ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.11, No.18, 1–10

Cooperative communication for efficient and scalable all-to-all barrier synchronization on mesh-based many-core NoCs

Xiaowen Chen[1,2a)], **Zhonghai Lu**[2], **Axel Jantsch**[2], **Shuming Chen**[1], **Yang Guo**[1], **and Hengzhu Liu**[1]

[1] *College of Computer, National University of Defense Technology, China*

[2] *Department of Electronic Systems, KTH-Royal Institute of Technology, Sweden*

a) *xiaowenc@kth.se*

Abstract: On many-core Network-on-Chips (NoCs), communication is on the critical path of system performance and contended synchronization requests may cause large performance penalty. Different from conventional algorithm-based approaches, the paper addresses the barrier synchronization problem from the angle of optimizing its communication performance and proposes cooperative communication as a means to achieve efficient and scalable all-to-all barrier synchronization on mesh-based many-core NoCs. With the cooperative communication, routers collaborate with one another to accomplish a fast barrier synchronization task. The cooperative communication is implemented in our router at low cost. Through comparative experiments, our approach evidently exhibits high efficiency and good scalability.

Keywords: cooperative communication, all-to-all barrier synchronization, many-core Network-on-Chips

Classification: Integrated circuits

References

[1] R. Marculescu, U. Y. Ogras, L. Peh, N. E. Jerger and Y. Hoskote: IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **28** (2009) 3. DOI:10.1109/TCAD.2008.2010691

[2] J. Hennessy and D. Patterson: *Computer Architecture: A Quantitative Approach* (Morgan Kaufmann, 2011) 5th ed.

[3] M. Monchiero, G. Palermo, C. Silvano and O. Villa: IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **14** (2006) 1049. DOI:10.1109/TVLSI.2006.884147

[4] A. Marongiu and L. Benini: CASES (2007) 145.

[5] T. Krishna, A. Kumar, L. Peh and J. Postman: IEEE Micro **29** [4] (2009) 48. DOI:10.1109/MM.2009.64

[6] J. L. Abellan, J. Fernandez and M. E. Acacio: IEEE Trans. Parallel Distrib. Syst. **23** (2012) 1453. DOI:10.1109/TPDS.2011.304

[7] B. Wilkinson: *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers* (Prentice Hall, New Jersey, 2004).

DOI: 10.1587/elex.11.20140542
Received June 6, 2014
Accepted August 13, 2014
Publicized September 3, 2014
Copyedited September 25, 2014

[8] O. Villa, G. Palermo and C. Silvano: CASES (2008) 81.
[9] J. M. Mellor-Crummey and M. Scott: ACM Trans. Comput. Syst. **9** (1991) 21. DOI:10.1145/103727.103729
[10] E. D. Brooks: Int. J. Parallel Program. **15** (1986) 295. DOI:10.1007/BF01407877
[11] S. Ma, N. E. Jerger and Z. Wang: HPCA (2012) 1. DOI:10.1109/HPCA.2012.6169049

1 Introduction and related work

While Network-on-Chip (NoC) [1] provides scalable bandwidth, it increases communication distance between communicating nodes. As a consequence, communication latency is negatively impacted due to longer path and possibly more contention. This brings a performance challenge for parallelized programs, which rely on efficient barrier synchronization to achieve high performance. Barrier synchronization is a classic problem that has been extensively studied in the context of parallel machines [2]. It should be carefully designed to achieve low latency communication and to minimize overall completion time. Towards single-chip systems, to speed up barrier synchronization in MPSoCs, Monchiero proposed a centralized hardware approach based on the master-slave algorithm [3]. Due to the centralized nature and non-availability of support for efficient communication, this proposal performs well only for less than 10 cores. In [4], Marongiu discussed the use of a run-time lightweight barrier construct in non-cache coherent MPSoCs. However, he fell short of exploiting efficient communication, harvesting only limited scalability. Targeting many-core CMPs, based on *G-line* technology [5], Abellan deployed a dedicated network to allow for fast and efficient signaling of barrier arrival and departure [6]. Although a dedicated network can achieve a better barrier synchronization performance, extra links and routing and arbitration logics are required, since the links and routers in the original on-chip network are not fully utilized.

Conventional approaches for addressing the barrier synchronization problem have been algorithm oriented. There are four main classes of algorithms: *master-slave* [7], *all-to-all* [8], *tree-based* [7, 9], *butterfly* [10]. Among them, the all-to-all algorithm takes a distributed solution. It assumes that each node keeps a local copy of the global barrier counter. Each barrier acquire request is broadcasted to all nodes to increment their own local barrier counters. Each node is released locally when all nodes reach the local barrier. Although this eliminates the barrier release overhead, but an increased number of broadcasted barrier acquire requests incur larger barrier acquire overhead. Therefore, the all-to-all algorithm is only suitable to small-scale systems and its performance becomes worse in large-scale systems.

It is a trend that many cores are networked in a single chip (named, many-core NoCs), so that communication is on the critical path of system performance and contended synchronization requests may cause large performance penalty. Motivated by this, different from the conventional algorithm-based optimizations mentioned above, the paper addresses performance optimization of barrier synchronization on mesh-based many-core NoCs from the angle of exploiting its

DOI: 10.1587/elex.11.20140542
Received June 6, 2014
Accepted August 13, 2014
Publicized September 3, 2014
Copyedited September 25, 2014

efficient communication. The cooperative communication is proposed. It is orthogonal to the all-to-all algorithm, i.e. as a means, it is combined with the all-to-all algorithm in order to achieve efficient and scalable all-to-all barrier synchronization on mesh-based many-core NoCs. The proposed cooperative communication is a kind of collective communication [11], which efficiently implements the gather communication for barrier acquire requests. It is called 'cooperative' since all routers collaborate with one another to accomplish a fast all-to-all barrier synchronization task. For instance, with the proposed cooperative communication, multiple barrier acquire packets can be merged into a single barrier acquire packet in a router if they aim for the same barrier and arrive at the router simultaneously, thus resulting in significant reduction of network workload, which shrinks the completion time. The complexity of implementing the cooperative communication in the router is low. Synthetic and application experiments show that our approach can significantly reduce synchronization completion time and increase application speedups.

2 Cooperative communication for scalable all-to-all barrier synchronization

2.1 Mesh-based many-core NoC

Due to its regularity, simplicity and modularity, the mesh network has been a popular topological option for NoC designs. We consider a regular mesh architecture for our many-core NoC. Fig. 1a shows a 3×3 example. Each processing core, **P**, is connected to a router, **R**. Routers are interconnected with bidirectional links. The mesh network we used is packet-switched, performs dimension-order XY routing, provides best-effort service and also guarantees in-order packet delivery. Besides, moving one hop in the network takes one cycle.

2.2 Cooperative communication

The idea of cooperative communication is to transfer barrier acquire packets to all nodes as fast as possible by (1) distributing barrier acquire packets to all nodes firstly along the east and west directions and secondly along the south and north directions as well as (2) merging multiple barrier acquire packets that aim for the same barrier into a single barrier acquire packet.

We exemplify this cooperative action. Fig. 2 shows a 3×3 mesh where all nodes aim for the same barrier. At cycle t (see Fig. 2a), all nodes send a barrier acquire request encapsulated by a barrier acquire packet to other nodes. At this time, the local barrier counter of each node is set to be '1' by itself. At cycle $t + 1$ (see Fig. 2b), an intermediate node may receive multiple barrier acquire packets. It firstly replicates and distributes them into different outports depending on their incoming ports. Then, for those barrier acquire packets to the same outport, if they target the same barrier, they are merged into a single barrier acquire packet. For instance, at cycle t, the three barrier acquire packets (marked with "**A,1**", "**C,1**" and "**E,1**") from node **A**, **C** and **E** go into the west, east and south port of router **B** respectively. At cycle $t + 1$, router **B** receives these three packets. According to our replication algorithm in Section 2.3, in router **B**, packet "**A,1**" is replicated to

DOI: 10.1587/elex.11.20140542
Received June 6, 2014
Accepted August 13, 2014
Publicized September 3, 2014
Copyedited September 25, 2014

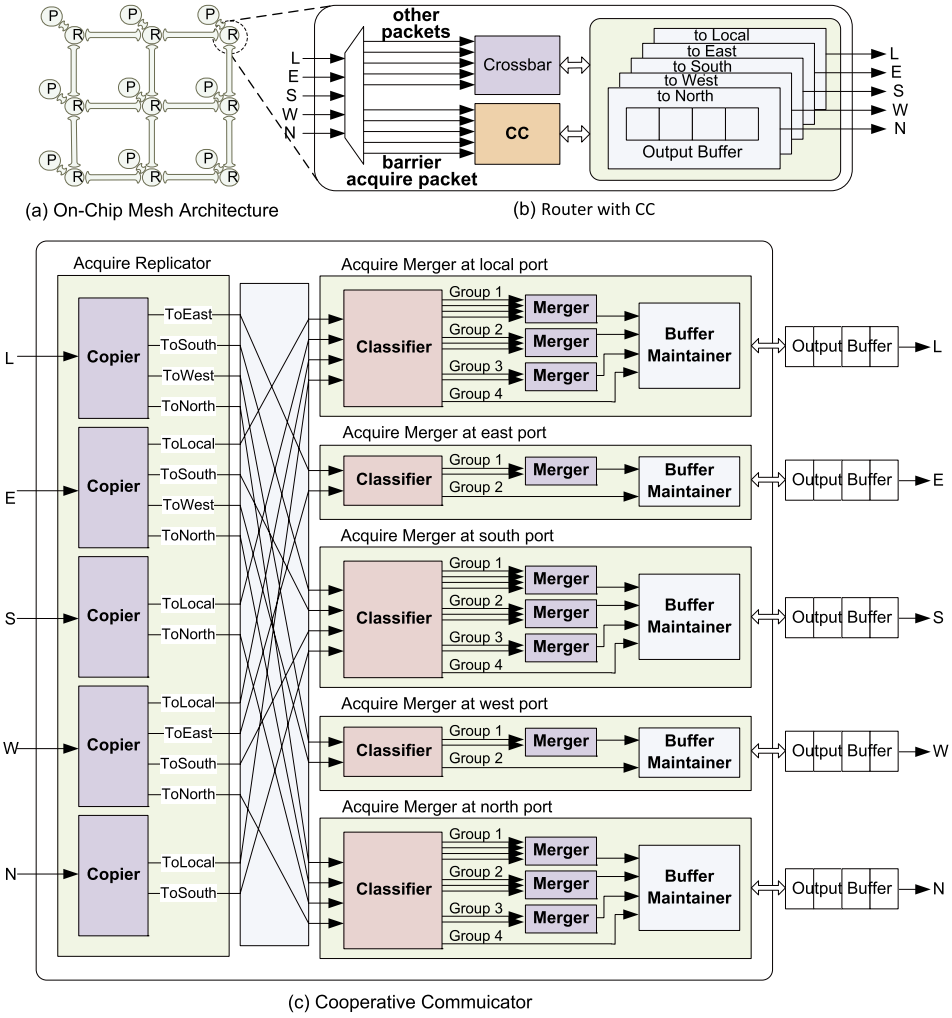

Fig. 1. A 3×3 mesh architecture with routers enhanced by Cooperative Communicators (CCs)

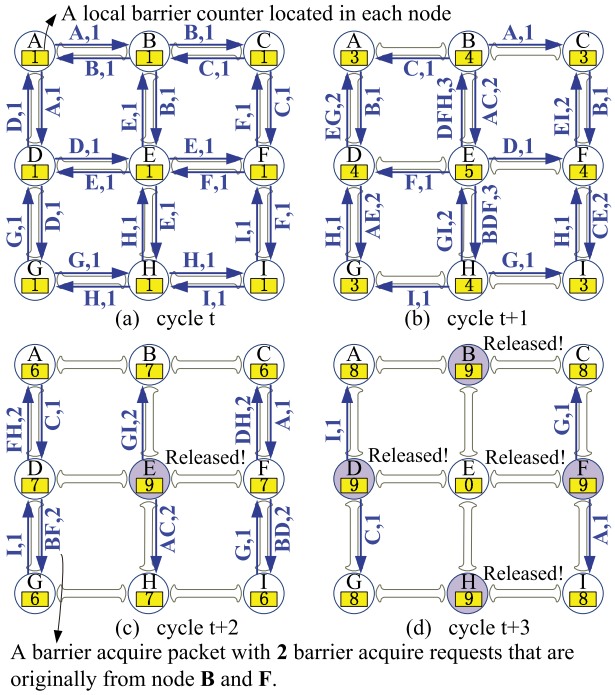

Fig. 2. Cooperative barrier communication

DOI: 10.1587/elex.11.20140542
Received June 6, 2014
Accepted August 13, 2014
Publicized September 3, 2014
Copyedited September 25, 2014

generate three packets: one is forwarded to node **B** to increment the local barrier counter and the other two are forwarded to the east and south outports respectively. Packet "**C,1**" is also replicated to generate three packets: one to increment the counter in node **B** and the other two to the west and south outports. Packet "**E,1**" only generates one packet to increment node **B**'s barrier counter. Thus, as shown in Fig. 2b, the local barrier counter in node **B** is updated as 4. Since packet "**A,1**" and "**C,1**" both have a copy to the south outport, the two copies are merged into one packet (marked with "**AC,2**") containing 2 barrier acquire requests. Further on, at cycle t + 2, more packets are merged and node **E** is first released since all barrier requests reach node **E**'s local barrier. Then, node **B**, **D**, **F** and **H** are released at cycle t + 3. At last, the barrier synchronization is completed after node **A**, **C**, **G** and **I** are released at cycle t + 4 (not drawn in the figure). In total, this process takes 5 cycles, transmitting 56 (= 24 + 18 + 10 + 4) packets. If unicast transmission is employed instead, it takes 16 cycles transmitting 144 packets (calculated in the experiment). Such cooperative action not only avoids serialization of packet transmission over shared links but also reduces workload.

2.3 Cooperative Communicator (CC)

To realize the cooperative communication, the router is enhanced with a *Cooperative Communicator* (CC). As shown in Fig. 1c, the CC consists of six functional units: an *Acquire Replicator* (AR) and five *Acquire Mergers* (AMs).

As depicted in Fig. 1c, the AR is responsible for replicating and distributing incoming barrier acquire packets to different outports. There are 5 copiers, one for each inport. To avoid redundant packet replication, the AR replicates a barrier acquire packet depending on the incoming direction of the barrier acquire packet. To facilitate the explanation, we use notations: $a(id, rn, in)$ represents a barrier acquire packet with a barrier id, rn count of barrier acquire requests, and its incoming port in, which can be L (Local), N (North), S (South), E (East), and W (West); notation $r(id, rn, in, out)$ represents a replicated barrier acquire packet with barrier id and rn count of barrier acquire requests from inport in to outport out. The acquire replicating algorithm acts according to the following formulas. (a)–(e) correspond to the 5 copiers in Fig. 1c respectively. Depending on the incoming port, the algorithm replicates the barrier acquire packet to different ports. Take (a) as an example. When a router receives a barrier acquire packet from the local port, it shall replicate four barrier acquire packets to four directions: E, S, W and N, one for each.

$$(a)\quad a(id_L, rn_L, L) \Rightarrow \begin{cases} r(id_L, rn_L, L, E) \\ r(id_L, rn_L, L, S) \\ r(id_L, rn_L, L, W) \\ r(id_L, rn_L, L, N) \end{cases}$$

$$(b)\quad a(id_E, rn_E, E) \Rightarrow \begin{cases} r(id_E, rn_E, E, L) \\ r(id_E, rn_E, E, S) \\ r(id_E, rn_E, E, W) \\ r(id_E, rn_E, E, N) \end{cases}$$

DOI: 10.1587/elex.11.20140542
Received June 6, 2014
Accepted August 13, 2014
Publicized September 3, 2014
Copyedited September 25, 2014

$$(c)\quad a(id_W, rn_W, W) \Rightarrow \begin{cases} r(id_W, rn_W, W, L) \\ r(id_W, rn_W, W, E) \\ r(id_W, rn_W, W, S) \\ r(id_W, rn_W, W, N) \end{cases}$$

$$(d)\quad a(id_S, rn_S, S) \Rightarrow \begin{cases} r(id_S, rn_S, S, L) \\ r(id_S, rn_S, S, N) \end{cases}$$

$$(e)\quad a(id_N, rn_N, N) \Rightarrow \begin{cases} r(id_N, rn_N, N, L) \\ r(id_N, rn_N, N, S) \end{cases}$$

For each outport, there is an AM. As shown in Fig. 1c, the AM is responsible for checking replicated barrier acquire packets from the AR and those barrier acquire packets that has been stored in the output buffer and then merging the barrier acquire packets aiming for the same barrier counter into one barrier acquire packet. Take the AM at the local outport as an example. Its function contains three steps. (1) The Classifier groups barrier acquire packets from the AR into several groups according to their barrier *id*. As there are 4 inputs, up to 4 barrier acquire packets may target the same barrier. Incoming barrier acquire packets may be classified into up to 4 groups, since they may aim for 4 different barriers, one group for one barrier. (2) A Merger merges a group of barrier acquire packets into one barrier acquire packet. It extracts all values in "ReqNum" field [1] of these packets, adds them together, and puts the sum into the "ReqNum" field of the merged barrier acquire packet. (3) If in the output buffer there is a stored barrier acquire packet that has the same barrier *id* with the merged barrier acquire packet from the Merger, the Buffer Maintainer adds the "ReqNum" of the merged barrier acquire packet into that of the stored barrier acquire packet; if not, the Buffer Maintainer puts the merged barrier acquire packet into the tail of the output buffer. To facilitate the explanation, notation $m(id, rn, out)$ represents a merged barrier acquire packet with barrier *id* and *rn* count of barrier acquire requests to outport *out*. The acquire merging algorithm for the AM at local outport is sketched as follows. (a)–(d) correspond to group 1–4 in Fig. 1c respectively. Take (b) as an example. Barrier acquire packets, which have different *id* with the barrier acquire packet from the east inport to the local outport and have the same *id* with the barrier acquire packet from the south inport to the local outport, are merged into one barrier acquire packet, which is then forwarded to the local outport of the router. The sum of their *rn* forms the *rn* in the merged packet.

$$(a)\quad \left.\begin{array}{l} r(id_E, rn_E, E, L) \\ r(id_S, rn_S, S, L) \\ r(id_W, rn_W, W, L) \\ r(id_N, rn_N, N, L) \end{array}\right\} \xrightarrow[id_i = id_E]{i \in \{E,S,W,N\}} m(id_E, \sum rn_i, L)$$

$$(b)\quad \left.\begin{array}{l} r(id_S, rn_S, S, L) \\ r(id_W, rn_W, W, L) \\ r(id_N, rn_N, N, L) \end{array}\right\} \xrightarrow[id_i \neq id_E, id_i = id_S]{i \in \{S,W,N\}} m(id_S, \sum rn_i, L)$$

[1]The barrier acquire packet has a "*ReqNum*" field that denotes how many barrier acquire requests are included in this packet. Initially, when a barrier acquire packet is issued by a node, its "*ReqNum*" is equal to 1. This field is updated upon merging.

DOI: 10.1587/elex.11.20140542
Received June 6, 2014
Accepted August 13, 2014
Publicized September 3, 2014
Copyedited September 25, 2014

$$(c)\quad \left.\begin{matrix} r(id_W, rn_W, W, L) \\ r(id_N, rn_N, N, L) \end{matrix}\right\} \xrightarrow[id_i \neq id_S, id_i = id_W]{i \in \{W,N\}, id_i \neq id_E} m(id_W, \sum rn_i, L)$$

$$(d)\quad r(id_N, rn_N, N) \xrightarrow[id_N \neq id_S, id_N \neq id_W]{id_N \neq id_E} m(id_N, rn_N, L)$$

2.4 Hardware implementation

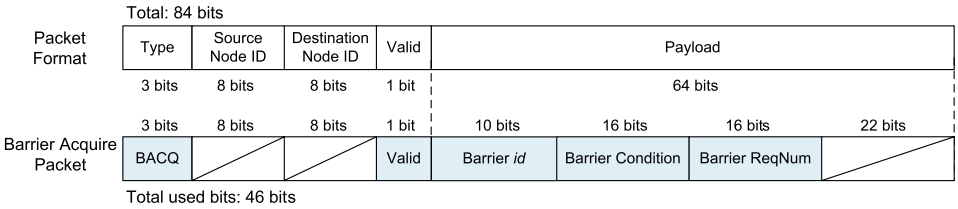

Fig. 3. Barrier acquire packet format

Fig. 3 gives the barrier acquire format. As it shows, the number of the used bits of the barrier acquire packet is 46, while the number of the total bits of the general packet is 84.

The CC design is synthesized in Synopsys® Design Compiler with TSMC® 65 nm process. Table I lists the logic synthesis results excluding the wire cost. For comparison, the router synthesis result is also given in the table. Since the barrier acquire packet only has 46 used bits and the CC is pure combinational logic, the CC only consumes 4.67k NAND gates and can run at 1.79 GHz (0.56 ns). The original router (Crossbar and Output Buffers) runs at 1.61 GHz (0.62 ns). When integrated into our router, since the CC in parallel with the Crossbar, it does not degrade the router's frequency.

Table I. Logic synthesis results of the router with the CC

	Area	Frequency
CC	13444.32 μm^2 (4.67k NAND gates)	1.79 GHz (0.56 ns)
Crossbar	36288.22 μm^2 (12.60k NAND gates)	1.61 GHz (0.62 ns)
Output Buffers	40670.59 μm^2 (13.12k NAND gates)	

Note: The area of a NAND gate is 2.88 μm^2.

3 Experiments and results

3.1 Experimental setup

The purpose of experiments is to investigate the performance gain of our proposal (all-to-all algorithm with cooperative communication, denoted by A2A+CC) in both efficiency and scalability. We compare our approach with the four algorithm-based mechanisms with unicast communication, namely, all-to-all algorithm with unicast (A2A+Un), master-slave algorithm with unicast (MS+Un), tree-based algorithm with unicast (Tree+Un), and butterfly algorithm with unicast (Butterfly+Un). We constructed a RT-level mesh-based many-core simulation platform as described in Section 2.1. Synthetic experiments and application benchmarks are performed on the platform with a variety of mesh ($M \times N$) sizes up to 256 nodes ($M = N = 16$).

DOI: 10.1587/elex.11.20140542
Received June 6, 2014
Accepted August 13, 2014
Publicized September 3, 2014
Copyedited September 25, 2014

3.2 Synthetic experiments

Case 1: The network size is varied and there is no other background traffic.

In this set of experiments, we discuss the pure performance of barrier synchronization on mesh-based many-core NoCs. All nodes participate in the barrier synchronization and there is no other background traffic. Node i sends a barrier acquire request after D_i cycles delay[2] when an experiment starts, and an experiment finishes when the release reaches all nodes. The completion time is measured from the starting till then. Fig. 4a–c plots the completion time versus the network size. Fig. 4a is with $MaxD = 0$, meaning that all nodes send barrier acquire requests at the same time. This setting is helpful to probe into the intrinsic performance of different barrier synchronization algorithms. From Fig. 4a–c, We can see that:

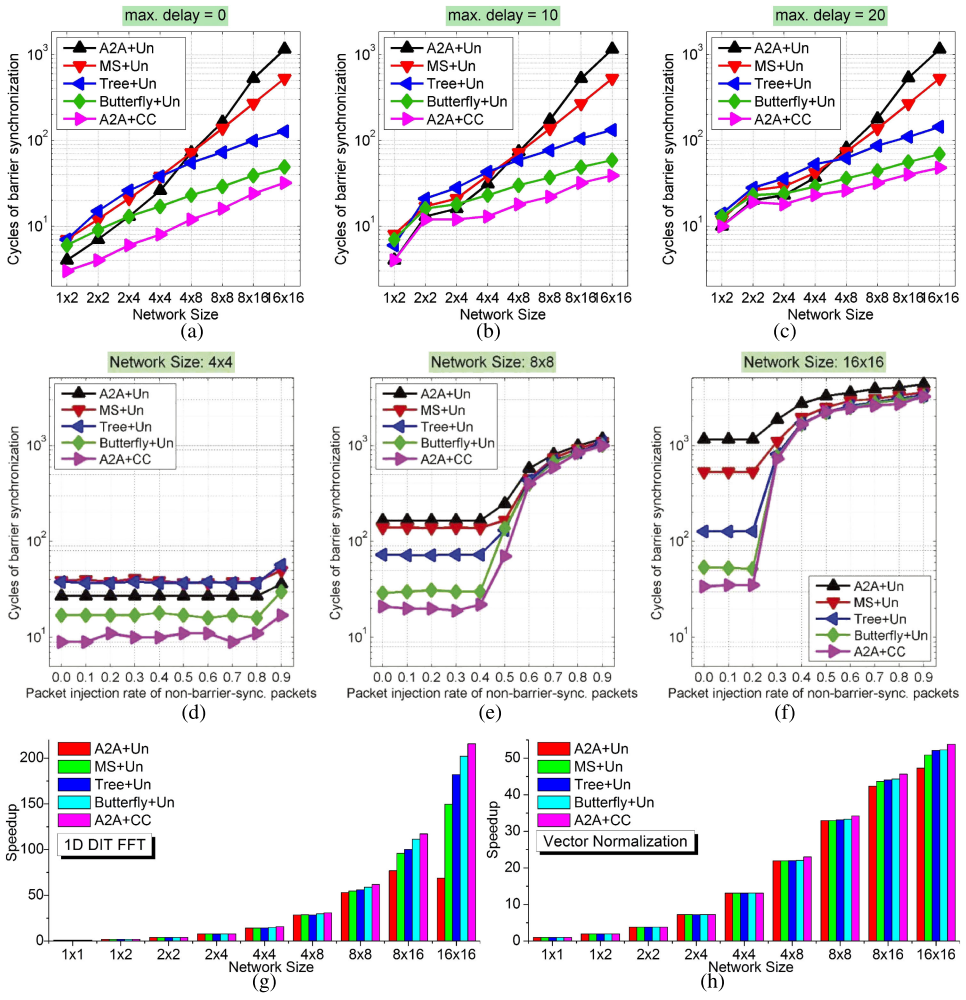

Fig. 4. (a)–(c): Completion time versus network size; (d)–(f): Completion time versus packet injection rate of non-barrier-sync. packets; (g): Speedup results of 1D DIT FFT; (h): Speedup results of Vector Normalization

[2]In experiments, we set a maximal delay: $MaxD$. For node i, its delay D_i is a random integer between 0 and $MaxD$.

DOI: 10.1587/elex.11.20140542
Received June 6, 2014
Accepted August 13, 2014
Publicized September 3, 2014
Copyedited September 25, 2014

- For all network sizes, our approach (A2A+CC) achieves minimal completion time. Due to specialized cooperative communication, there is no contention in the network. Consequently, the completion time can be theoretically determined as $M + N + max\{D_i - D_j\}$, which matches the simulation results.
- A2A+Un shows better performance than MS+Un, Tree+Un, and Butterfly+Un for networks of small sizes, but it does not scale well due to quadratically increased number of packets $(MN)^2$. With the help of our cooperative communication, the all-to-all algorithm (see A2A+CC) becomes efficient and scalable and its completion time increases very slowly as the network size is scaled up.
- The completion time of Tree+Un also increases slowly due to its alleviated network contention. But from 8 cores upward, Tree+Un is 3 to 5 times worse than A2A+CC due to increased non-contentional delay since the barrier synchronization event has to move up and down the entire logical tree.
- Among algorithm-based schemes, Butterfly+Un shows the best performance and scalability. Still, our A2A+CC is outstanding. For the experiments, our A2A+CC reduces the completion time of Butterfly+Un by 47% on average.

Case 2: The network size is fixed and there exists other background traffic.

In this set of experiments, we discuss the performance of barrier synchronization when there are other packets traversing in the on-chip network. We consider three network sizes: 4×4, 8×8 and 16×16. When an experiment starts, all nodes generate non-barrier-synchronization packets periodically to destination nodes, which are selected randomly. When the network load becomes stable[3], all nodes send barrier acquire requests and an experiment finishes when the release reaches all nodes. The completion time is measured from the starting till then. Fig. 4d–f plots the completion time versus the packet injection rate of non-barrier-synchronization packets under the network size of 4×4, 8×8 and 16×16, respectively. We can see that:

- When the network load is not heavy (e.g. in Fig. 4e, the packet injection rate of non-barrier-synchronization packets is from 0.0 to 0.4), the effect of other network traffic on the performance of barrier synchronization is trivial, since the network is able to deliver all generated packets timely. The performance of A2A+CC is better than that of the four algorithm-based schemes when the packet injection rate is from 0.0 to 0.4.
- As the network is overloaded (e.g. in Fig. 4e, the packet injection rate of non-barrier-synchronization packets is from 0.5 to 0.9), a number of packets are blocked in the network, resulting in heavy network contention. Under these circumstances, A2A+CC still has the fewest cycles. However, which solution is better becomes insignificant, since the huge network delay dominates.

3.3 Application benchmarks

In this set of experiments, two real applications, namely, 1024-point 1D DIT FFT and 256-element Vector Normalization (VN), are realized. Computational tasks are uniformly mapped onto all nodes. Depending on the mesh size ($M \times N$), each node

[3]In the experiments, the network load is considered to be stable after the warmup phase when each node has generated 1000 packets.

DOI: 10.1587/elex.11.20140542
Received June 6, 2014
Accepted August 13, 2014
Publicized September 3, 2014
Copyedited September 25, 2014

assumes J/MN tasks, where J is the total number of tasks. For the FFT, J is 1024 there are 9 ($log_2 1024 - 1$) times of barrier synchronization. For the VN, J is 256 and there are 2 times of barrier synchronization. We investigate the application speedup with respect to the number of cores from 1 up to 256.

Fig. 4g and Fig. 4h show the speedup (Ω_m)[4] results, which exhibit the same performance trend as Fig. 4a–c and clearly show the performance gain with A2A+CC. Note that, due to overwhelming contention, A2A+Un's speedup for 16×16 decreases. For the 16×16 case, compared with A2A+Un, MS+Un, Tree+Un, and Butterfly+Un, the respective performance improvement for the FFT is 68.13%, 30.67%, 15.58%, and 6.33%, and for the VN is 12.21%, 5.54%, 3.21%, and 2.92%. As expected, the speedup improvement for FFT is higher than that for VN. The reason is that the barrier synchronization related communication task of FFT takes a larger portion of time, and thus the optimization in communication enhances performance more significantly. Since the FFT and the VN have computation tasks besides synchronization tasks, the improvement is less than that from synthetic experiments.

4 Conclusion and future work

On-chip network in many-core NoCs brings not only challenge but also opportunity to realize efficient and scalable synchronization for parallel programs running on different cores. In this paper, we have exploited this potential in a regular mesh-based many-core NoCs. We addressed the barrier synchronization problem by optimizing its communication performance, since communication is on the critical path of system performance on mesh-based many-core architectures and contended synchronization requests may cause large performance penalty. The main idea is to propose the cooperative communication and combine it with the all-to-all algorithm so as to achieve high efficiency and scalability of all-to-all barrier synchronization on mesh-based many-core NoCs. The cooperative communication is implemented in the router at low cost. Comparative experiments prove that, with the proposed cooperative communication, the all-to-all algorithm is pushed from the worst to the best solution on mesh-based many-core NoCs.

In the future, we plan to link our approach on other irregular and regular NoC topologies.

Acknowledgments

The research is partially supported by the National Natural Science Foundation of China (No. 61133007 and No. 61402500) and the Doctoral Program of the Ministry of Education in China (No. 20134307120034 and No. 20134307110019).

DOI: 10.1587/elex.11.20140542
Received June 6, 2014
Accepted August 13, 2014
Publicized September 3, 2014
Copyedited September 25, 2014

[4] $\Omega_m = T_{1node}/T_{mnode}$, where T_{1node} is the single node execution time as the baseline and T_{mnode} is the execution time of m node(s).