Area and Performance Optimization of Barrier Synchronization on Multi-core Network-on-Chips

Xiaowen Chen[1,2], Shuming Chen[1], Zhonghai Lu[2] and Axel Jantsch[2]

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

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

[1]{xwchen,smchen}@nudt.edu.cn   [2]{xiaowenc,zhonghai,axel}@kth.se

Abstract—**Barrier synchronization is commonly and widely used to synchronize the execution of parallel processor cores on multi-core Network-on-Chips (NoCs). Since its global nature may cause heavy serialization resulting in large performance penalty, barrier synchronization should be carefully designed to have low latency communication and to minimize overall completion time. Therefore, in the paper, we propose a fast barrier synchronization mechanism, targeting Multi-core NoCs. The fast barrier synchronization mechanism includes a dedicated hardware module, named *Fast Barrier Synchronizer (FBS)*, integrated with each processor node. It offers a set of barrier counters and can concurrently process synchronization requests issued by the local node and remote nodes via the on-chip network. The salient feature of our fast barrier synchronization mechanism is that, once the barrier condition is reached, the "barrier release" acknowledgement is routed to all processor nodes in a *broadcast* way in order to save chip area by avoiding storing source node information and to minimize completion time by avoiding serialization of barrier releasing. Synthesis results suggest that the FBS can run over 1 GHz in SMIC® 130nm technology with small area overhead. We implemented a FBS-enhanced multi-core NoC architecture on our FPGA platform using the Xilinx® Virtex 5 as the FPGA chip. FPGA utilization and simulation results show that our fast barrier synchronization demonstrates both area and performance advantages over the barrier synchronization counterpart with *unicast* barrier releasing.**

Keywords-Barrier Synchronization; Multi-core; Network-on-Chips (NoCs)

I. INTRODUCTION

The processor trend shows that the single-chip computing architecture is evolving from single-core to multi- and many cores [1][2]. Network-on-Chip (NoC) [3][4] is recognized as the scalable solution to interconnect and organize so many cores and hence has attracted significant attentions over the last ten years since various buses do not scale well with the system size. Parallelized applications running on multi-core NoC architectures require efficient support for synchronization, since communication is on the critical path of system performance and contended synchronization requests may cause large performance penalty. *Barrier synchronization* is commonly and widely used to synchronize the execution of parallel processor nodes. It requires low latency communication to minimize overall completion time. If barrier synchronization does not carefully designed, it may require thousands of cycles to be completed when hundreds of processors are involved, since its global nature may cause heavy serialization resulting in

bad performance. Therefore, in the paper, we are motivated to propose a fast barrier synchronization mechanism, targeting Multi-core Network-on-Chips (NoCs).

The fast barrier synchronization mechanism hosts a dedicated hardware module, named *Fast Barrier Synchronizer (FBS)*, integrated with each node on the Multi-core NoCs. The FBS provides a set of barrier counters in order to synchronize the execution of parallel processor cores. The FBS could concurrently process synchronization requests issued by the local node and remote nodes via the on-chip network. The salient feature of our fast barrier synchronization mechanism is that, considering the limited chip area, the "barrier release" acknowledgement is routed to all processor nodes in a *broadcast* way when the barrier condition is reached so as to save chip area by avoiding storing the information of source nodes that send barrier requests and to minimize completion time by avoiding serialization of *unicast* barrier releasing.

To evaluate our fast barrier synchronization mechanism, we investigate the FBS's hardware cost and the latency of successful barrier synchronization with it. We first implemented the FBS in synthesizable Verilog and synthesized it using Synopsys® Design Compiler under SMIC® 130nm technology. The area cost and clock frequency for different number of barrier counters are obtained. The synthesis results show that the FBS can run over 1 GHz with small area overhead and it has higher frequency and lower area cost than the barrier synchronization counterpart that sends "barrier release" acknowledgements in a traditional *unicast* way. Further, we construct a multi-core NoC architecture including the FBSs and the on-chip network. The network size is configurable. We implemented the multi-core NoC on our FPGA platform using the Xilinx® **Virtex 5** [5] as the FPGA chip. The FPGA utilization on both registers and LUTs demonstrates the area advantage of the multi-core NoC with our fast barrier synchronization mechanisms over the multi-core NoC with the barrier synchronization counterpart. Finally, the synthetic workloads are preformed on the FPGA platform. The results show that, as the network size is scaled up, the latency of successful synchronization of our fast barrier synchronization is lower than that of the counterpart.

The rest of the paper is organized as follows. Section II discusses the related work. In Section III, we introduce architectures of the multi-core NoC and the Fast Barrier Synchronizer. Then, we detail our fast barrier synchronization mechanism in Section IV. We evaluate the mechanism by reporting area and performance results in Section V. Finally we conclude in Section VI.
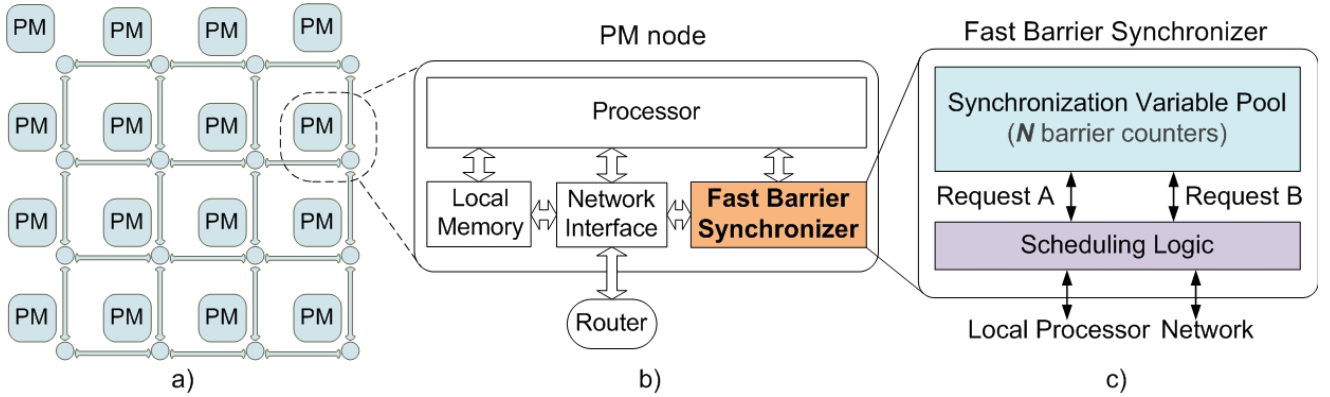
Fig. 1. a) A 16-node mesh multi-core NoC, b) Processor-Memory node, and c) Fast Barrier Synchronizer

II. RELATED WORK

Synchronization attracted a large body of research in multiprocessor systems [6][7]. Barrier synchronization is commonly and widely used to synchronize the execution of parallel processor nodes. the most straightforward barrier synchronization algorithms are the centralized barriers. In a centralized barrier, every processor increments a global counter and then waits for all the processors to arrive at the barrier. To avoid serialization at the barrier counter, a tournament barrier uses a binary tree [8]. Every processor starts at a separate leaf. At every node, one of the child processors moves up the tree after both the child processors have arrived. The barrier completion event is propagated down the tree. Compared with it, the barrier synchronization in this paper uses a packet-switched 2D mesh network as its propagation topology, since our target architecture is multi-core NoCs. In the field of chip multiprocessors (CMPs), Liu demonstrated the importance and the impact (in terms of performance overhead and power consumption) of barrier synchronization [9], while Marongiu discusses the use of a run-time lightweight barrier construct in non cache coherent MPSoC platforms [10]. In [11], Sampson presented a mechanism for barrier synchronization on CMPs. This mechanism starves threads and thus forces their execution to stop by forcing the invalidation of selected I-cache lines. Threads are let free when all have entered the barrier. In [12], Monchiero proposes a centralized HW solution to perform barrier synchronization in embedded systems. However, since all synchronization requests issued by all nodes flow through the network into the centralized synchronization module, contention latency induced by heavy traffic near the module brings negative performance. And his system size is limited by 8 cores. In the paper, we investigate fast barrier synchronization support on multiprocessor systems with the system size scaling up to 64 cores.

III. MULTI-CORE NETWORK-ON-CHIPS WITH FAST BARRIER SYNCHRONIZATION

Fig. 1 a) shows an example of our multi-core NoC architecture. The system is composed of 16 Processor-Memory (PM) nodes interconnected via a packet-switched network. The network topology is a mesh, which is a most popular NoC topology proposed today [13]. As shown in Fig. 1 b), each PM node contains a processor, a local memory, a Network Interface (NI) and a *Fast Barrier Synchronizer (FBS)*. All local memories can logically form a single global memory address space. The FBS is connected to the processor and the Network Interface (NI) and provides efficient barrier synchronization support. As shown in Fig. 1 c), the FBS is mainly composed of a *Synchronization Variable Pool* and a *Scheduling Logic*. The Synchronization Variable Pool contains N barrier counters with 8 bits each. These barrier counters are globally addressed and visible to all PM nodes. Each barrier counter can support at most 512 ($=2^8$) paralleled processes or threads. There are two access ports for these barrier counters. The Scheduling Logic serves synchronization requests from the local processor and remote processors via the on-chip network concurrently, and sends Request A and B to the Synchronization Variable Pool to increment/decrement the barrier counters. It is responsible for guaranteeing the correctness of handling barrier synchronization as well as maintaining a fast barrier synchronization.

IV. FAST BARRIER SYNCHRONIZATION

In barrier synchronization, every synchronized PM node sends a "barrier acquire" request where the barrier counter is in order to increment the barrier counter and then waits for all synchronized PM nodes' arrival at that barrier. The destination SH, which hosts the barrier counter, can concurrently respond to the "barrier acquire" requests from both the local PM node and the remote PM nodes via the on-chip network. If only one request from either the local PM node or the remote PM node comes, the barrier counter is incremented by 1. If both the local and the remote request come, the barrier counter is incremented by 2. Once the last synchronized PM node arrives at the barrier, a "barrier release" acknowledgment is sent back to let free all synchronized PM node. Since barriers can be used more than once in a PM node, in order to avoid that a PM node enters the barrier for a second time before previous PM nodes have left the barrier for the first time, a two phases mechanism is used, as shown in Fig. 2 a) and b): (1) *Arrival Phase*: Every
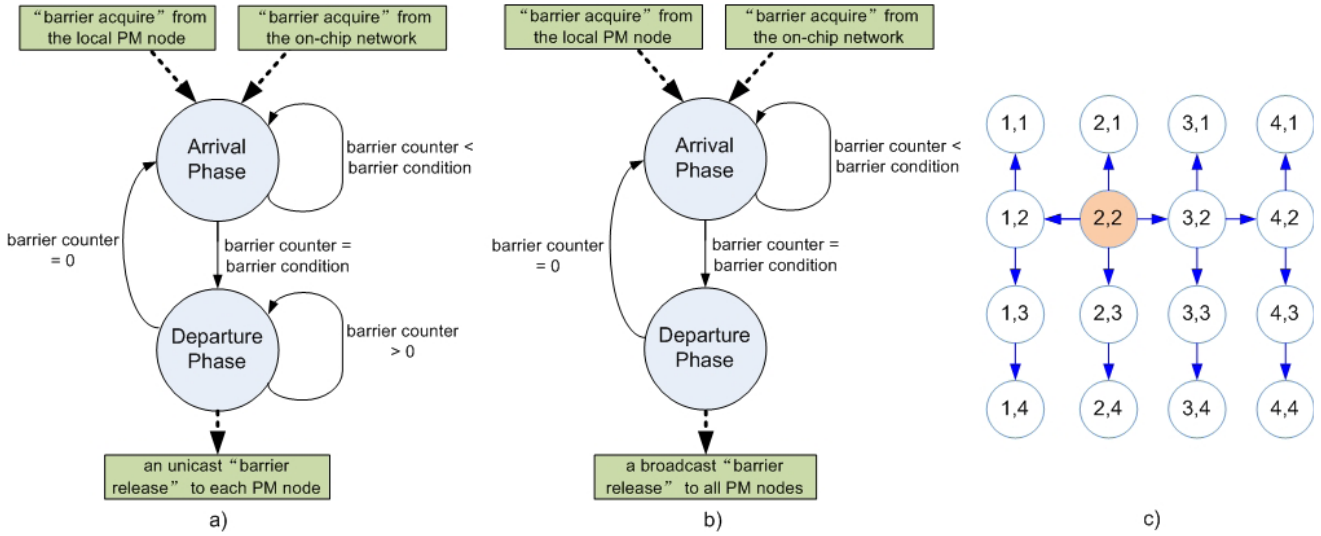
Fig. 2. a) state transition of the barrier synchronization counterpart with *unicast* barrier releasing, b) state transition of our barrier synchronization with *broadcast* barrier releasing, and c) a short example of how a "barrier release" acknowledgement propagates over the network

PM node enters this phase and does not leave it until all PM nodes have arrived at the same phase, (2) *Departure Phase:* When every PM node finishes the Arrival Phase, a "barrier release" acknowledgement is given.

In order to minimize serialization of barrier releasing, we adopt a *broadcast* way to let free all synchronized PM node once the barrier condition is reached. We compare it with the barrier mechanism counterpart that stores the node No. information of all synchronized PM nodes and sends "barrier release" acknowledgements to all synchronized PM nodes one by one in an *unicast* way.

Fig. 2 a) shows the state transition of the barrier synchronization counterpart with *unicast* barrier releasing. As we can see, In *Arrive Phase*, when a "barrier acquire" request comes, the barrier counter is incremented by 1 and the source PM node number is stored. If two "barrier acquire" requests come simultaneously, the barrier counter is incremented by 2 and the two source PM node numbers are stored. Once the last synchronized PM node arrives and the barrier condition is reached, the state changes to *Departure Phase*. In *Departure Phase*, "barrier release" acknowledgements are sent back to each source PM node one by one in an *unicast* (i.e. point-to-point) way according to the stored source PM node numbers. Once a "barrier release" acknowledgment is sent out, the barrier counter is decremented by 1. When the barrier counter equals zero, the state returns to *Arrival Phase*. In multi-core NoC architecture, this *unicast* way mainly has two shortcomings: (i) All synchronized PM nodes' numbers need to be stored. This leads to larger area especially when the number of total synchronized PM nodes is increasing. (ii) In *Departure Phase*, "barrier release" acknowledgements are serialized and there is an acknowledgment for each synchronized source PM node. The serialization of acknowledgments results in bad performance. The increase of the number of synchronized PM nodes leads to the increase of message

number of "barrier release" acknowledgements. Hence, the local hotspot is formed and the network contention becomes heavy near the destination PM node that hosts the barrier counter.

To overcome the two shortcomings, we propose the barrier mechanism shown in Fig. 2 b). In *Arrival Phase*, once one or two "barrier acquire" requests come, the barrier counter is incremented by 1 or 2. However, diffident with the barrier synchronization counterpart in Fig. 2 a), the source PM node number doesn't need to be stored. Once the last synchronized PM node arrives and the barrier condition is reached, the state changes to *Departure Phase*. The *Departure Phase* only takes 1 clock cycle and accomplishes two things: (1) a "barrier release" acknowledgement is sent out in a broadcast way. The "barrier release" acknowledgement propagates over the on-chip network and is routed to all PM nodes. After receiving the acknowledgement, synchronized source PM nodes are let free to continue their execution. Other PM nodes without participating in the barrier synchronization could also receive this acknowledgment; they just ignore and discard it. (2) the barrier counter is set to be zero. After these two things, the state returns to *Arrival Phase*. In comparison with the barrier mechanism in Fig. 2 a), our barrier mechanism has two advantages: (i) Since the source synchronized PM nodes' No. information are not stored, the area cost is low. (ii) There is only a *broadcast* "barrier release" acknowledgement, so the serialization of barrier releasing is avoided and the efficiency is improved. Although the PM nodes that don't attend the barrier synchronization could receive the *broadcast* "barrier release" acknowledgement, they just only receive one unused message, causing 1 clock cycle delay. The destination PM node hosting the barrier counter only injects one message of "barrier release" acknowledgement into the network and hence will not leads to local hotspot near the destination PM node.
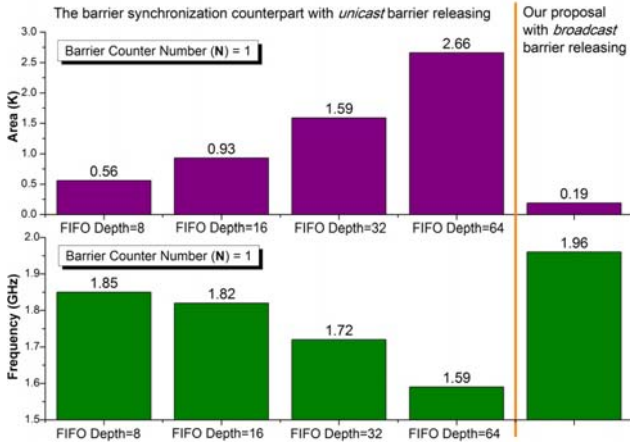
Fig. 3. Area and Frequency comparison between our fast barrier synchronization with *broadcast* barrier releasing and the barrier synchronization counterpart with *unicast* barrier releasing

Fig. 4. Area and Frequency of our fast barrier synchronization with *broadcast* barrier releasing for different number of barrier counters

Fig. 2 c) shows an example of how a "barrier release" acknowledgement propagates over the on-chip network. Assume that Node(2,2) sends out the "barrier release" acknowledgement in a *broadcast* way, the acknowledgement propagates firstly along the left and right directions and secondly along the top and bottom directions to arrive at every PM nodes.

V. PERFORMANCE EVALUATION

In this section, we construct a FPGA version of our multi-core NoC architecture as shown in Fig. 1. In the FPGA version, the PM node includes the FBS and the Network Interface rather than the Local Memory. The processor in each PM node is replaced by a barrier synchronization request generator that imitates processor's behavior to issue requests. The network performs dimension-order XY routing, provides best-effort service, and guarantees in-order packet delivery. Moving one hop in the network takes one cycle. Besides, the network size is configurable.

To evaluate our fast barrier synchronization mechanism, we investigate the hardware cost and the latency of successful barrier synchronization. In comparison with the barrier synchronization counterpart with *unicast* barrier releasing, our fast barrier synchronization mechanism demonstrates both area and performance advantages.

A. Hardware Cost

We first implemented the FBS in synthesizable Verilog and synthesized it using Synopsys® Design Compiler under SMIC® 130nm technology. The area cost and clock frequency for different number of barrier counters are obtained. Further, we implemented the multi-core NoC architecture on our FPGA platform using a Xilinx® **Virtex 5** [5] as the FPGA chip. The FPGA register and LUT utilizations of the multi-core NoC with different network size are obtained.

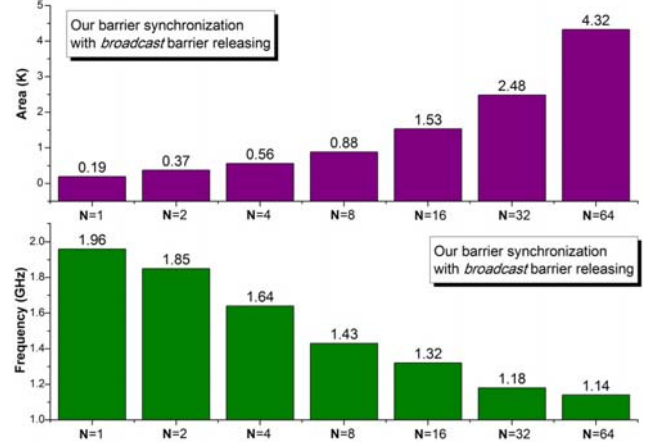In order to compare the barrier synchronization counterpart with *unicast* barrier releasing, we assume that the barrier number (**N**) is 1 and the FIFO depth of the barrier synchronization counterpart varies from 8, 16, 32 to 64. Fig. 3 shows the area and frequency comparison between our fast barrier synchronization and the barrier synchronization counterpart. Our fast barrier synchronization doesn't need to save source node information, while the barrier synchronization counterpart has to host a FIFO saving source node information. Hence, as shown in the figure, the FBS has higher clock frequency and lower area cost than the barrier synchronization counterpart. The FBS in our fast barrier synchronization could run 1.96 GHz with the area overhead of 0.19K. The barrier synchronization counterpart consumes more area and has lower frequency as the increase of the FIFO depth. The FIFO depth limits the maximal number of barrier synchronized PM nodes. Although the barrier synchronization counterpart could reach 1.85 GHz frequency and 0.56K area, the maximal number of barrier synchronized PM nodes is only 8. For the sake of supporting more barrier-synchronized PM nodes, the FIFO depth has to be increased, consuming more area and resulting in lower frequency. In contrast, since our fast synchronization mechanism broadcasts the "barrier release" acknowledgments without FIFOs storing source node information, it features both area and frequency advantages over the counterpart. To evaluate the hardware cost, Fig. 4 shows the area cost and clock frequency of the FBS in our fast synchronization mechanism with the barrier counter number (N) varying from 1, 2, 4, 8, 16, 32 to 64.We can see that, as the barrier counter number (**N**) increases, the area cost increases from 0.19K to 4.32K and the clock frequency is always over 1 GHz.

Fig. 5 shows the register and LUT utilization comparison between the multi-core NoC with our fast barrier synchronization and that with the barrier synchronization counterpart versus the network size from 1x2(2), 2x2(4), 2x4(8), 4x4(16), 4x8(32) to 8x8(64). Assume that the barrier counter number (**N**) is 1 and the FIFO depth of the barrier synchronization counterpart is 64. As we can see, the multi-
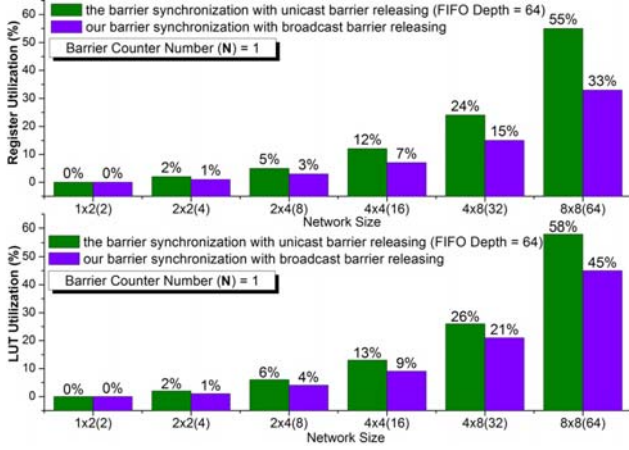
Fig. 5. Register and LUT utilization comparison between our fast barrier synchronization with *broadcast* barrier releasing and the barrier synchronization counterpart with *unicast* barrier releasing

Fig. 6. Register and LUT utilization of our fast barrier synchronization with *broadcast* barrier releasing for different network size

core NoC with our fast barrier synchronization consumes less FPGA utilization on both registers and LUTs. The advantage becomes obvious with the increase of the network size. For instance, when the network is 1x2(2), the two multi-core NoCs both take very small FPGA resources, approximately equaling 0%. When the network size is scaled up to 8x8(64), the multi-core NoC with the barrier synchronization counterpart consumes 55% register utilization and 58% LUT utilization, while the multi-core NoC with our fast barrier synchronization only consumes 33% register utilization and 45% LUT utilization. With the same FPGA, our solution could support larger-scale multi-core NoC systems than the counterpart. Fig. 6 illustrates the FPGA utilization of the multi-core NoC with our fast barrier synchronization under different barrier counter number (N) and different network size. We can see that, as the barrier counter number (N) increases and the network size is scaled up, the FPGA utilization increases from approx. 0% to 58% for registers and from approx. 0% to 61% for LUTs.

B. Cost of Sucessful Synchronization

We implemented the FBS-enhanced multi-core architecture on our FPGA platform using a Xilinx® **Virtex 5** [5] as the FPGA chip and performed synthetic workloads to evaluate our fast barrier synchronization mechanism. In the experiments, a barrier counter in the central PM node is chosen. When the simulation starts, each synchronized PM node sends a "barrier acquire" request to the FBS in the central PM node. Once all synchronized PM nodes arrive at the barrier, a "barrier release" acknowledgement is returned from the central PM node. The simulation ends after all synchronized PM nodes receive the "barrier release" acknowledgement. The cycles of successful synchronization for both the barrier synchronization counterpart with barrier releasing in an *unicast* way and our fast barrier synchronization with barrier releasing in a *broadcast* way are obtained and analyzed. The experimental results show that our barrier synchronization demonstrates performance advantage over the barrier synchronization counterpart, espe-
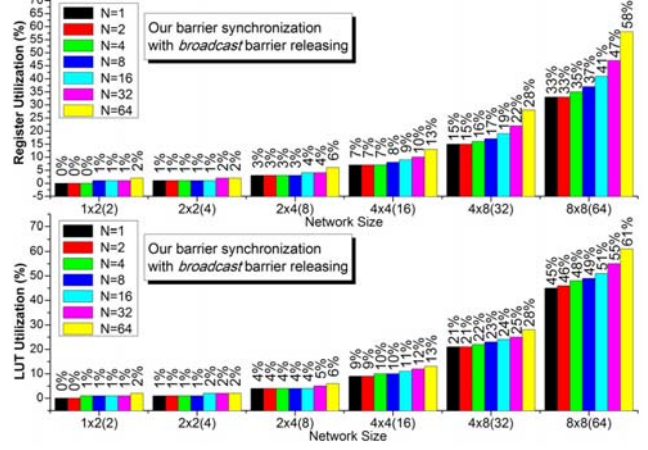
cially when the network size is scaled up and the number of synchronized PM nodes participating the barrier synchronization increases.

Case 1: the network size is varied and all PM nodes participate the barrier synchronization.

Fig. 7 plots the cycles of accomplishing a barrier synchronization with the network size varying from 1x2(2), 2x2(4), 2x4(8), 4x4(16), 4x8(32) to 8x8(64). From the figure, we can see that (i) as the network size is scaled up, the latency of barrier synchronization is increasing for both our barrier synchronization and the barrier synchronization counterpart. This is because network communication delay increases. (ii) Our barrier synchronization obtains better performance than the barrier synchronization counterpart, especially when the network size increases. This is because the number of synchronized PM nodes participating the barrier synchronization increases as the network size increases. In the barrier synchronization counterpart, the serialization of barrier releasing consumes more time, enlarging negative performance. However, our fast barrier synchronization mechanism eliminates the serialization, resulting in better performance.

Case 2: the network size is fixed to be 8x8 and the number of synchronized PM nodes that participate the barrier synchronization is varied.

Fig. 8 plots the cycles of accomplishing a barrier synchronization versus the ratio of synchronized PM nodes in barrier synchronization to all PM nodes. 1/32 means there are 2 (=64/32) PM nodes participating the barrier synchronization, while 1 means all (64) PM nodes participate the barrier synchronization. From the figure, we can have that (i) as the number of synchronized PM nodes increase, the latency of barrier synchronization increases, this is because more time is needed to finish the barrier synchronization, and (ii) when more PM nodes participate the barrier synchronization, the serialization of barrier releasing in the barrier synchronization counterpart becomes worse and hence results in larger latency than our barrier synchronization. In contrast, our fast barrier synchronization
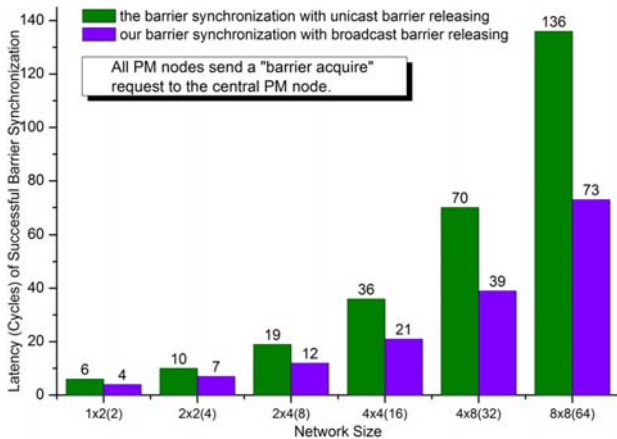
Fig. 7. Case 1: cycles of of accomplishing a barrier synchronization during a simulation

Fig. 8. Case 2: cycles of of accomplishing a barrier synchronization during a simulation

mechanism obtains better performance.

VI. CONCLUDING REMARK

Parallelized applications running on multi-core Network-on- Chips (NoCs) require efficient support for synchronization, since communication is on the critical path of system performance. Barrier synchronization should be carefully designed to feature low latency communication and minimal overall completion time. In the paper, we propose a fast barrier synchronization mechanism, targeting multi-core NoCs. The fast barrier synchronization mechanism includes a dedicated hardware module, named *Fast Barrier Synchronizer (FBS)*, integrated with each processor node. It offers a set of barrier counters and can concurrently process synchronization requests issued by the local node and remote nodes via the on-chip network. In order to save chip area cost by avoiding storing source node information and to minimize completion time by avoiding serialization of barrier releasing, the "barrier release" acknowledgement is designed to be routed to all processor nodes in a broadcast way as soon as the barrier condition is reached. The FBS is synthesized in SMIC® 130nm technology. We also implemented a FBS-enhanced multi-core NoC architecture on our FPGA platform using the Xilinx® **Virtex 5** as the FPGA chip. Synthesis results, FPGA utilizations and simulation reports show that our fast barrier synchronization demonstrates both area and performance advantages over the barrier synchronization counterpart with *unicast* barrier releasing. In the future, we will investigate the power savings with our fast barrier synchronization.

REFERANCE

[1] M. Horowitz and W. Dally, "How scaling will change processor architecture," in Int'l Solid-State Circuits Conf. (ISSCC'04), Digest of Technical Papers, Feb. 2004, pp. 132–133.

[2] S. Borkar, "Thousand core chips: A technology perspective," in Proc. of the 44th Design Automation Conf. (DAC'07), Jun. 2007, pp. 746–749.

[3] A. Jantsch and H. Tenhunen, Networks on chip. Kluwer Academic Publishers, 2003.

[4] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," ACM Comp. Surveys, vol. 38, no. 1, pp. 1–51, Mar. 2006.

[5] Virtex 5 xc5vlx330. [Online]. Available: http://www.xilinx.com/ support/documentation/data sheets/ds100.pdf

[6] N.-F. Tzeng and A. Kongmunvattana, "Distributed shared memory systems with improved barrier synchronization and data transfer," in Proc. of the 11th Int'l Conf. on Supercomputing (ICS'97), 1997, pp. 148–155.

[7] S. Kumar, D. Jiang, R. Chandra, and J. Singh, "Evaluating synchronization on shared address space multiprocessors: methodology and performance," in Proc. of the Int'l Conf. on Measurement and modeling of computer systems (SIGMETRICS'99), 1999, pp. 23–34.

[8] 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.

[9] C. Liu, A. Sivasubramaniam, M. Kandemir, and M. J. Irwin, "Exploiting barriers to optimize power consumption of cmps," in Proc. of th 19th IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS'05), 2005.

[10] A. Marongiu, L. Benini, , and M. K. Lightweight, "Lightweight barrierbased parallelization support for non-cache-coherent mpsoc platforms," in Proc. of th ACM Int'l Conf. on Compilers, architecture, and synthesis for embedded systems (CASES'05), 2007, pp. 145–149.

[11] J. Sampson, R. Gonzalez, J.-F. Collard, N. Jouppi, and M. Schlansker, "Fast synchronization for chip multiprocessors," ACM Computer Architecture News, vol. 33, no. 4, pp. 64–69, Apr. 2005.

[12] M. Monchiero, G. Palermo, C. Silvano, and O. Villa, "Efficient synchronization for embedded on-chip multiprocessors," IEEE Trans. on VLSI, vol. 14, no. 10, pp. 1049–1062, Oct. 2006.

[13] P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," IEEE Trans. on Computers, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.