ELectronics EXpress

IEICE Electronics Express, Vol.8, No.22, 1856–1862

Cooperative communication based barrier synchronization in on-chip mesh architectures

Xiaowen Chen[1,2ab)], **Zhonghai Lu**[2], **Axel Jantsch**[2], **Shuming Chen**[1], **and Hai Liu**[1]

[1] *National University of Defense Technology, 410073, Changsha, China*

[2] *KTH - Royal Institute of Technology, 16440 Kista, Stockholm, Sweden*

a) *xiaowenc@kth.se*

b) *xwchen@nudt.edu.cn*

Abstract: We propose cooperative communication as a means to enable efficient and scalable barrier synchronization on mesh-based many-core architectures. Our approach is different from but orthogonal to conventional algorithm-based optimizations. It relies on collaborating routers to provide efficient gather and multicast communication. In conjunction with a master-slave algorithm, it exploits the mesh regularity to achieve efficiency. The gather and multicast functions have been implemented in our router. Synthesis results suggest marginal area overhead. With synthetic and benchmark experiments, we show that our approach significantly reduces synchronization completion time and increases speedup.

Keywords: cooperative communication, barrier synchronization

Classification: Other communication hardwares

References

[1] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach, 4th Edition*, Morgan Kaufmann Publishers, 2007.

[2] T. Hoefler, et al., "A Survey of Barrier Algorithms for Coarse Grained Supercomputers," *Chemnitzer Informatik Berichte*, vol. 4, no. 3, Dec. 2004.

[3] B. Wilkinson, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall, 2004.

[4] O. Villa, et al., "Efficiency and Scalability of Barrier Synchronization on NoC Based Many-core Architectures," *Proc. Int'l Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES'08)*, pp. 81–89, 2008.

[5] J. Mellor-Crummey and M. Scott, "Algorithms for Scalable Synchronization on Shared-memory Multiprocessors," *ACM Trans. Computer Systems*, vol. 9, no. 1, pp. 21–65, Jan. 1991.

[6] E. D. Brooks, "The Butterfly Barrier," *Int'l J. of Parallel Programming*, vol. 15, pp. 295–307, Oct. 1986.

[7] M. Monchiero, et al., "Efficient Synchronization for Embedded On-chip Multiprocessors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*,

DOI: 10.1587/elex.8.1856
Received August 04, 2011
Accepted October 15, 2011
Published November 25, 2011

vol. 14, no. 10, pp. 1049–1062, Oct. 2005.
[8] A. Marongiu, et al., "Lightweight Barrier-based Parallelization Support for non-cache-coherent MPSoC Platforms," *Proc. Int'l Conf. on Compilers, Architectures, and Synthesis for Embedded Systems (CASES'05)*, pp. 145–149, 2007.
[9] P. McKinley, et al., "Collective Communication in Wormhole-routed Massively Parallel Computers," *IEEE Computer*, vol. 28, no. 12, pp. 39–50, Oct. 1995.
[10] Q. Ali, et al., "Modeling Advanced Collective Communication Algorithms on Cell-based Systems," *Proc. ACM Symp. on Principles and Practice of Parallel Programming (PPoPP'10)*, pp. 293–304, Jan. 2010.

1 Introduction and related work

Barrier synchronization is a classic problem which has been extensively studied in the context of parallel machines [1, 2]. It should be carefully designed to achieve low latency communication and to minimize overall completion time. Conventional approaches for addressing the barrier synchronization problem have been algorithm oriented. There are four main classes of algorithms: *master-slave* [3], *all-to-all* [4], *tree-based* [3, 5], and *butterfly* [6]. Recently, as a single chip is being able to integrate many cores, barrier synchronization becomes a critical concern in single-chip systems due to its impact on application performance. To speed up barrier synchronization in MPSoCs, Monchiero et al. proposed a centralized hardware approach [7] based on the master-slave algorithm. Due to the centralized nature and non-availability of support for collective communication, this proposal performs well only for less than 10 cores. In [8], Marongiu et al. discussed the use of a runtime lightweight barrier construct in non-cache coherent MPSoCs. However, they fell short of exploiting efficient communication, harvesting only limited scalability.

As many cores are networked in a single chip, communication is on the critical path of system performance and contended synchronization requests may cause large performance penalty. Motivated by this, this paper chooses another direction (i.e. exploiting efficient communication) to address the barrier synchronization problem. Barrier synchronization typically involves two kinds of collective communication [9, 10], namely, *gather* for barrier acquire and *multicast* for barrier release. In this paper, we propose the cooperative gather and multicast communication as a means to achieve efficient and scalable barrier synchronization. It is cooperative since all participating routers collaborate with each other to accomplish a barrier synchronization task. With the cooperative gather communication, multiple barrier acquire packets for the same barrier can be merged into a single barrier acquire packet when they pass through a router. With the cooperative multicast communication, barrier release packets are replicated at intermediate nodes depending on the incoming ports that barrier acquire packets went through before. Therefore, a multi-destination header is not required and wiring cost is saved. Though

DOI: 10.1587/elex.8.1856
Received August 04, 2011
Accepted October 15, 2011
Published November 25, 2011

different, our approach is orthogonal to the algorithm-based approaches. We combine it with the master-slave algorithm to show its effectiveness. Through experiments on synthetic and benchmark cases, we show that our approach greatly reduces barrier synchronization time and improves speedup for different scales of mesh networks.

2 Cooperative barrier communication

2.1 Mesh-based many-core architecture

We consider a regular mesh architecture for our many-core system. Fig. 1 a shows a 3×3 example. Each processing core, **P**, is connected to a router, **R**. Routers are interconnected with bidirectional links. The mesh network 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 gather and multicast

The idea of cooperative barrier gathering is to merge multiple barrier acquire packets from slave nodes into one barrier acquire packet at intermediate routers when they traverse in the network. We exemplify this packet merging action. Fig. 2 a shows a 4×4 mesh on which node $(2, 2)$ is the master node and all others are slave nodes, targeting the same barrier. At cycle t, all nodes

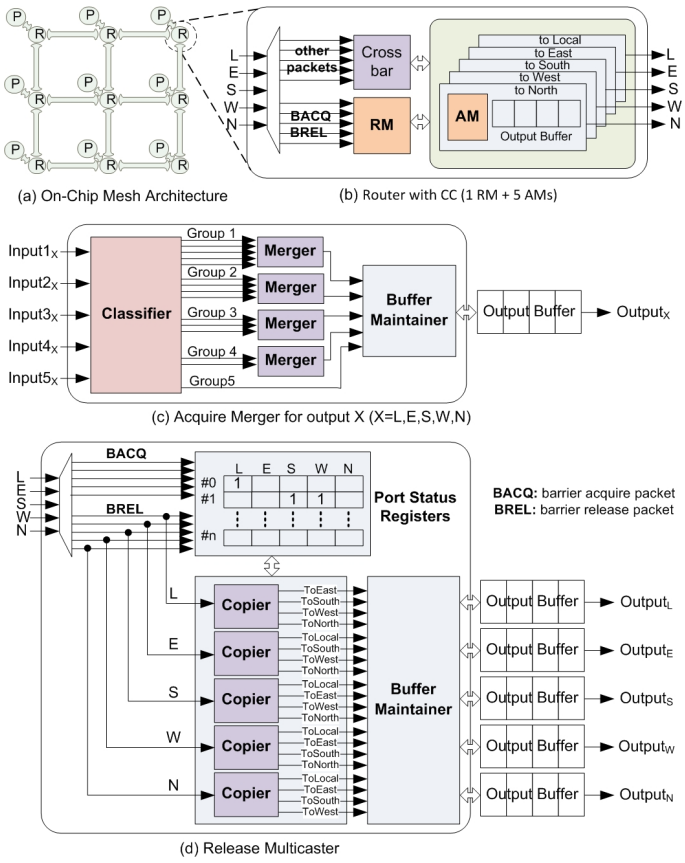

Fig. 1. A 3×3 mesh architecture with a router enhanced by a cooperative communicator (CC)

DOI: 10.1587/elex.8.1856
Received August 04, 2011
Accepted October 15, 2011
Published November 25, 2011

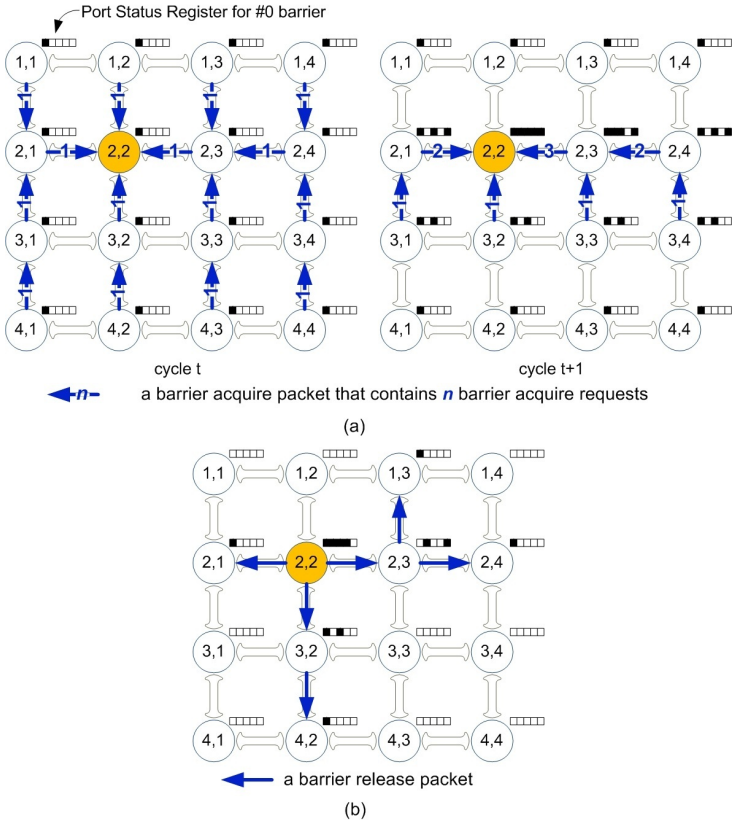

Fig. 2. (a) Barrier acquire packet merging and (b) barrier release packet multicasting

send a barrier acquire request encapsulated by a barrier acquire packet to the master node, as shown in the left picture of Fig. 2 a. At cycle t + 1, an intermediate node may receive multiple barrier acquire packets. As they target the same barrier, these packets are merged into a single packet. For instance, the barrier acquire packets from nodes $(1, 3)$, $(2, 4)$ and $(3, 3)$ reach node $(2, 3)$ (see the left picture of Fig. 2 a), and are then merged into one packet (see the right picture of Fig. 2 a). Such merging action not only avoids serialization of packet transmission over shared links but also reduces workload.

The idea of cooperative barrier multicasting is to replicate and distribute barrier release packets to all nodes as fast as possible. At barrier acquire stage of barrier synchronization, when barrier acquire packets pass through a router, their incoming ports are recorded. Such incoming port information in all routers constructs the multicast path for multicasting barrier release packets. At barrier release stage, barrier release packets are routed and replicated depending on the recorded incoming port information. Hence, the multi-destination header is not required and wiring recourses are saved. In our design, we use a set of Port Status Registers (PRSs) (See Fig. 1 d) to record barrier acquire packets' incoming ports. For instance, when a barrier acquire packet aims for #1 barrier and goes into a router via the east inport, the 'E' field of #1 PRS is set. Fig. 2 b shows an example of the multicasting action on the 4×4 mesh. For instance, the 'L' and 'S' fields of #0 PRS

DOI: 10.1587/elex.8.1856
Received August 04, 2011
Accepted October 15, 2011
Published November 25, 2011

in router $(3,2)$ were set. Therefore, when a barrier release packet for #0 barrier comes from router $(2,2)$ to router $(3,2)$, it's duplicated to generate two copies. One is sent to the local node via the local outport and the other is sent to router $(4,2)$ via the south outport.

2.3 Cooperative Communicator (CC)

To realize the cooperative gather and multicast, the router is enhanced with a *Cooperative Communicator* (CC). The CC consists of six functional units: five *Acquire Mergers* (AMs) (see Fig. 1 c) and a *Release Multicaster* (RM) (see Fig. 1 d).

For each outport, there is an AM. As shown in Fig. 1 c, the AM is responsible for checking incoming barrier acquire packets and those barrier acquire packets that has been stored in the output buffer and merging the barrier acquire packets that aim for the same barrier counter into one barrier acquire packet. Its function contains three steps. (1) The Classifier groups incoming barrier acquire packets into several groups according to their barrier *id*s. Incoming barrier acquire packets may be classified into up to 5 groups, since they may aim for 5 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.

As depicted in Fig. 1 d, the RM is responsible for establishing the multicast path at barrier acquire stage and replicating and distributing barrier release packets at barrier release stage. At barrier acquire stage, when a barrier acquire packet with barrier *id* comes from port p ($p \in \{L, E, S, W, N\}$), the corresponding p field in the #*id* PRS is set. After barrier acquiring, all PRSs in routers provide enough information to construct the path for multicasting barrier release packets. At barrier release stage, when a barrier release packet is coming, the RM replicates and distributes the barrier release packet depending on the recorded incoming port information in PRSs. The Buffer Maintainer takes charge of putting the generated barrier release packets into their corresponding output buffers. Meanwhile, the corresponding records are removed out of PRSs. There are 5 copiers, one for replicating the barrier release packets from each port.

The CC design is synthesized under TSMC® 65 nm process. It consumes 10.98k NAND gates (10 PRSs) and runs at 2.44 GHz (0.41 ns).

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

DOI: 10.1587/elex.8.1856
Received August 04, 2011
Accepted October 15, 2011
Published November 25, 2011

3 Experiments and results

3.1 Experimental setup

We compare our approach (master-slave algorithm with cooperative communication, denoted by MS+CC) with the four algorithm-based mechanisms with unicast communication, namely, master-slave algorithm with unicast (MS+Un), all-to-all algorithm with unicast (A2A+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 ($M = N = 16$). All nodes participate in the barrier synchronization and the master node is a network center ($\lceil M/2 \rceil, \lceil N/2 \rceil$).

3.2 Synthetic experiments

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, and the completion time is measured from the starting till then. Fig. 3 a–c plots the completion time of barrier synchronization versus the network size. We can see that:

- For all network sizes, our approach (MS+CC) achieves minimal completion time. Due to specialized gather and multicast communication, there is no contention incurred in the network. As a consequence, the completion time can be theoretically determined as $(\lfloor M/2 \rfloor + \lfloor N/2 \rfloor + 2) \times 2 + max\{D_i - D_j\}$.
- As the network size increases, the completion time of MS+CC increases very slowly, while that of A2A+Un and MS+Un increases quickly due to the serialization of barrier acquiring and releasing. A2A+Un shows better performance than MS+Un and Tree+Un for networks of small sizes, but it does not scale well due to quadratically increased number of packets $(MN)^2$.
- 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 4 times worse than MS+CC respectively 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 with unicast, Butterfly+Un shows the best performance and scalability. Still, our MS+CC is outstanding and it reduces the completion time of Butterfly+Un by 36% on average.

3.3 Application benchmarks

In this set of experiments, a real application, namely, 1024-point 1D DIT FFT, is mapped. Its computational tasks are uniformly mapped onto all nodes. Depending on the mesh size ($M \times N$), each node assumes J/MN

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

DOI: 10.1587/elex.8.1856
Received August 04, 2011
Accepted October 15, 2011
Published November 25, 2011

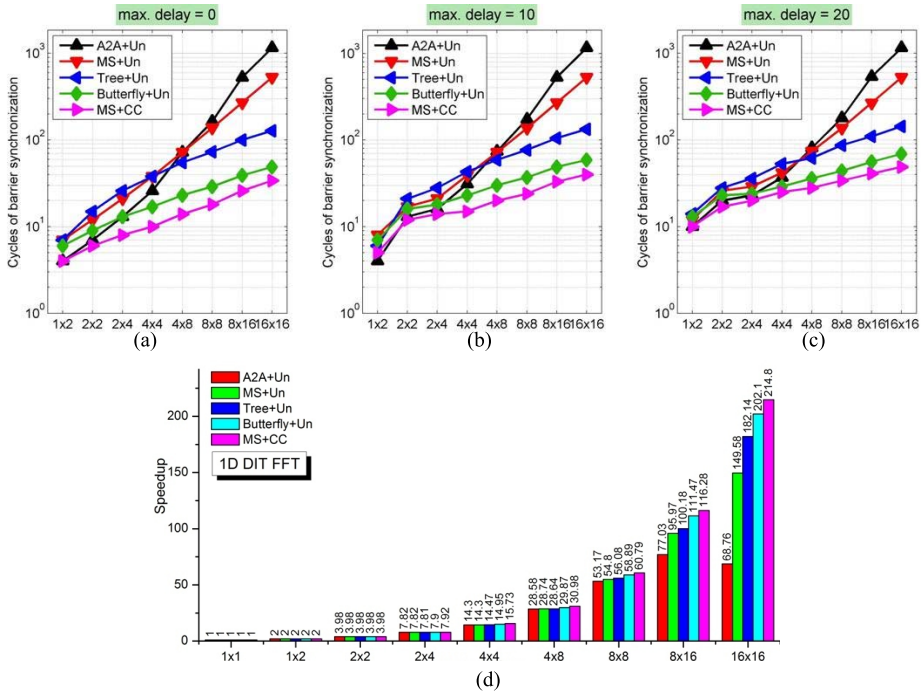

Fig. 3. (a)–(c): Completion time versus network size; (d): Speedup results of 1024-point 1D DIT FFT

tasks, where J is the total number of tasks. For the FFT, J is 1024 and there are 9 times of barrier synchronization.

Fig. 3 d shows the speedup $(\Omega_m)^3$ results, which exhibit the same performance trend as Fig. 3 a. Note that, due to overwhelming contention, A2A+Un's speedup for 16×16 decreases. For 16×16 case, compared with A2A+Un, MS+Un, Tree+Un, and Butterfly+Un, the respective performance improvement is 67.99%, 30.38%, 15.22%, and 5.93%. Since application cases have computation tasks besides synchronization tasks, the improvement is less than that from synthetic experiments.

4 Future work

In the future, we plan to link this cooperative communication strategy with other algorithms. Another direction is to look into the power savings, as our scheme greatly decreases the amount and distance of communication.

Acknowledgment

The research is partially supported by the Major Project of "Core electronic devices, High-end general chip and Fundamental software" in China (No. 2009ZX01034-001-001-006), the National Natural Science Foundation of China (No. 61070036 and No. 61133007), and the National 863 Program of China (No. 2009AA011704).

[3] $\Omega_m = \mathrm{T}_{1\mathrm{node}}/\mathrm{T}_{m\mathrm{node}}$, where $\mathrm{T}_{1\mathrm{node}}$ is the single node execution time as the baseline and $\mathrm{T}_{m\mathrm{node}}$ is the execution time of m node(s).

DOI: 10.1587/elex.8.1856
Received August 04, 2011
Accepted October 15, 2011
Published November 25, 2011