in network-on-chip design," in *Proceedings of the 9th International Symposium on Networks-on-Chip (NOCS)*, 2015.

[14] A. K. Abousamra, R. G. Melhem, and A. K. Jones, "Deja vu switching for multiplane NoCs," in *Proceedings of the 6th International Symposium on Networks-on-Chip (NOCS)*, 2012.

[15] D. J. Sorin, M. D. Hill, and D. A. Wood, "A primer on memory consistency and cache coherence," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 3, pp. 1–212, 2011.

[16] Q. Cai, J. González, R. Rakvic, G. Magklis, P. Chaparro, and A. González, "Meeting points: using thread criticality to adapt multicore hardware to parallel regions," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[17] S. M. Neuman, J. E. Miller, D. Sanchez, and S. Devadas, "Using application-level thread progress information to manage power and performance," in *Proceedings of the 35th IEEE International Conference on Computer Design (ICCD)*, 2017, pp. 501–508.

[18] Y. Yao and Z. Lu, "Opportunistic competition overhead reduction for expediting critical section in NoC based CMPs," *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[19] ——, "iNPG: Accelerating critical section access with in-network packet generation for NoC based many-cores," in *Proceedings of the 24th International Symposium on High-Performance Computer Architecture (HPCA)*, 2018.

[20] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[21] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM Transctions on SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[22] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.